INF 117
Project in Software Engineering

Lecture Notes –Spring Quarter, 2008

Michele Rousseau

Set 6 – System Architecture, UML

What's Next

APRIL 2008

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
13 Week 3	14	15 Tues Rec	16	17	18	19
20 Week 4	21	22 Earth Day	23	24 Req. Iter. #3 Des. Iter: #1 Project Plan #2	25 Cust. Milestone: Req. Approved	26
27 Week 5	28	29	30 Stud. Pres-Des Order 2,3,4,1 Team App: #2 Peer Eval #2	1 Des. Iter: #2 Test-Plan It #2 (Incl Des)	2 Team Log #2 Course Log #1	

Announcements

- Reqs should be complete
  - Except minor changes
  - Make sure you show the client
- Start working on Design
  - UML
- Team logs due today

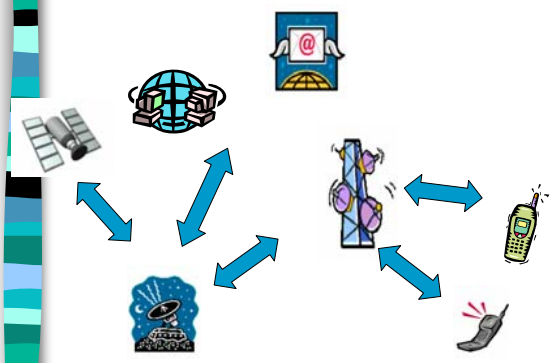
Today's Class

- Design 3 Iterations
- System Architecture
- UML

Design Iterations

- Iteration #1:
  - High Level Design Complete
  - As much Detailed (modular level) Complete as possible
- Iteration #2:
  - Complete Detailed Design
  - Integration Test Plans
- Iteration #3:
  - Minor changes based on feedback
  - Note: you should begin coding after 2nd iteration

System Architecture



Software Architecture: Essentials

- Components
  - What are the main parts?
  - What aspects of the requirements do they correspond to? Where did they come from?
  - Examples: filters, databases, GUIs, interpreters
- Connections
  - How do components communicate?
  - Examples: procedure calls, messages, pipes, event broadcast
- Constraints (including constraints on change)

How is it all organized?

Architectural design process

- System structuring
  - Decompose the system into principal sub-systems
  - identify communications between them
- Control modelling
  - A model of the control relationships between the different parts of the system
- Modular decomposition
  - The identified sub-systems are decomposed into modules

Architecture Design: Advanced

- Architectural styles
  - Restrict the way in which components can be connected
  - Prescribe patterns of interaction
  - Promote fundamental principles
  - Common styles: layered, client server, etc
- Architecture description
  - Boxes and arrows
  - UML
  - Architecture description languages

From Architecture to Modules

- Repeat the design process
  - Design the internal architecture of a component
  - Define…
    - the purpose of each module
    - the provided interface of each module
    - the required interface of each module
- Do this over and over again
  - Until each module has a simple, well-defined…
    - …internal architecture
    - …purpose
    - …provided interface
    - …required interface
- Until all modules "hook up"

Some Principles

- Rigor
  - ensures all requirements are addressed
- Separation of concerns
  - Modularity
    - allows work in isolation because *components are independent* of each other
    - decompose a complex system into less complex sub-systems; *divide & conquer*
    - *(re-)use* existing modules
    - *understand* the system in pieces
  - Abstraction
    - allows work in isolation because *well-defined interfaces* guarantee that components will work together

Some More Principles

- Anticipation of change
  - allows changes to be absorbed seamlessly
- What makes a good module?
  - *High cohesion*: all internal parts are closely related.
  - *Low coupling*: modules rely on each other as little as possible
  - *Information Hiding*: Each module hides its internal structure.
  - *Generality*: allows components to be reused throughout the system
  - *Incrementality*: allows the software to be developed with intermediate working results
- Remember to document your rationale!

UML

UML Concepts

- Display the boundary of a system & its major functions using use cases and actors
- Illustrate use case realizations with interaction diagrams
- Illustrate scenarios with use case diagrams and sequence 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

Diagrams in UML

- A diagram is a view into a model
  - Presented from the aspect of a particular stakeholder
  - Provides a partial representation of the system
  - Is (should be?) semantically consistent with other views

Types of UML Diagrams

Structure (6 types)

- Class diagrams
- Object diagram
- Package diagram
- Composite structure diagram
- Component diagram
- Deployment Diagram

Behavior (4 types)

- Activity diagram
- Use Case diagram
- State machine diagram
- Interaction diagrams
  - Sequence diagram
  - Communication diagram
  - Interaction overview diagram
  - Timing diagram

If the appropriate diagram is not part of UML ***use it anyways***

UML & the S/W Process (Design)

- **Use Cases**
  - Define the system Boundaries
- **Class Diagrams**
  - From a software perspective
    - Show classes & how they interrelate
- **Sequence Diagrams**
  - For Common Scenarios
    - Pick most significant scenarios from Use Cases
    - Use CRC cards or sequence diagrams to determine how the software should behave
      - Class, Responsibilities, Collaborators (CRC) cards are index cards used to represent
        - the responsibilities of classes
        - interaction between the classes
- **Package Diagrams**
  - Show large-scale organization of the system
- **State Diagrams**
  - Used for classes with complex lifecycles
- **Deployment Diagrams**
  - Show the physical layout of the software

All of these can be used for design

Class Diagrams

"A *Class Diagram* describes the types of objects in the system and the various kinds of *static relationships* that exist among them"

Class Name
Attributes (Name:type)
Operations (Name: Parameters)

Makes it easier to see the big picture

- Know what a class does at a glance

Attributes and Operations

- Attributes →
  - Describes a property as a line of text within the class box
  - Attribute name corresponds to the name of a field in a programming language
  - *Visibility Marker* →
    - Denotes whether an attribute is Public (+) or Private (-)
- Operations →
  - Actions that a class knows to carry out
  - *Corresponds* to methods on a class

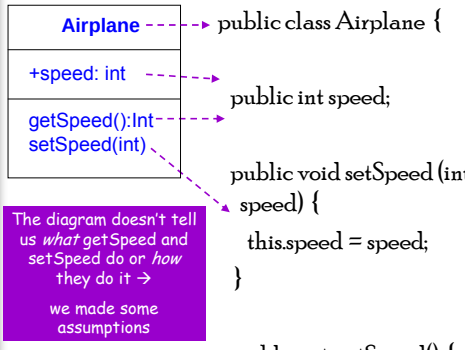
Operation & Method are *not* the same thing

An Operation is the procedure declaration

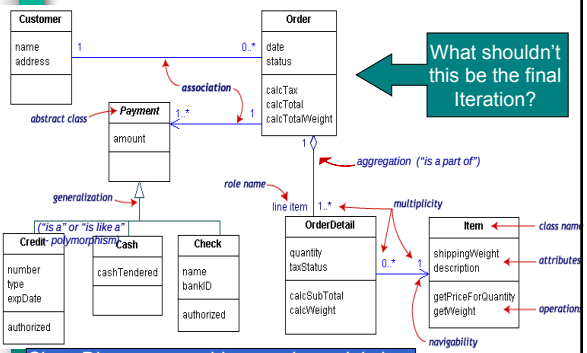
A Method is the body of a procedure

- Associations →
  - Describe the relationship between two classes

Example of a Class



Class Diagram (Example)



Sequence Diagrams

- One type of Interaction Diagram
- Also describe the behavior of the system
- Details how operations are carried out
  - What messages are sent when
- Organized according to time
- Objects listed from left to right
  - According to when they take part in the message sequence

Sequence Diagrams

- **Represent one scenario**
- **Describes the dynamic behavior of a system**
- Details how operations are carried out
  - What messages are sent when
- **Good at**
  - describing the behavior of several objects within a single use case
  - Showing collaborations between objects
  - Describing system interaction w.r.t. time (objects ordered from L to R – according to time)
- **Not good at precise definition of the behavior**

Sequence Diagrams (2)

Basic Elements of a Sequence Diagram

- **Objects**
  - Lifelines: time goes from top to bottom
  - May be several instances of one class
  - Boxes on lifelines show if object is active
- **Messages**
  - Analogous to method calls in a program
  - Can have parameters
- **Special messages**
  - New — shown by position of object
  - Delete — shown with a big X
  - Return messages
  - Self-calls

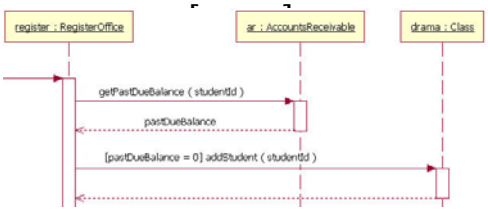
Sequence Diagrams: Terms

- A lifeline, represents the time that an object exists
  - Represented as a vertical line.
- An activation bar represents the duration of execution of the message
  - Represented by a vertical rectangle
- A message call is represented by an arrow between activation bars
  - A simple message return is represented by a dashed arrow
- A self call is when an object calls itself
- A note is used to clarify details
  - Represented with a dog-eared rectangle

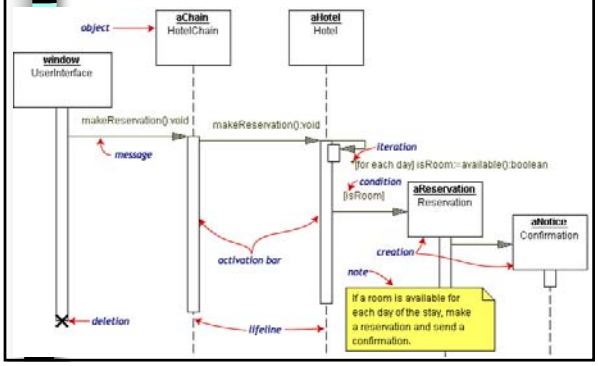
(Notes can be put into any kind of UML diagram)

Guards

- When a condition must be met before a message is sent
- Represented by brackets on the



Sequence Diagram Example: Hotel Reservation



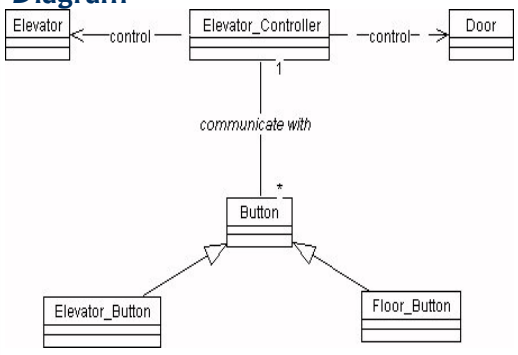
Putting them together

- Class Diagrams
- Scenarios
- Use Cases
- Sequence Diagrams

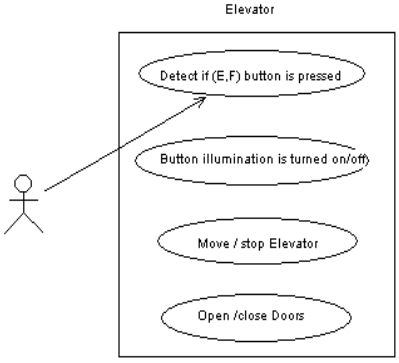
UML is iterative & Incremental

- How do they all work together

Elevator Example: Basic Class Diagram



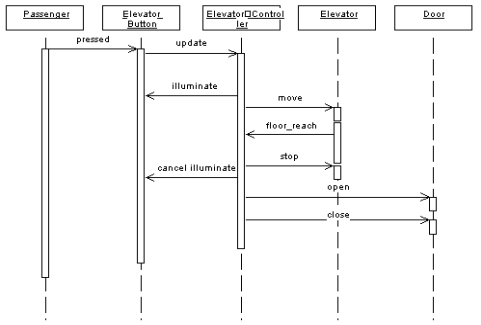
Elevator Example: Use Case



Elevator Example: Scenario

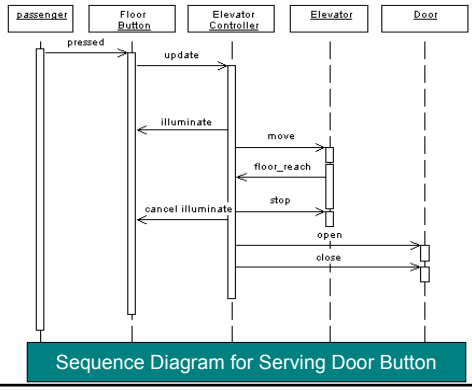
- Passenger pressed floor button
- Elevator system detects floor button pressed
- Elevator moves to the floor
- Elevator doors open
- Passenger gets in and presses elevator button
- Elevator doors closes
- Elevator moves to required floor
- Elevator doors open
- Passenger gets out
- Elevator doors closes

Elevator Example: Sequence Diagram



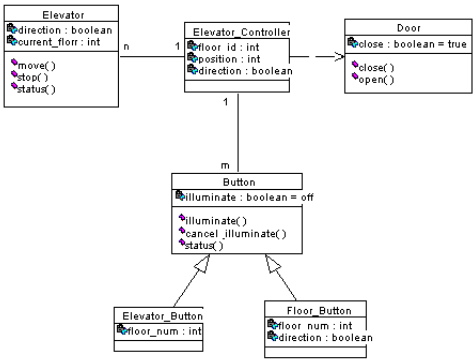
Sequence Diagram for Serving Elevator Button

Elevator Example: Sequence Diagram



Sequence Diagram for Serving Door Button

Elevator Example: Revising the Class Diagram

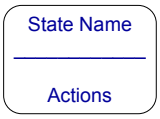


State Transition Diagrams

- An ***event*** occurs at a point in time and
  - transmits information from one object to another
- An ***action*** occurs in response to an event and cannot be interrupted
- An ***activity*** is an operation with certain duration that can be interrupted by another event
- A ***guard*** is a logical condition placed before a transition that returns either a true or a false.

State Transition Diagrams: Notation

- State symbol:

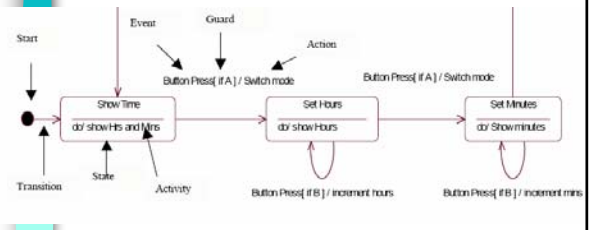


- Transition Symbol:

Example: Simple Digital Watch

- Has a display and two buttons to set it
  - A & B Button
- Watch has two modes of operation
  - display time – hours:minutes
  - set time – two modes
    - Set minutes.
- The "A" button is used to select modes.
    - Set hours
  - On each press, the mode advances in sequence:
    - display → set hours → set minutes → display etc.
  - Within the sub modes, the "B" button is used to advance the hours or minutes once each time it is pressed.
- Buttons must be released before they can generate another event.

State Transition EX: Digital Watch



Activity Diagrams

- A flow chart with support for parallel behavior
- Branches and Merges model the conditional behavior
- Branch: has a single incoming transition multiple, conditional, outgoing transitions
- Merge: where conditional behavior terminates

Each branch has a corresponding merge

- Represented as a Diamond

Activity Diagram (2)

- Forks and Joins model parallel behavior
- Fork: has a single incoming transition and multiple outgoing transitions (exhibiting parallel behavior)
- Join: synchronizes the parallel behavior
  - All parallel behaviors complete at the join

Each Fork has a corresponding Join

- Represented as a thick line

- Conditional Thread: A condition on the thread originating from the fork to create an exception for the join rule.

Activity Diagram (3)

- Synch State: synchronizes different activities so they make a transition to the next activity at the same time

- When to use Activity Diagrams?
  - When modeling parallel behavior or
  - Documenting the logic of a business process

Activity Diagram:

ATM Ex.