UML: Unified Modeling Language

CS169
Lecture 5

Modeling

- Describing a system at a high level of abstraction
  - A model of the system
  - Used for requirements and specification
- Many notations over time
  - State machines
  - Entity-relationship diagrams
  - Dataflow diagrams
  - … see last lecture …

Recent History: 1980's

- The rise of object-oriented programming
- New class of OO modeling languages
- By early '90's, over 50 OO modeling languages

Recent History: 1990's

- Three leading OO notations decide to combine
  - Grady Booch (BOOCH)
  - Jim Rumbaugh (OML: Object Modeling Technique)
  - Ivar Jacobsen (OOSE: OO Soft. Eng)
- Why?
  - Natural evolution towards each other
  - Effort to set an industry standard

UML

- UML stands for
  Unified Modeling Language
- Design by committee
  - Many interest groups participating
  - Everyone wants their favorite approach to be "in"

UML (Cont.)

- Resulting design is huge
  - Many features
  - Many loosely unrelated styles under one roof
- Could also be called
  Union of all Modeling Languages

This Lecture

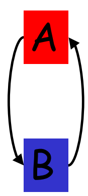
- We discuss
  - Use Case Diagrams — for functional models
  - Class Diagrams — for structural models
  - Sequence Diagrams
  - Activity Diagrams
  - State Diagrams

  (Sequence, Activity and State Diagrams: for dynamic models)
- This is a subset of UML
  - But probably the most used subset

Running Example: Automatic Train

- Consider an unmanned people-mover
  - as in many airports
- Train
  - Moves on a circular track
  - Visits each of two stations (A and B) in turn
  - Each station has a "request" button
    - To stop at this station
  - Each train has two "request" buttons
    - To stop at a particular station

Picture



Use-Cases

- Describe functionality from the user's perspective
- One (or more) use-cases per kind of user
  - May be many kinds in a complex system
- Use-cases capture requirements

An Example Use-Case in UML

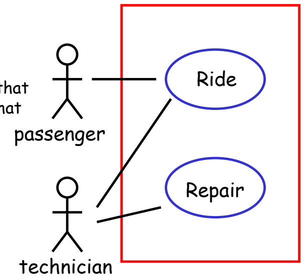
- Name
  - Normal Train Ride
- Actors
  - Passenger
- Entry Condition
  - Passenger at station
- Exit Condition
  - Passenger leaves station

An Example Use-Case in UML

- Event-flow
  - Passenger presses request button
  - Train arrives and stops at platform
  - Doors open
  - Passenger steps into train
  - Doors close
  - Passenger presses request button for final stop
  - …
  - Doors open at final stop
  - Passenger exits train
- Non-functional requirements

Use Case Diagram

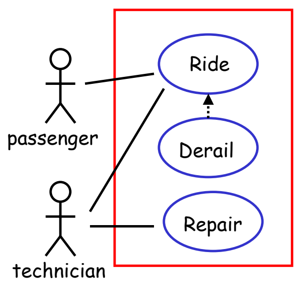
- Graph showing
  - Actors
  - Use cases
  - Edges actor-case if that actor is involved in that case
- Actors
  - Stick figures
- Use cases
  - Ovals



Exceptional Situations

- Use cases have relationships
  - Inclusion (E.g., push button included in ride)
  - Variations
- UML has a special notation
  - The "extends" relationship to express a exceptional variation of a use case
  - Normally used to express errors

Extension



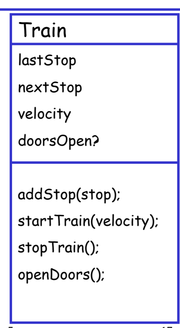
Dotted arrow pointing to "normal" case

Summary of Use Cases

- Use Case Diagram
  - Shows all actors, use cases, relationships
- 5 parts to each use case
  - Name, Actors, Entry/Exit Conditions, Event Flow
  - Actors are agents external to the system
    - E.g., users
  - Event flows are sequence of steps
    - In English

Class Diagrams

- Describe classes
  - In the OO sense
- Each box is a class
  - List fields
  - List methods
- The more detail, the more like a design it becomes

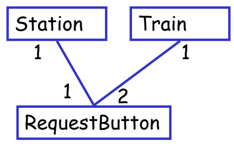


Class Diagrams: Relationships

- Many different kinds of edges to show different relationships between classes
- Mention just a couple

Associations

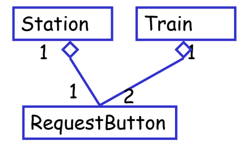
- Capture n-m relationships
  - Subsumes ER diagrams
- Label endpoints of edge with cardinalities
  - Use * for arbitrary
- Typically realized with embedded references
- Can be directional (use arrows in that case)



One request button per station; each train has two request buttons

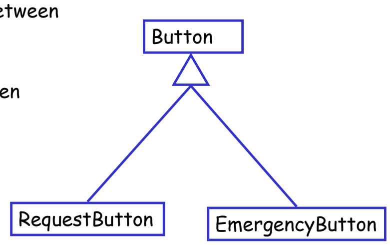
Aggregation

- Show *"contains a"* relationships
- Station and Train classes can contain their respective buttons
- Denoted by open diamond on the "contains" side



Generalization

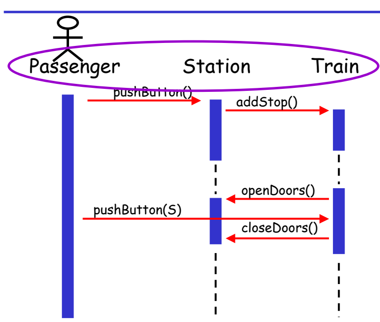
- Inheritance between classes
- Denoted by open triangle



Sequence Diagrams

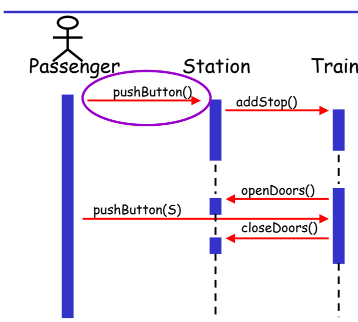
- A table
  - Columns are classes or actors
  - Rows are time steps
  - Entries show control/data flow
    - Method invocations
    - Important changes in state

Example Sequence Diagram



Classes & Actors

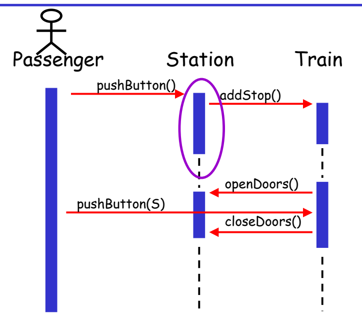
Example Sequence Diagram



Method invocation

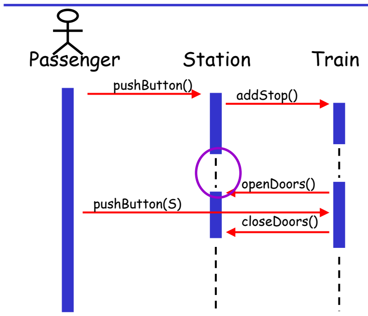
Note: These are all synchronous method calls. There are other kinds of invocations.

Example Sequence Diagram



Invocation lifetime spans lifetimes of all nested invocations

Example Sequence Diagram



"Lifelines" fill in time between invocations

Sequence Diagrams Notes

- Sequence diagrams
  - Refine use cases
  - Gives view of dynamic behavior of classes
    - Class diagrams give the static class structure
- Not orthogonal to other diagrams
  - Overlapping functionality
  - True of all UML diagrams

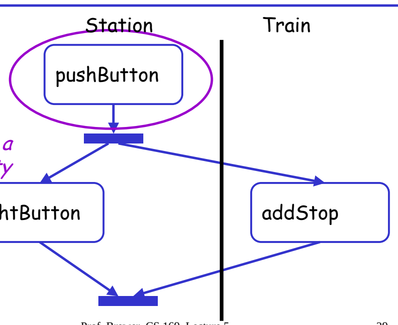
Activity Diagrams

- Reincarnation of flow charts
  - Uses flowchart symbols
- Emphasis on control-flow
- Two useful flowchart extensions
  - Hierarchy
    - A node may be an activity diagram
  - Swim lanes

Example Activity Diagram

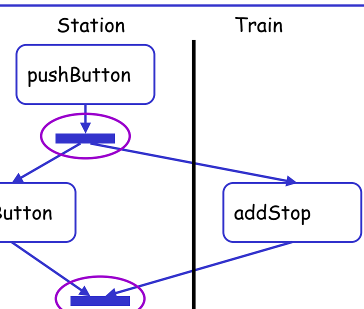
Activities in rounded rectangles

May itself be a nested activity diagram



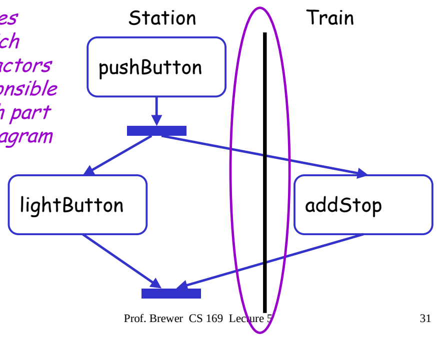
Example Activity Diagram

Concurrency, fork & join



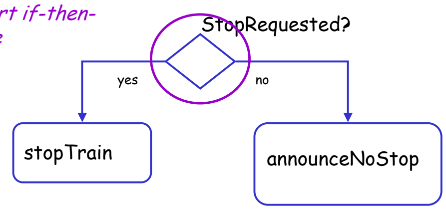
Example Activity Diagram

Swim lanes show which classes/actors are responsible for which part of the diagram



Another Example Activity Diagram

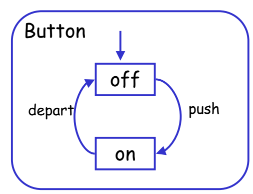
Classic flow-chart if-then-else



StateCharts

- Hierarchical finite automata
  - Invented by David Harel, 1983
- Specify automata with many states compactly
- Complications in meaning of transitions
  - What it means to enter/exit a compound state

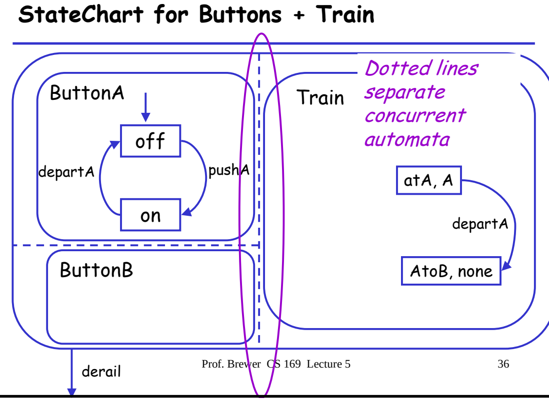
Example Simple StateChart



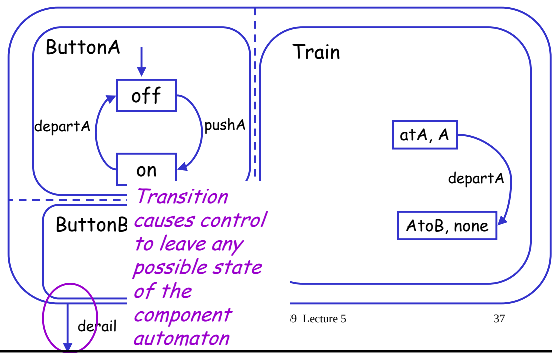
StateChart for the Train

- A train can be
  - At a station
  - Between stations
- Pending requests are subset of {A,B}
- 16 possible states
  - Transitions: pushA, pushB, departA, departB, …

StateChart for Buttons + Train



StateChart for Buttons + Train



Opinions about UML: What's Good

- A common language
  - Makes it easier to share requirements, specs, designs
- Visual syntax is useful, to a point
  - A picture is worth 1000 words
  - For the non-technical, easier to grasp simple diagrams than simple pseudo-code
- To the extent UML is precise, forces clarity
  - Much better than natural language
- Commercial tool support
  - Something natural language could never have

Opinions On UML: What's Bad

- Hodge-podge of ideas
  - Union of most popular modeling languages
  - Sublanguages remain largely unintegrated
- Visual syntax does not scale well
  - Many details are hard to depict visually
    - Ad hoc text attached to diagrams
  - No visualization advantage for large diagrams
    - 1000 pictures are very hard to understand
- Semantics is not completely clear
  - Some parts of UML underspecified, inconsistent
  - Plans to fix

UML is Happening

- UML is being widely adopted
  - By users
  - By tool vendors
  - By programmers
- A step forward
  - Seems useful
  - First standard for high-levels of software process
  - Expect further evolution, development of UML