Introduction to the Unified Modeling Language UML

Jonathan I. Maletic, Ph.D.
Department of Computer Science
Kent State University

UML Part I

- Introduction to UML
- Overview and Background

Objectives of UML

- UML is a general purpose notation that is used to
    - visualize,
    - specify,
    - construct, and
    - document

  the artifacts of a software systems.

Background

- UML is the result of an effort to simplify and consolidate the large number of OO development methods and notations
- Main groups: Booch [91], Rumbaugh [91], Jacobson [92]
- Object Management Group – www.omg.org

Types of Diagrams

- Structural Diagrams – focus on static aspects of the software system
  - Class, Object, Component, Deployment
- Behavioral Diagrams – focus on dynamic aspects of the software system
  - Use-case, Interaction, State Chart, Activity

Structural Diagrams

- **Class Diagram** – set of classes and their relationships. Describes interface to the class (set of operations describing services)
- **Object Diagram** – set of objects (class instances) and their relationships
- **Component Diagram** – logical groupings of elements and their relationships
- **Deployment Diagram** - set of computational resources (nodes) that host each component.

Behavioral Diagram

- **Use Case Diagram** – high-level behaviors of the system, user goals, external entities: actors
- **Sequence Diagram** – focus on time ordering of messages
- **Collaboration Diagram** – focus on structural organization of objects and messages
- **State Chart Diagram** – event driven state changes of system
- **Activity Diagram** – flow of control between activities

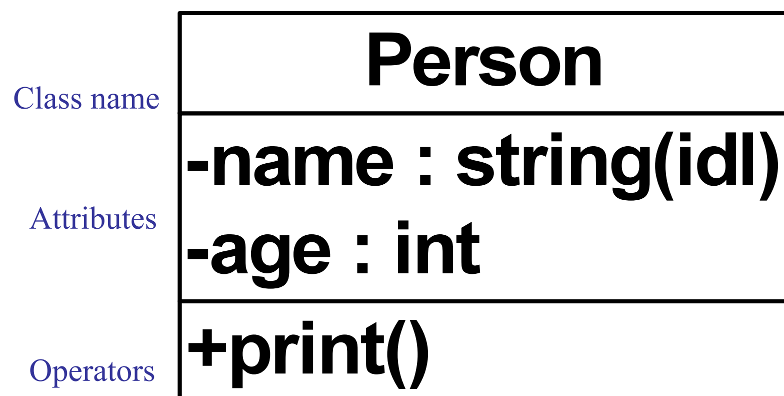
Development Process

- Requirements elicitation – High level capture of user/ system requirements
  - Use Case Diagram
- Identify major objects and relationships
  - Object and class diagrams
- Create scenarios of usage
  - Class, Sequence and Collaboration diagrams
- Generalize scenarios to describe behavior
  - Class, State and Activity Diagrams
- Refine and add implementation details
  - Component and Deployment Diagrams

UML Part II

- Class Diagrams

A Class in UML

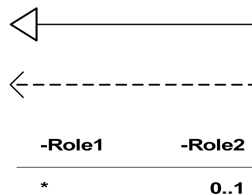


An Object in UML

object name
and class

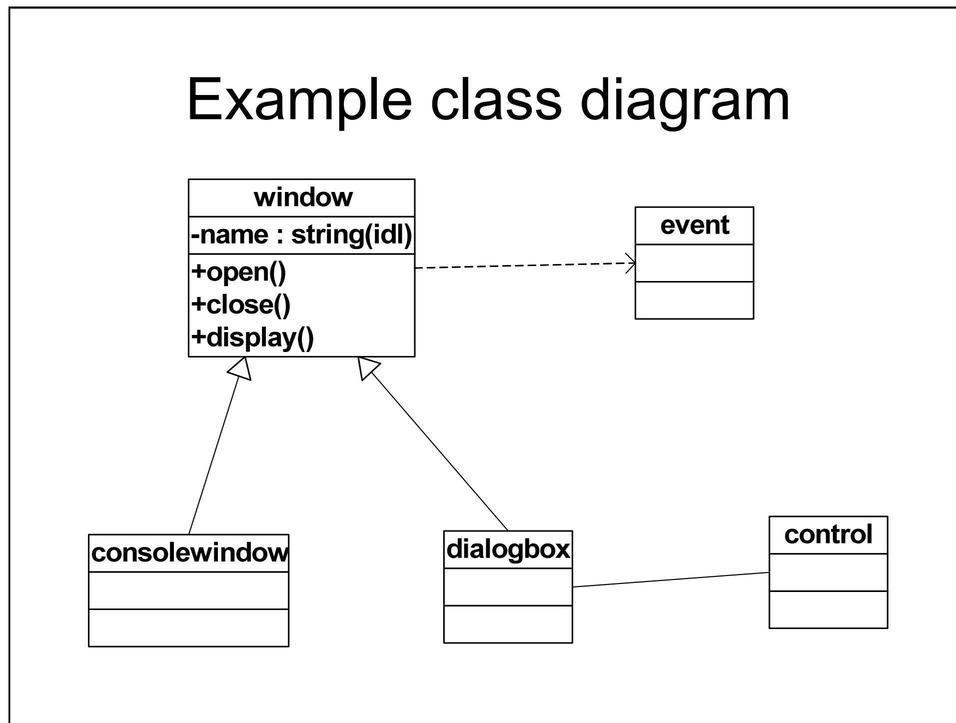
Class Relationships in UML

- Generalization
- Dependency
- Association



- These can represent inheritance, using, aggregation, etc.

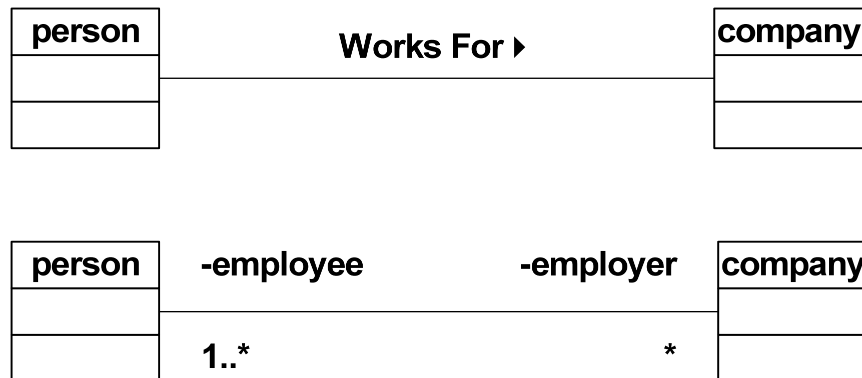
Example class diagram



Association

- Structural relationship between peer classes (or objects).
- Association can have a name and direction, or be bi-directional
- Role names for each end of the association
- Multiplicity of the relationship

Examples of Association



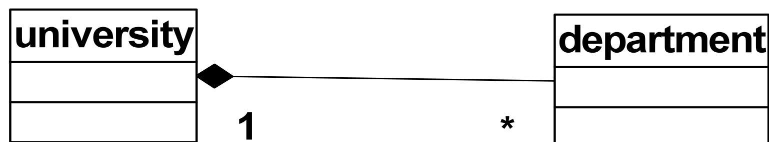
Association code example

```
class Person {
public:
private:
  Company *employer;
};

class Company {
public:
private:
  Person **employee;
};
```

- Each instance of Person has a pointer to its employer
- Each instance of Company has a collection of pointers denoting its employees

Aggregation

- Special type of association
- Part of relationship
- Can use roles and multiplicity



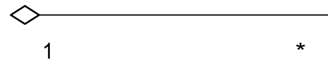
Aggregation code example

- A part-of (has-a) relationship (physical containment)

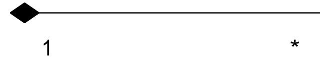
```
class university {
public:
  university();
  university(vector<*department>);

private:
  vector<*department> dept[n];
};
```

Aggregation vs Composition

- Aggregation is a shared containment. No strong life time dependency.



- Composition is constrained aggregation that implies ownership. The life time of the aggregates are determined by the object.

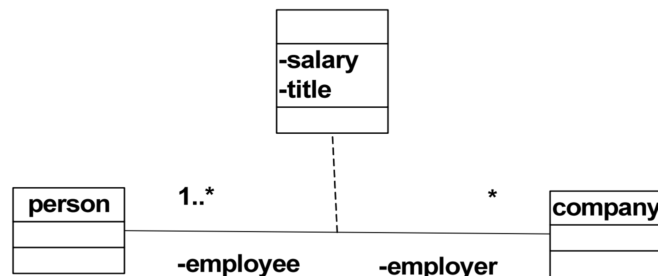


Composition code example

```
//Composition
class car {
public:
  car() {carb = new Carburetor();};

private:
  Carburetor *carb;
};
```

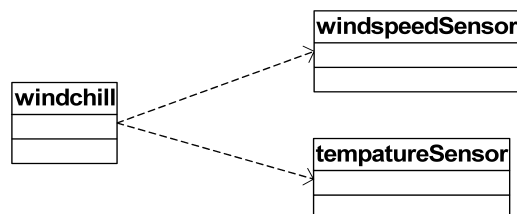
Link Attributes

- Associations may have properties in the same manner as objects/classes.
- Salary and job title can be represented as



Dependency

- Represents a using relationship
- If a change in specification in one class effects another class (but not the other way around) there is a dependency



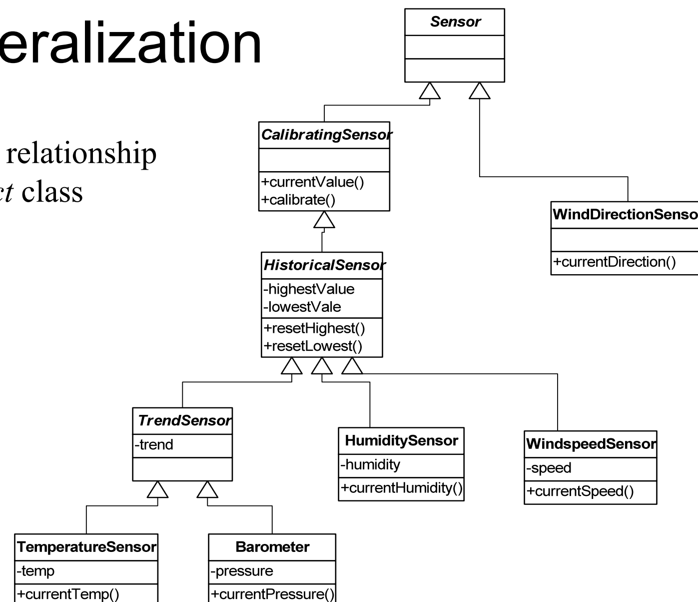
Dependency code example

- A using relationship

```
class windchill {
public:
  windchill();

private:
  static sensor *windspeed; //Part of class, not
  static sensor *temp;      // part of object.
};
```

Generalization

- An is-a relationship
- *Abstract* class



Generalization code example

- An is-a relationship

```
class windDirection : sensor {
public:
  windDirection() : sensor() {};

private:

};
```

Which Relation is Right?

- Aggregation – aka is-part-of, is-made-of, contains
- Use association when specific (persistent) objects have multiple relationships (e.g., there is only one Bill Gates at MS)
- Use dependency when working with static objects, or if there is only one instance
- Do not confuse part-of with is-a

Object Model

- Abstraction – separate behavior from implementation
- Encapsulation – separate interface from implementation
- Modularity – high cohesion and low coupling
- Hierarchy – Inheritance
- Polymorphism – dynamic variable binding
- Typing – strong enforcement
- Concurrency – active vs. inactive
- Persistence – existence transcends runtime

Object Modeling

- Given the high-level requirements (use cases)
- Define the object model
  - Identify objects
  - Compile a data dictionary
  - Identify association and aggregations
  - Identify attributes of objects
  - Generalize objects into classes
  - Organized and abstract using inheritance
  - Iterate and refine model
  - Group classes into modules/components

What is a good Class?

- Should provide a crisp abstraction of something from the problem domain (or solution) domain
- Embody a small well defined set of responsibilities and carry them out well
- Provides clear separation of abstraction, specification, and implementation
- Is understandable and simple yet extendable and adaptable.

Types of Objects

- *Boundary* – represent the interactions between the system and actors
- *Control* – represent the tasks that are performed by the user and supported by the system
- *Entity* – represent the persistent information tracked by the system
- See [Jacobson '99]

Example: Weather Monitoring Station

- This system shall provide automatic monitoring of various weather conditions. Specifically, it must measure:
  - wind speed and direction
  - temperature
  - barometric pressure
  - humidity
- The system shall also proved the following derived measurements:
  - wind chill
  - dew point temperature
  - temperature trend
  - barometric pressure trend

Weather Monitoring System Requirements

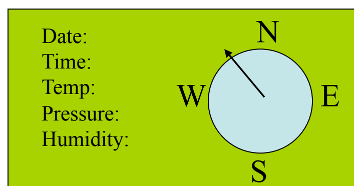
- The system shall have the means of determining the current time and date so that it can report the highest and lowest values for any of the four primary measurements during the previous 24 hour period.
- The system shall have a display that continuously indicates all eight primary and derived measurements, as well as current time and date.
- Through he use of a keypad the user may direct the system to display the 24 hour low or high of any one primary measurement, with the time of the reported value.
- The system shall allow the user to calibrate its sensors against known values, and set the current time and date.

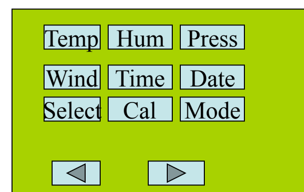
Hardware Requirements

- Use a single board computer (486?)
- Time and date are supplied by an on-board clock accessible via memory mapped I/O
- Temperature, barometric pressure, and humidity are measured by on board circuits with remote sensors.
- Wind direction and speed are measure from a boom encompassing a wind vane (16 directions) and cups (which advance a counter every revolution)
- User input is provided through an off the shelf keypad, managed by onboard circuit supplying audible feed back for each key press.
- Display is off the self LCD with a simple set of graphics primitives.
- An onboard timer interrupts the computer every 1/60 second.

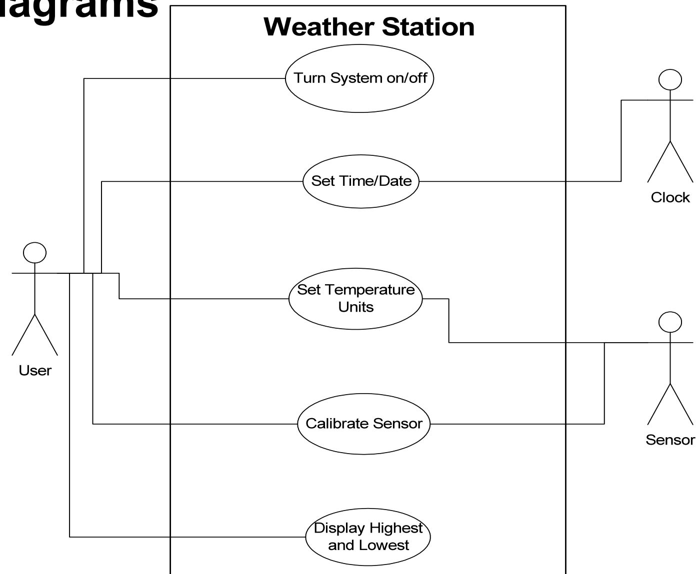
Display and Keypad

- LCDDisplay – Values and current system state (Running, Calibrating, Selecting, Mode)
  - Operations: drawtext, drawline, drawcircle, settextsize, settextstyle, setpensize
- Keypad allows user input and interaction
  - Operations: last key pressed
  - Attributes: key

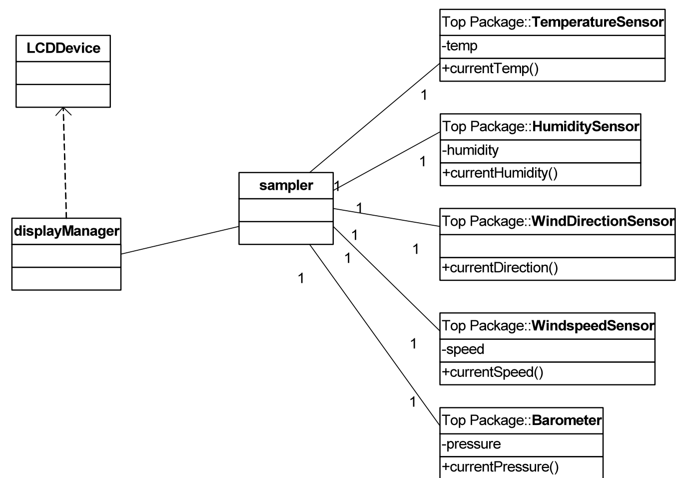




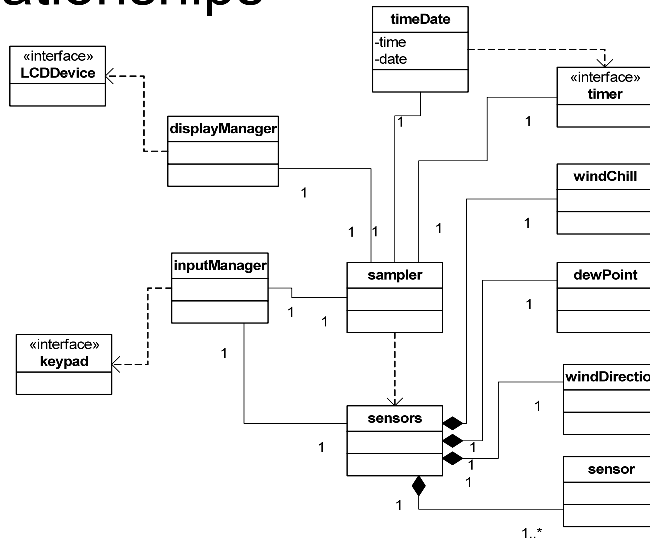
Use Diagrams



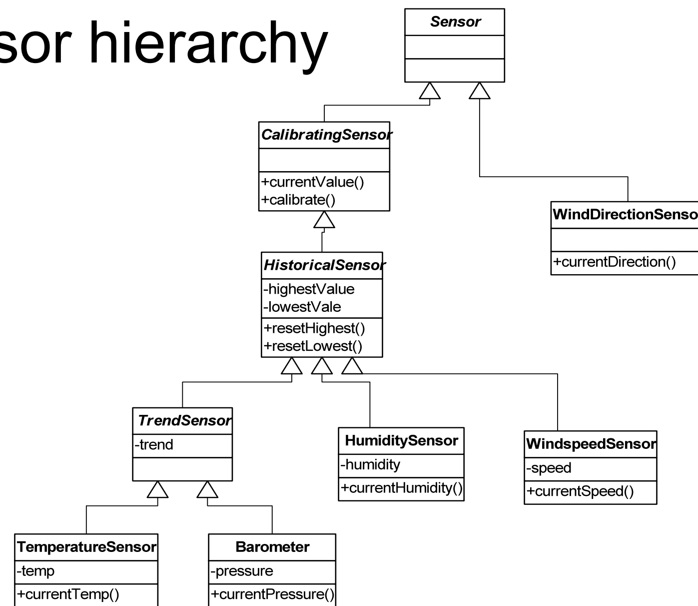
Relationships



Relationships



Sensor hierarchy



UML Part III

- Use Case Diagrams

Use Case Diagrams

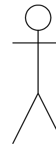
- Describes a set of sequences.
- Each sequence represents the interactions of things outside the system (*actors*) with the system itself (and key abstractions)
- Use cases represent the functional requirements of the system (non-functional requirements must be given elsewhere)

Use case

A Use Case

- Each use case has a descriptive name
- Describes what a system does but not how it does it.
- Use case names must be unique within a given package
- Examples: withdraw money, process loan

Actor



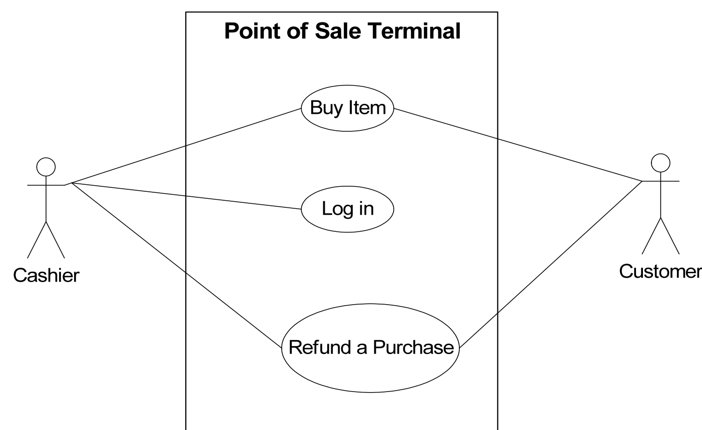
An Actor

- Actors have a name
- An actor is a set of roles that users of use cases play when interacting with the system
- They are external entities
- They may be external an system or DB
- Examples: Customer, Loan officer

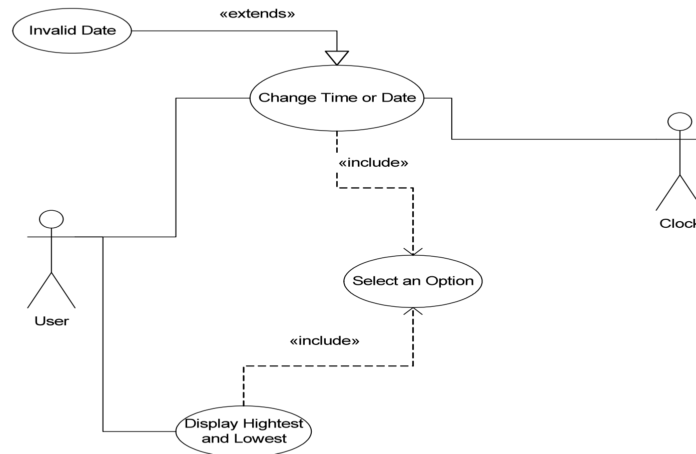
What is a Use Case

- Use case captures some user-visible functionality
- Granularity of functionality depends on the level of detail in your model
- Each use case achieves a discrete goal for the user
- Use Cases are generated through requirements elicitation

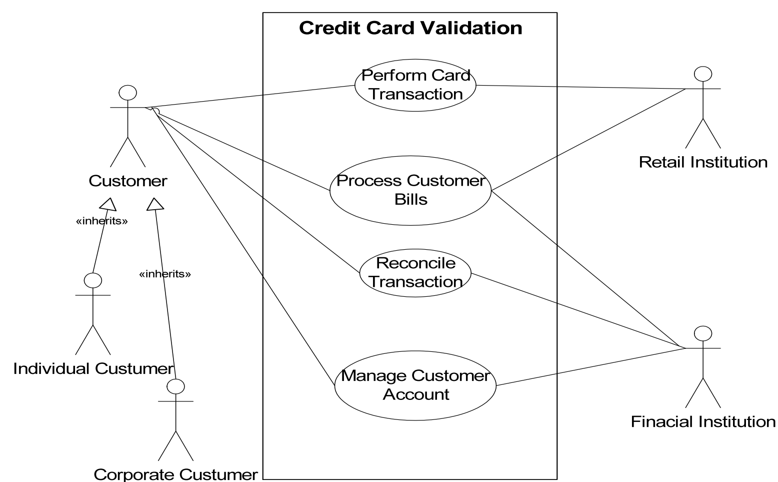
Example



Extend and Include



Example (generalization)



UML Part IV

- Modeling Behavior
- Sequence Diagrams

Refining the Object Model

- Typically, only very simplistic object models can be directly derived from use cases.
- A better understanding of the behavior of each use case is necessary (i.e., analysis)
- Use interaction diagrams to specify and detail the behavior of use cases
- This helps to identify and refine key abstractions and relationships
- Operations, attributes, and messages are also identified during this process

Interaction Diagrams

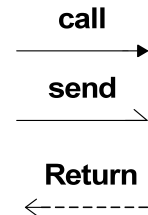
- There is one (or more) Interaction diagram per use case
  - Represent a sequence of interactions
  - Made up of objects, links, and messages
- Sequence diagrams
  - Models flow of control by time ordering
  - Emphasizes passing messages wrt time
  - Shows simple iteration and branching
- Collaboration diagrams
  - Models flow of control by organization
  - Structural relationships among instances in the interaction
  - Shows complex iteration and branching

Sequence Diagrams

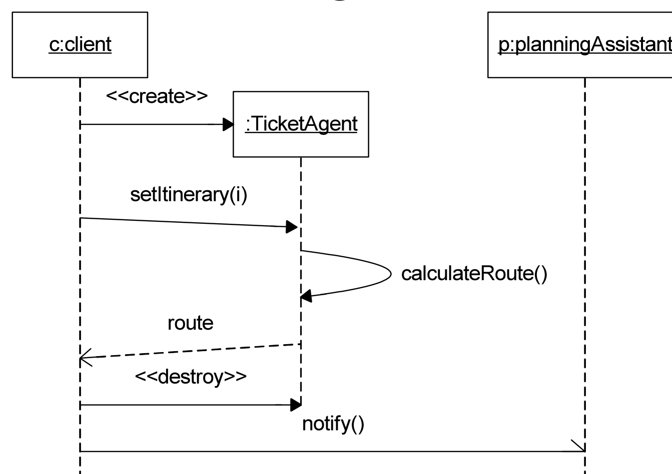
- X-axis is objects
  - Object that initiates interaction is left most
  - Object to the right are increasingly more subordinate
- Y-axis is time
  - Messages sent and received are ordered by time
- Object life lines represent the existence over a period of time
- Activation (double line) is the execution of the procedure.

Message Passing

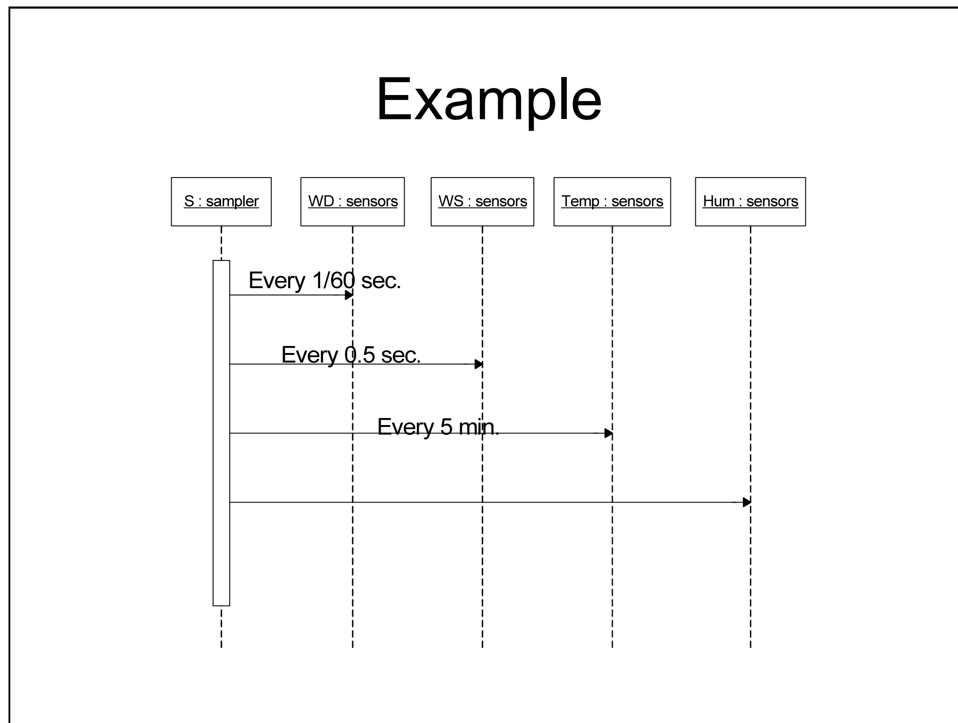
- Send – sends a signal (message) to an object
- Return – returns a value to a caller
- Call – invoke an operation
- Stereotypes
  - <<create>>
  - <<destroy>>

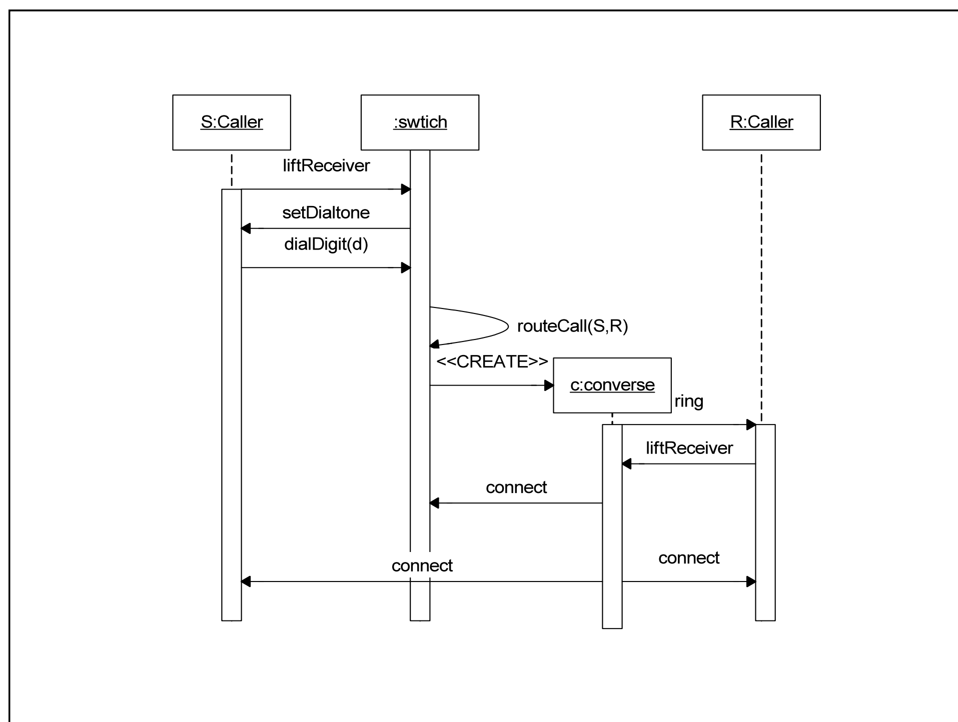


Example UML Sequence Diagram

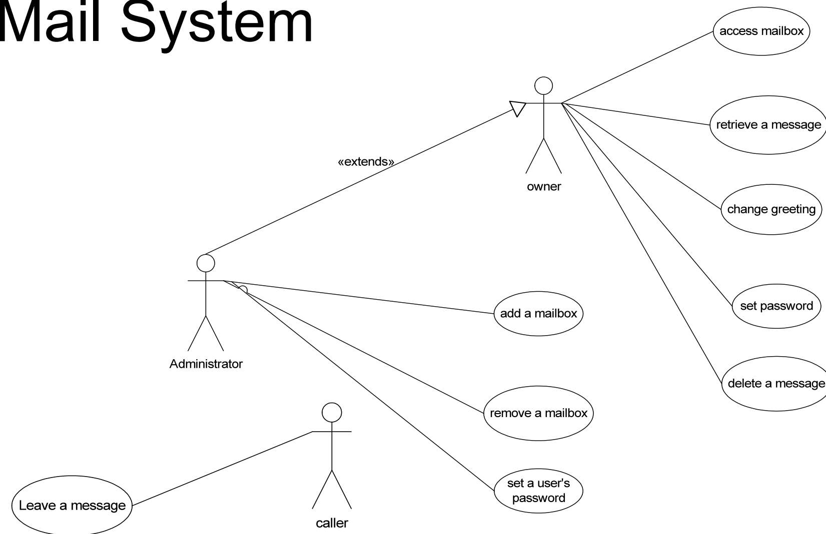


Example

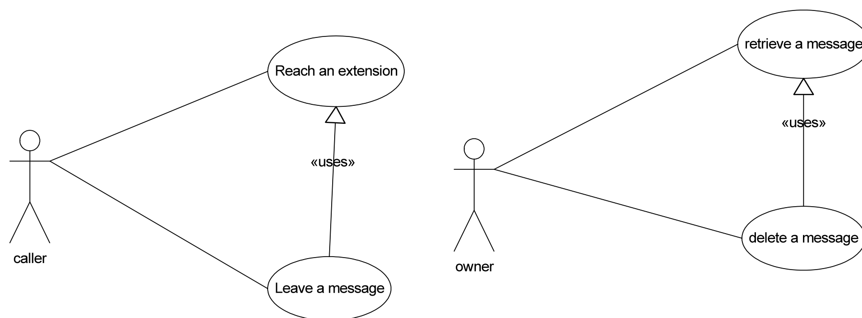




Mail System

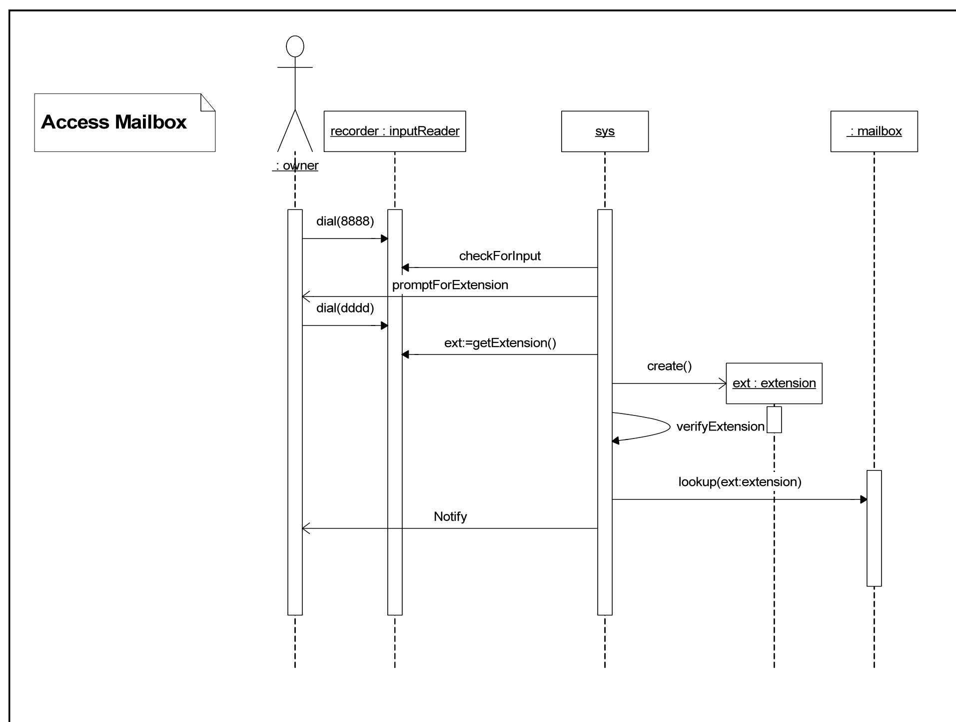


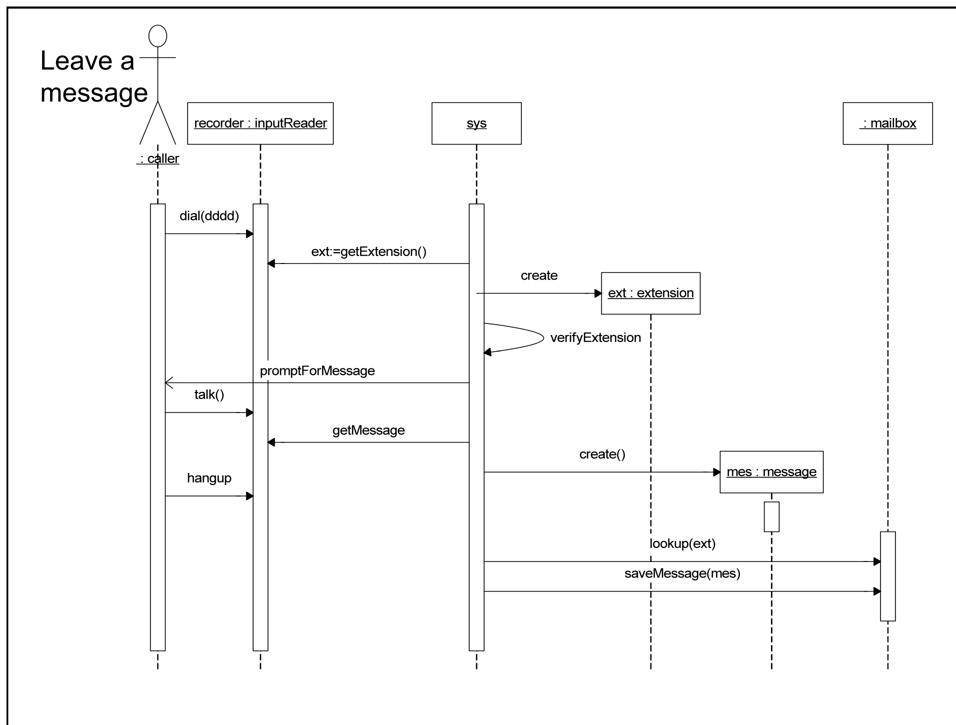
Mail System (2)

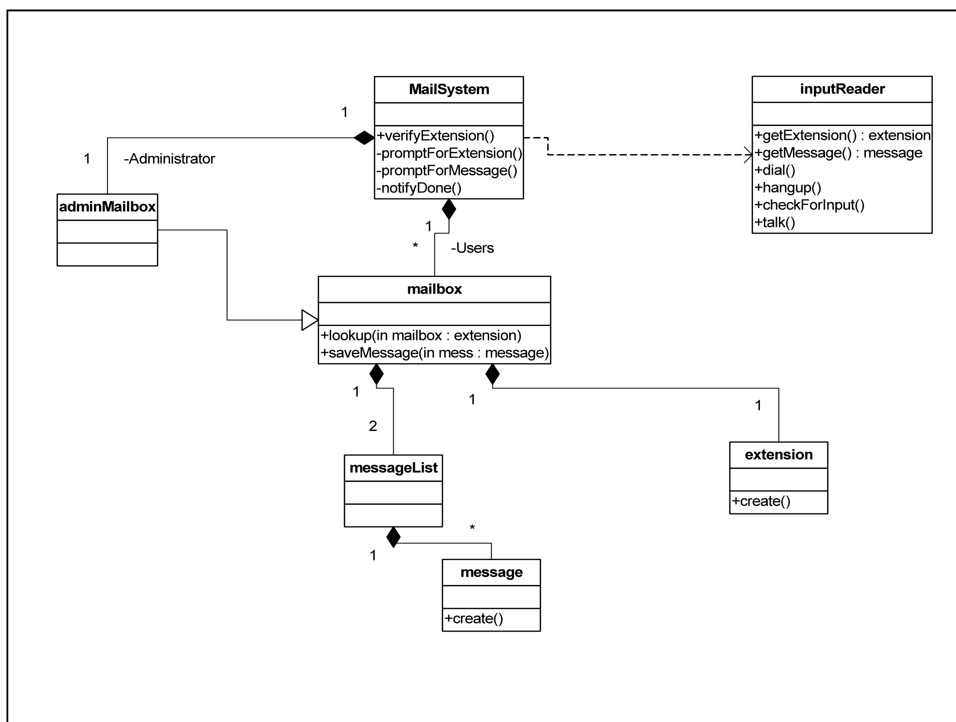


Mail System Objects

- Caller, owner, administrator
- Mailbox, extension, password, greeting
- Message, message list
- Mail system
- Input reader/device




Leave a message
: caller
recorder : inputReader
sys
: mailbox
dial(dddd)
ext:=getExtension()
create
ext : extension
verifyExtension
promptForMessage
talk()
getMessage
create()
mes : message
hangup
lookup(ext)
saveMessage(mes)


MailSystem
+verifyExtension()
-promptForExtension()
-promptForMessage()
-notifyDone()
inputReader
+getExtension() : extension
+getMessage() : message
+dial()
+hangup()
+checkForInput()
+talk()
1
-Administrator
adminMailbox
-Users
*
mailbox
+lookup(in mailbox : extension)
+saveMessage(in mess : message)
2
messageList
extension
+create()
message
+create()

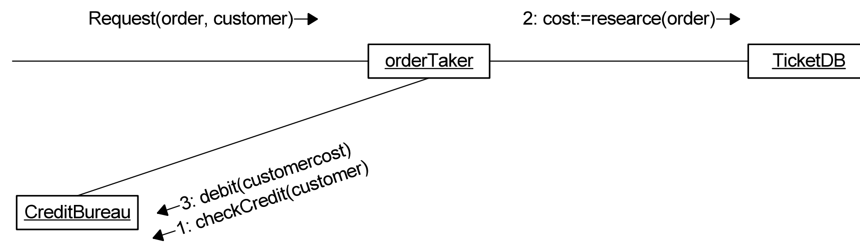
Properties of Sequence Diagrams

- Initiator is leftmost object (boundary object)
- Next is typically a control object
- Then comes entity objects

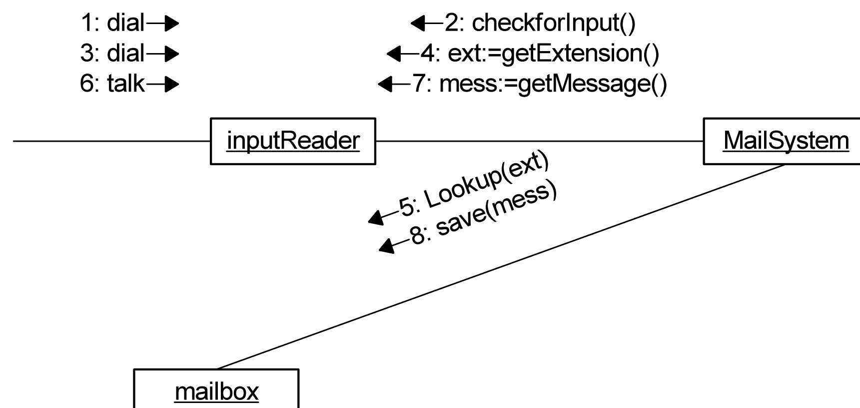
Collaboration Diagrams

- Emphasizes the organization of the objects that participate in an interaction
- Classifier roles
- Association
- Messages, flow, and sequencing

Example Collaboration Diagram



Leave a Message



Collaboration vs Sequence

- The two diagrams really show the same information
- Collaboration diagrams show more static structure (however, class diagrams are better at this)
- Sequence diagrams clearly highlight the orderings and very useful for multi-tasking

Summary (Interaction Diagrams)

- Well structured interaction diagrams:
  - Is focused on communicating one aspect of a system's dynamics
  - Contains only those elements that are essential to understanding
  - Is not so minimalistic that it misinforms the reader about the semantics that are important
- Diagrams should have meaningful names
- Layout diagram to minimize line crossings
- Use branching sparingly (leave for activity dia)

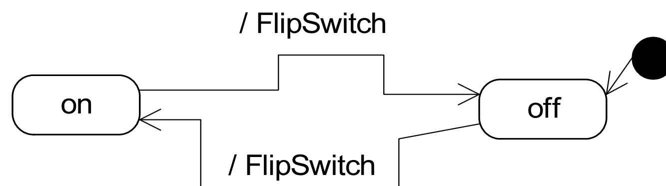
State Diagrams

- Finite state machines (i.e., automata, Mealy/ Moore, state transition)
- Used to describe the behavior of one object (or sometimes an operator) for a number of scenarios that affect the object
- They are not good for showing interaction between objects (use interaction diagrams)
- Only use when the behavior of a object is complex and more detail is needed

State Diagram Features

- Event – something that happens at a specific point
  - Alarm goes off
- Condition – something that has a duration
  - Alarm is on
  - Fuel level is low
- State – an abstraction of the attributes and relationships of an object (or system)
  - The fuel tank is in a too low level when the fuel level is below level x for n seconds

Example: on/off Switch



Using guards and actions