CSE 416

UML
Sequence Diagrams

For the Object-Oriented (Java)
part of your system

Reading / Reference

- Reading

www.ibm.com/developerworks/rational/library/3101.html

- Reference

www.lucidchart.com/pages/uml-sequence-diagram

Interaction Diagrams

- Sequence diagrams and collaboration diagrams
- A series of diagrams describing the *dynamic behavior* of an object-oriented system
- Often used to model a use case
- The purpose of Interaction diagrams is to:
  - Model interactions between objects
  - Verify that a use case description can be supported by the existing classes
  - Identify new classes
  - Assign responsibilities/operations to classes

We focus on sequence diagrams

3

UML Sequence Diagram

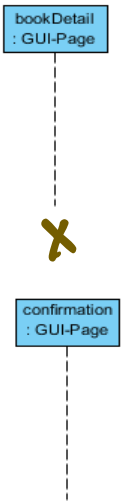
- Sequence diagram - an interaction diagram that models a single scenario (use case) executing in the system
  - perhaps 2nd most used UML diagram (behind class diagram)
- Illustrates how objects interact with each other
- Emphasizes time ordering of messages
- Can model simple sequential flow, branching, iteration, recursion and concurrency

4

Lifeline

- Think of a lifeline as a "live" object
- Lifelines usually represent object instances
- An "X" is shown when the object is destroyed
- Placement
  - usually across the top of the diagram
  - Depending on tool, you might lower the placement of the lifeline if object activation occurs during the use case

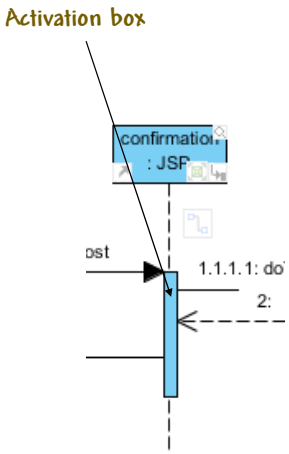
Might not be shown if it doesn't clarify the design



5

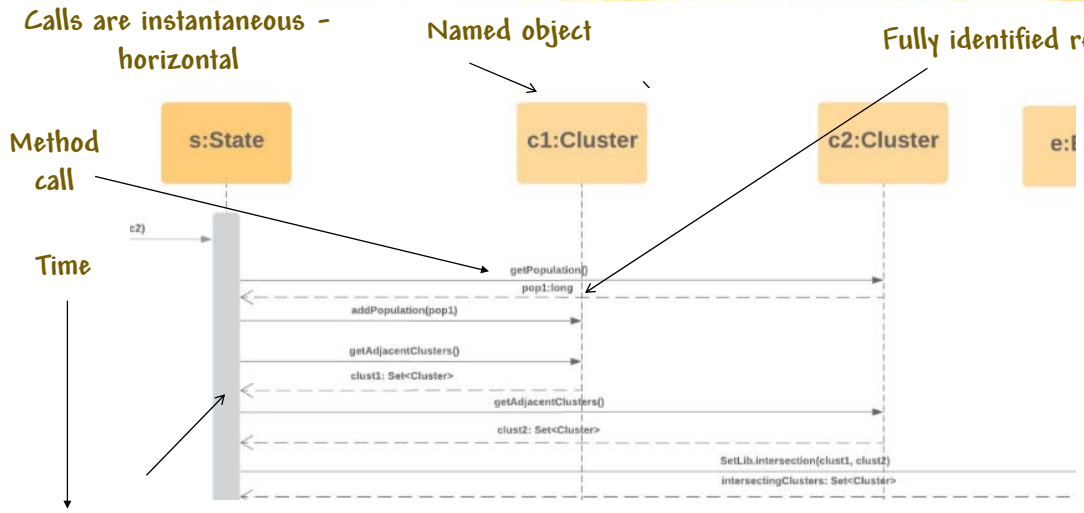
Indicating Method Calls

- Activation box: thick box over object's life line; drawn when object is on the stack
  - Either that object is running its code, or it is on the stack waiting for another object's method to finish
  - Nest to indicate recursion



6

Sequence Diagram Syntax



7

Key Components

- Participant: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached arrow or an arrow attached to an actor
- Message: communication between objects/actors
- Axes in a sequence diagram:
  - horizontal: which object/participant is acting
  - vertical: time (down -> forward in time)

In a GUI system the initial participant is usually an actor

8

Messages

- An interaction between two objects is performed as a message sent from one object to another
  (e.g., method call)
- If an object sends a message to another object, object 1 must have visibility to object 2 (i.e., have a handle)
- A message is represented by an arrow between the life lines of two objects
  - Self calls are also allowed
  - The time required by the receiver object to process the message is denoted by an *activation-box*.
- A message is labeled at a minimum with the method name, and if needed, the parameters

9

Messages

- Solid arrow heads represent synchronous calls
  - a synchronous message waits until the message is done (e.g., invoking a subroutine)
- Open arrow heads represent asynchronous messages
  - An asynchronous message can continue processing and doesn't have to wait for a response
  - Example: Ajax calls from GUI
- Dashed lines represent reply messages.

Some of these formatting issues are tool dependent

10

Arrow Labels

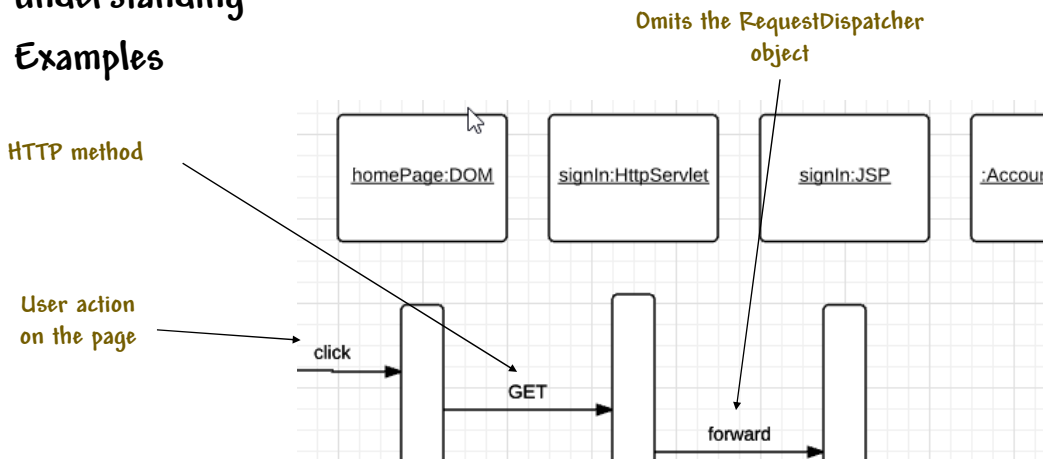
- Method call
  - Label the call arrow with the method name
  - Include parameters if they are not obvious
- Return
  - Model a return value only when you need to refer to it elsewhere, e.g. as a parameter passed in another message

In general, don't model obvious interactions if the modeling tool is not able to automatically generate code

11

Simplification

- You can simplify in some cases, once you have shown understanding
- Examples



12

Simplification - Fundamental Objects

- DOM - object representation of the GUI
- XMLHttpRequest- Standard browser object - interacts with the server
- Window object uses a fetch method to interact with server
- Servlet - standard server object to handle the request from the client
- Persistence layer - Standard server object to receive object requests for the DB
  - Best represented as the em:EntityManager object
  - Receives calls as in JPA

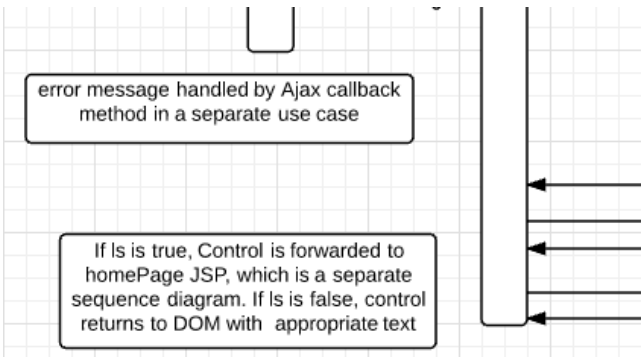
These concepts covered in more depth in a future session

You should show these fundamental objects once in your sequence diagrams when there is client and/or DB interaction

13

Simplification Comments

- Some simplification should be indicated with comments



14

Realistic Design Approach

- Use your sequence diagrams to identify classes and class attributes needed in your class diagram
- Work both simultaneously (e.g. add methods to your class diagram once you see that you need it)
- Don't be reluctant to modify your design during this stage

15

Project Team Approach

- The first few diagrams will be very difficult to do
- Do the first few as a team (with lots of team interaction)
- Once your team begins to understand your design philosophy and framework philosophy, you will be able to assign parts to team members
- Look for common design approaches (e.g., DB access, server access, session management), you might be able to use sub-diagrams

16

Project Hints

- Be sure to show an understanding of the object in your GUI (i.e., DOM)
- Concentrate on server logic
  - GUI object interaction will vary based on your choice of development framework
  - Generalize the DB component in your initial sequence diagrams (e.g., just show a general call to a persistence layer object)
  - If you use JDBC, you will need to show details of the use of interaction with JDBC objects

17

Design Review

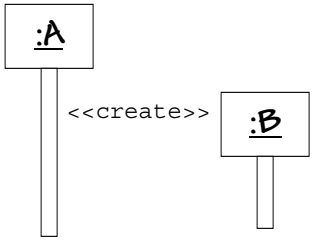
- Design review will be organized along the lines of use cases (and corresponding sequence diagrams and data flow diagrams)
- Your team gets to pick the first use case to show
- Clarity of thinking and consistency are more important than getting the best possible design approach

18

Object Instantiation

- An object may create another object via a <<create>> message.

Preferred

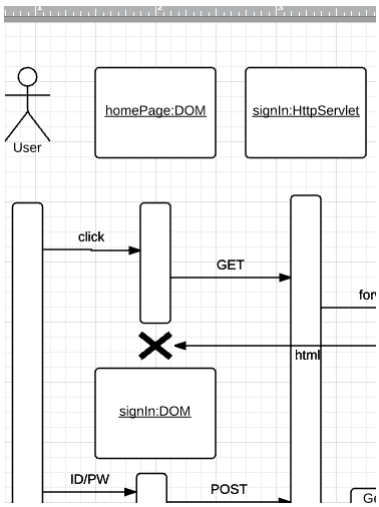


Using new is OK, but you will be probably use the factory design pattern

19

Object Destruction

- An object may destroy another object via a <<destroy>> message.
  - Avoid modeling object destruction unless memory management is critical
  - You will probably only show this when the return of an html page has the effect of destroying the previous page in the browser



20

Indicating Selection and Loops

- frame: box around part of a sequence diagram to indicate selection or loop
  - if -> (opt) [condition]
  - if/else -> (alt) [condition], separated by horiz. dashed line
  - loop -> (loop) [condition or items to loop over]

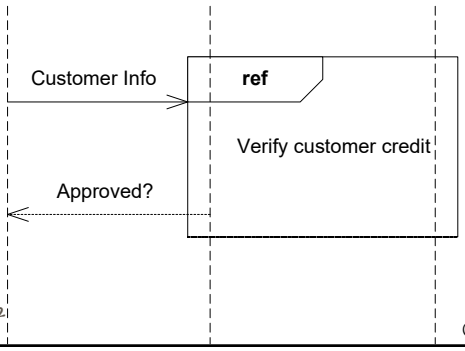
Loops are not very helpful in sequence diagrams. If you need to show a loop, you might just indicate it with a comment

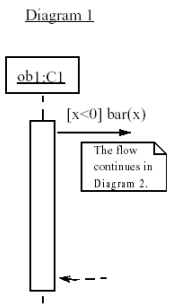
21

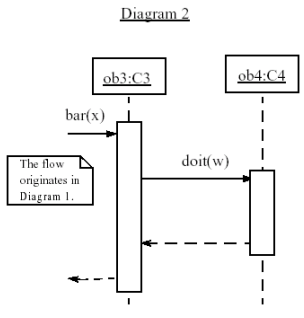
Linking Sequence Diagrams

- If one sequence diagram is too large or refers to another diagram, indicate it with either:
  - An unfinished arrow and comment
  - A "ref" frame that names the other diagram

Although this might result from the use case being too large



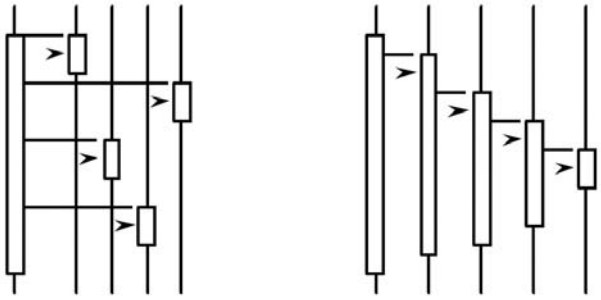




22

(De)centralized System Control

- What can you say about the control flow of each of the following systems?
  - centralized?
  - distributed?



23

Why Not Just Code It?

- Sequence diagrams can be somewhat close to the code level. So why not just code the algorithm rather than drawing it as a sequence diagram?
  - Allows you to think through design issues
  - A good sequence diagram is well above the level of the real code
  - Tool might generate code
  - Sequence diagrams are language-agnostic (can be implemented in many different OO languages)
  - Easier to do as a team
  - Can see many objects/classes at the same time

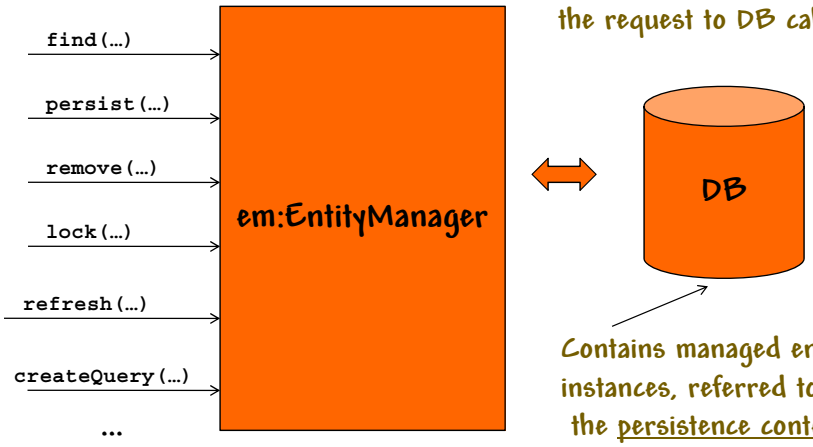
24

Slide from a future session

EntityManager

Methods operate on entity objects

EntityManager translates the request to DB calls



Use EntityManager instead of direct DB access

Contains managed entity instances, referred to as the persistence context

DB may also contain detached entity instances

25