Sequence Diagrams

Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

mfelici@inf.ed.ac.uk

What are Sequence Diagrams?

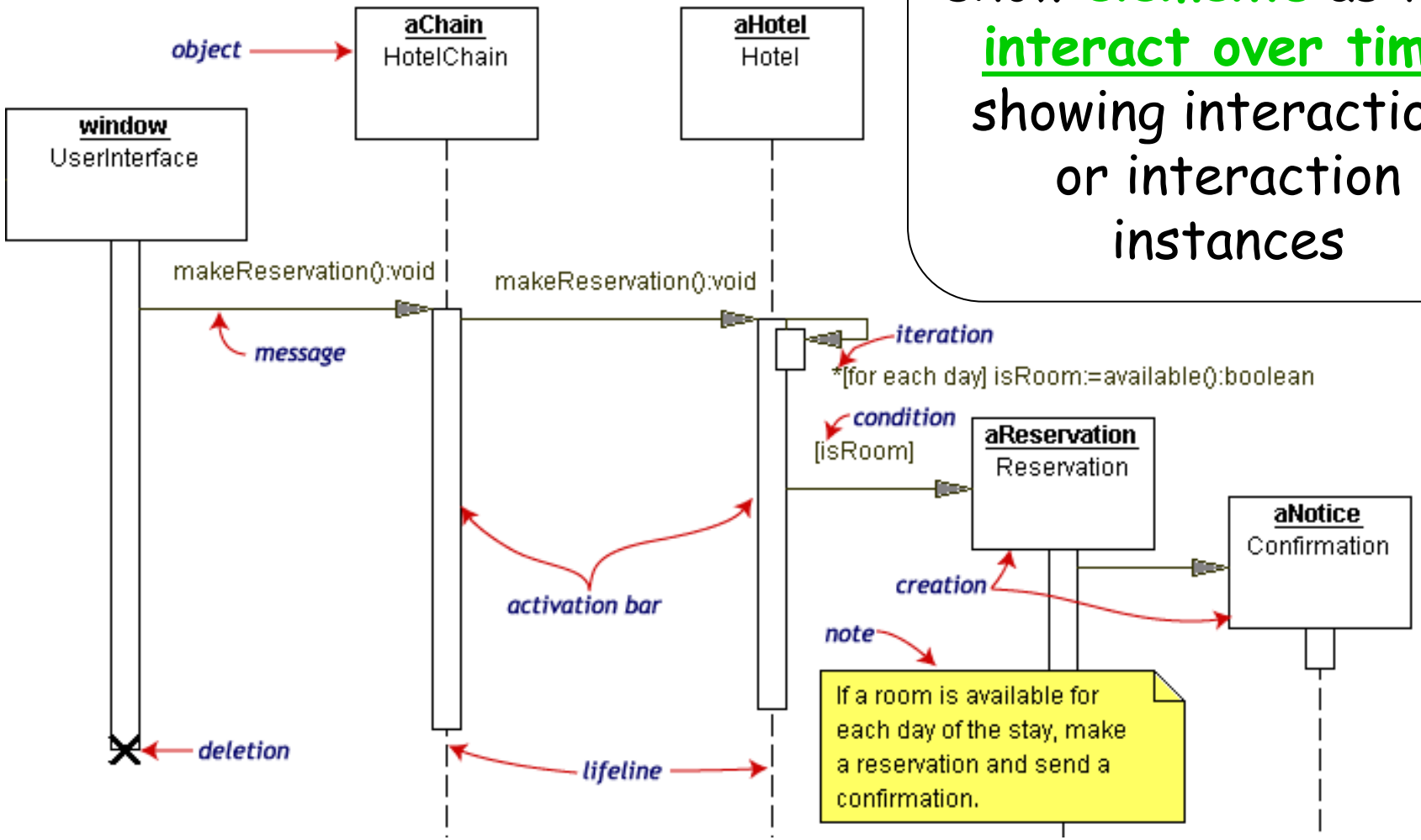
- **Sequence Diagrams** are **interaction diagrams** that detail how **operations** are carried out
  - Interaction diagrams model important runtime interactions between the parts that make up the system
- **Sequence Diagrams**
  - capture the interaction between **objects** in the context of a **collaboration**
  - show object instances that play the roles defined in a collaboration
  - don't show the structural relationships between objects
  - show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when

What Do Sequence Diagrams Model?

- Model high-level **interaction** between active **objects** in a system
- Model the interaction between object instances within a **collaboration** that realizes a **use case**
- Model the interaction between objects within a collaboration that realizes an **operation**
- Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction
- Capture the interaction that takes place in a collaboration that either realizes a use case or an operation (**instance diagrams** or **generic diagrams**)
- Capture high-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as **system sequence diagrams**)

Sequence Diagrams at a Glance

Sequence Diagrams show **elements** as they **interact over time**, showing interactions or interaction instances

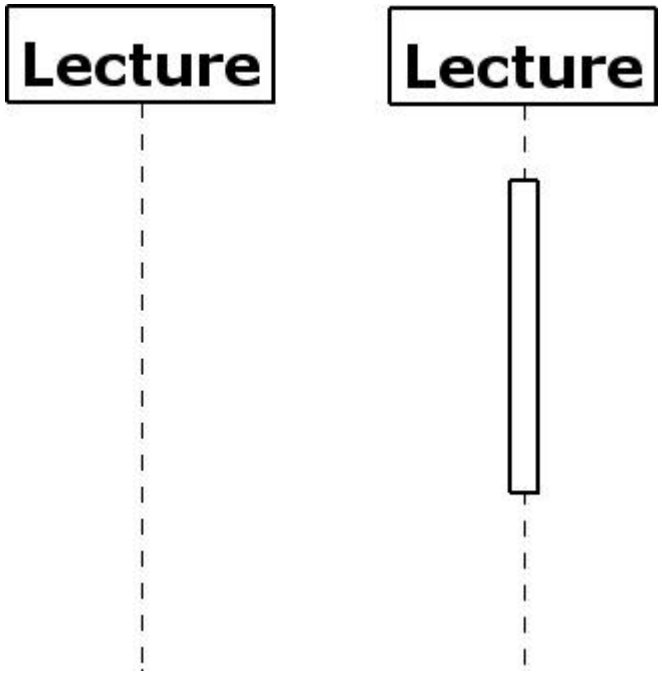


Participants in a Sequence Diagram

- A **sequence diagram** is made up of a collection of **participants**
- Participants- the system parts that interact each other during the sequence
- **Classes** or **Objects**: each object (class) in the interaction is represented by its named icon along the top of the diagram
- In **UML 1.x**, participants were usually software objects (instances of classes) in object oriented programming sense
- In **UML 2.0**, as general modeling language, participants are also at the level of system parts

Time in a Sequence Diagram

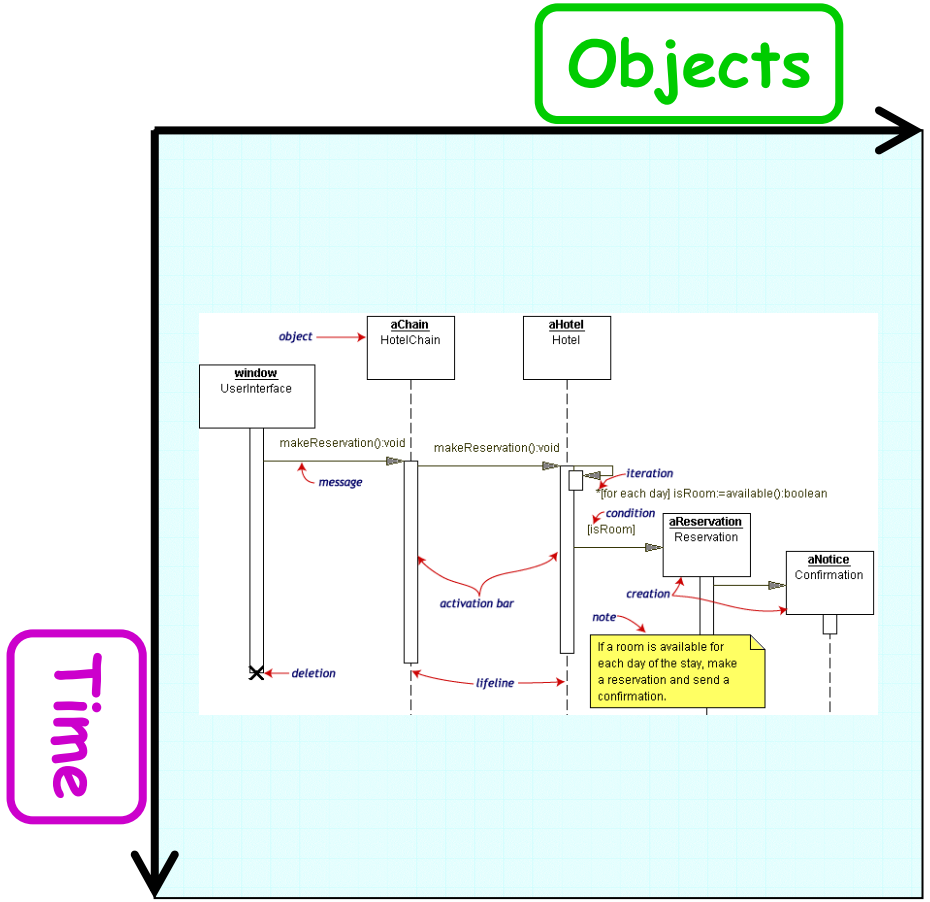
- **Sequence diagrams** are organized according to time
- Each participant has a corresponding **lifeline**
- **Lifelines**: each vertical dotted line is a **lifeline**, representing the time that an **object** exists
- **Activations**: an activation (shown as tall, thin rectangle on a **lifeline**) represents the period during which an element is performing an operation. The top and the bottom of the of the rectangle are aligned with the **initiation** and the **completion time** respectively



- **Notes**
  - Time in a sequence diagram is all a about ordering, not duration
  - The vertical space in an interaction diagram is not relevant for the duration of the interaction

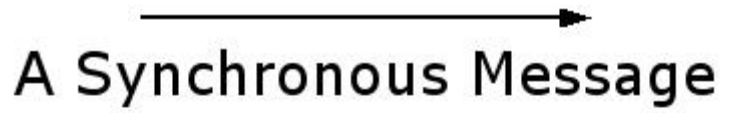
Sequence Diagrams' Dimensions

- **Objects**
  - The horizontal axis shows the elements that are involved in the interaction
  - Conventionally, the **objects** involved in the operation are listed from left to right according to when they take part in the message sequence
  - However, the elements on the horizontal axis may appear in any order
- **Time**
  - The vertical axis represents **time** proceedings (or progressing) down the page



Events, Signals and Messages

- An Event is any point in an interaction where something occurs
- Signals and Messages are similar concepts
  - Messages on a sequence diagram are specified using an arrow from the participant (message caller) that wants to pass the message to the participant (message receiver) that is to receive the message
  - Messages: a message (or stimulus) is represented as an arrow going from the sender to the top of the activation bar of the message on the receiver's lifeline



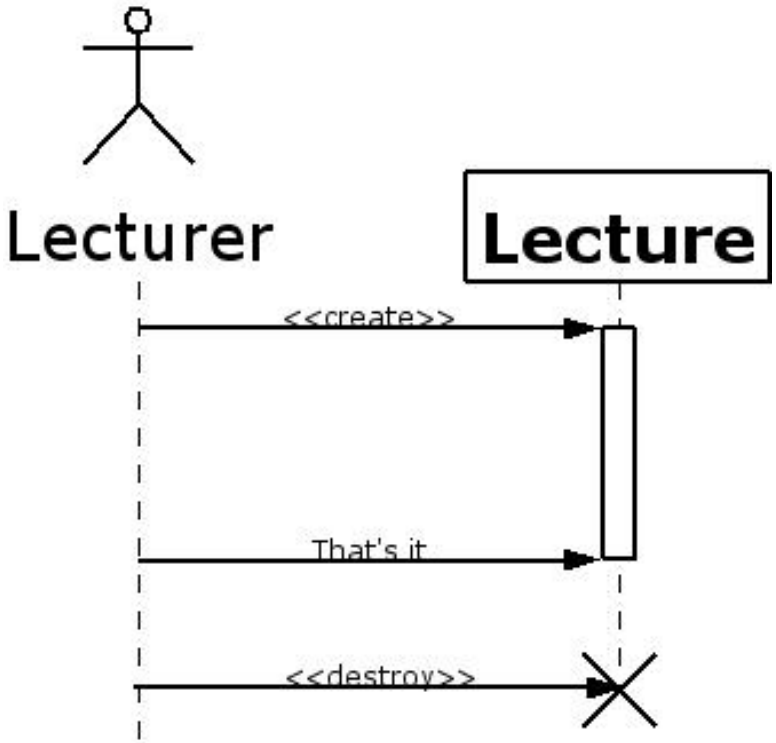
Types of Messages

- **Reflexive Communications**: similar to a reflexive association or link, an element may communicate with itself where communication is sent from the element to itself. Sending messages to itself means an object has two activations simultaneously.
- **Repetitions**: involve repeating a set of messages or stimuli within a generic-form interaction. Messages are grouped together in a rectangle. The expression in square brackets, [ ], is a **condition**. The asterisk "*****" means **iteration**.
- **Conditionality**: **branching** results in a choice of two different messages (or operation calls) being sent to the same object, the lifeline of the object splits with two activations. The separate lifelines merge back together after the completion of different actions in response to the different messages.
- **Return Values**: often worthwhile to label the return value because it may be used later in the interaction

Creation and Destruction Messages

- **Element Creation**: when an element is created during an interaction, the communication that creates the element is shown with its arrowhead to the element

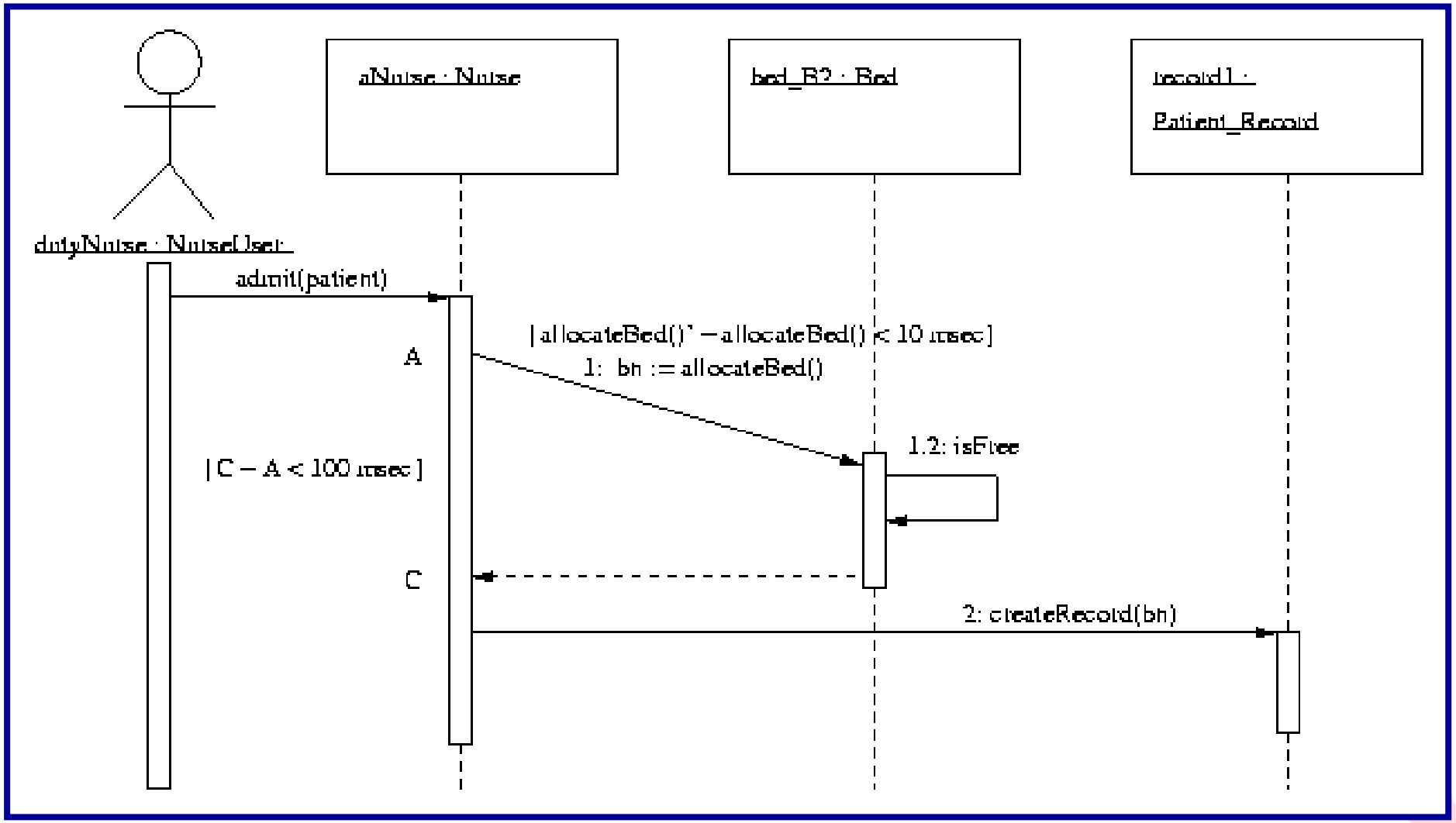
- **Element Destruction**: When an element is destroyed during an interaction, the communication that destroys the element is shown with its arrowhead to the element's lifeline where the destruction is marked with a large X symbol



Timing

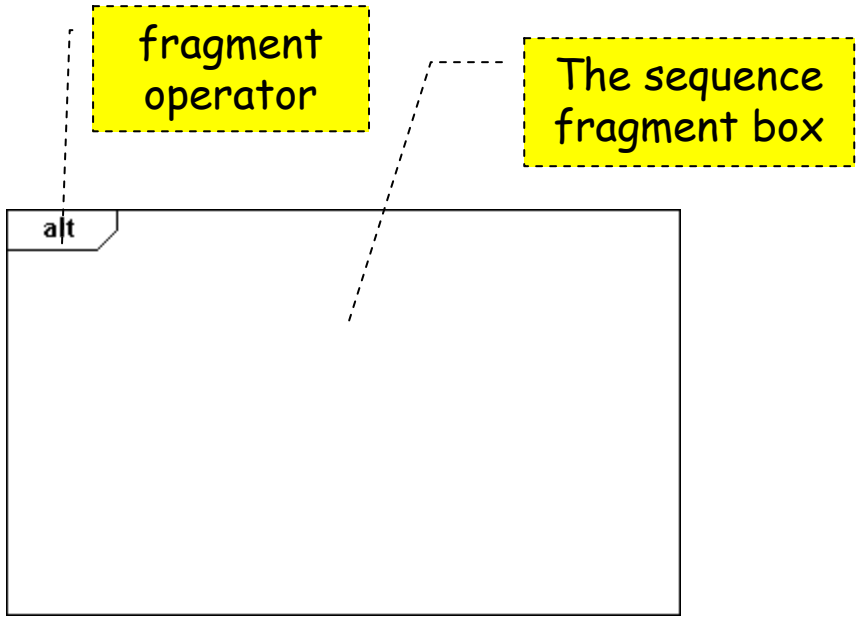
- Sequence Diagrams easily deal with **Timing** information
- **Constraints**: are usually used to show timing constraints on messages. They can apply to the timing of one message or intervals between messages.
  - Label the points of issue and return for a message. Use these labels in expressing timing constraints.
  - This technique also works for message sending that takes time (so arrows are sloping down).
- **Durations**. The duration of activations or the time between messages can be show with construction marks.
  - Metric information in the diagram contribute to representing timing, but this is not recommended (why not?)
  - Although if the line representing the message is horizontal, it is unclear whether it applies to the time the message is sent or received

A Sequence Diagram with Timing



Sequence Fragments

- UML 2.0 introduces Sequence (or Interaction) Fragments
- A sequence fragment is represented as a box, called a **combined fragment**, which encloses a portion of the interactions within a sequence diagram
- The **fragment operator** (in the top left cornet) indicates the type of fragment
- Fragment types: **ref**, **assert**, **loop**, **break**, **alt**, **opt**, **neg**



- Sequence fragments make it easier to create and maintain accurate sequence diagrams

How to Produce Sequence Diagrams

1. **Decide on Context**: Identify **behavior** (or **use case**) to be specified
2. **Identify structural elements**:
    1. Model **objects** (classes)
    2. Model **lifelines**
    3. Model **activations**
    4. Model **messages**
    5. Model **Timing constraints**
3. Refine and elaborate as required

How do interaction diagrams help?

- **Check use cases**: this is the main emphasized aspect
- **Check class can provide an operation**: shows how a class realizes some operation by interacting with other objects
- **Describe design pattern**: parameterizing by class provides a scheme for a generic interaction (part of Software Architecture)
- **Describe how to use a component**: captures how components can interact

Summary

- **Sequence Diagrams**
  - capture some elements of the dynamics of systems
  - Support a number of different activities
  - Describe interaction in some detail, including timing
- **Dimensions**: Objects and Time
- **Basics**: Objects, Lifelines, Activations, Messages, etc.
- **Timing**