Sequence Diagrams

Massimo Felici

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
- Interactions Diagrams
  - Sequence diagrams
  - Interaction overview diagrams
  - Timing diagrams
  - Communication diagrams

What do Sequence Diagrams model?

- capture the interaction between objects in the context of a collaboration
- show object instances that play the roles defined in a collaboration
- show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when
- show elements as they interact over time, showing interactions or interaction instances
- do not show the structural relationships between objects

Slide 2: 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 realises 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 realises 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)

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 class (object) in the interaction is represented by its named icon along the top of the diagram

Slide 3: Participants in a Sequence Diagram

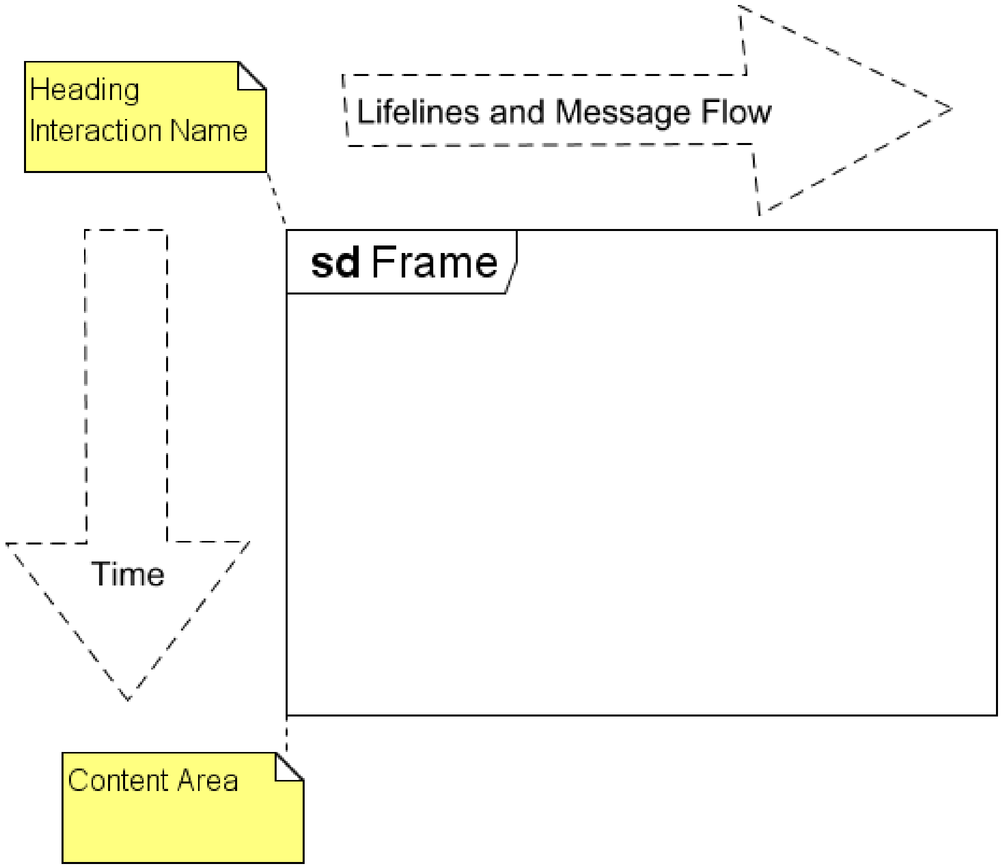
- In UML 1.x, participants were usually software objects (instances of classes) in object-oriented programming sense.

- In UML 2.0, as general modeling language, participants are also at the level of system parts.

Sequence Diagrams

- Frames
- Lifelines
- Messages and Focus Control
- Combined Fragments
- Interaction Occurrences
- States
- Continuations
- Textual Annotation
- Tabular Notation

Frames

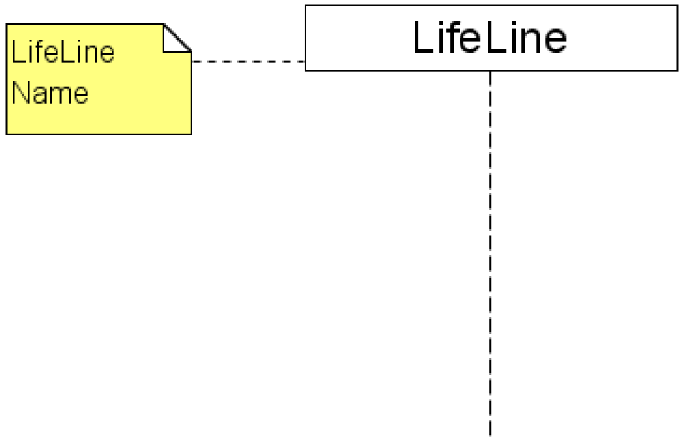

Heading
Interaction Name
Lifelines and Message Flow
sd Frame
Time
Content Area

Slide 5: Sequence Diagrams Dimensions

Time. The vertical axis represents time proceedings (or progressing) down the page. Note that 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.

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.

Lifelines



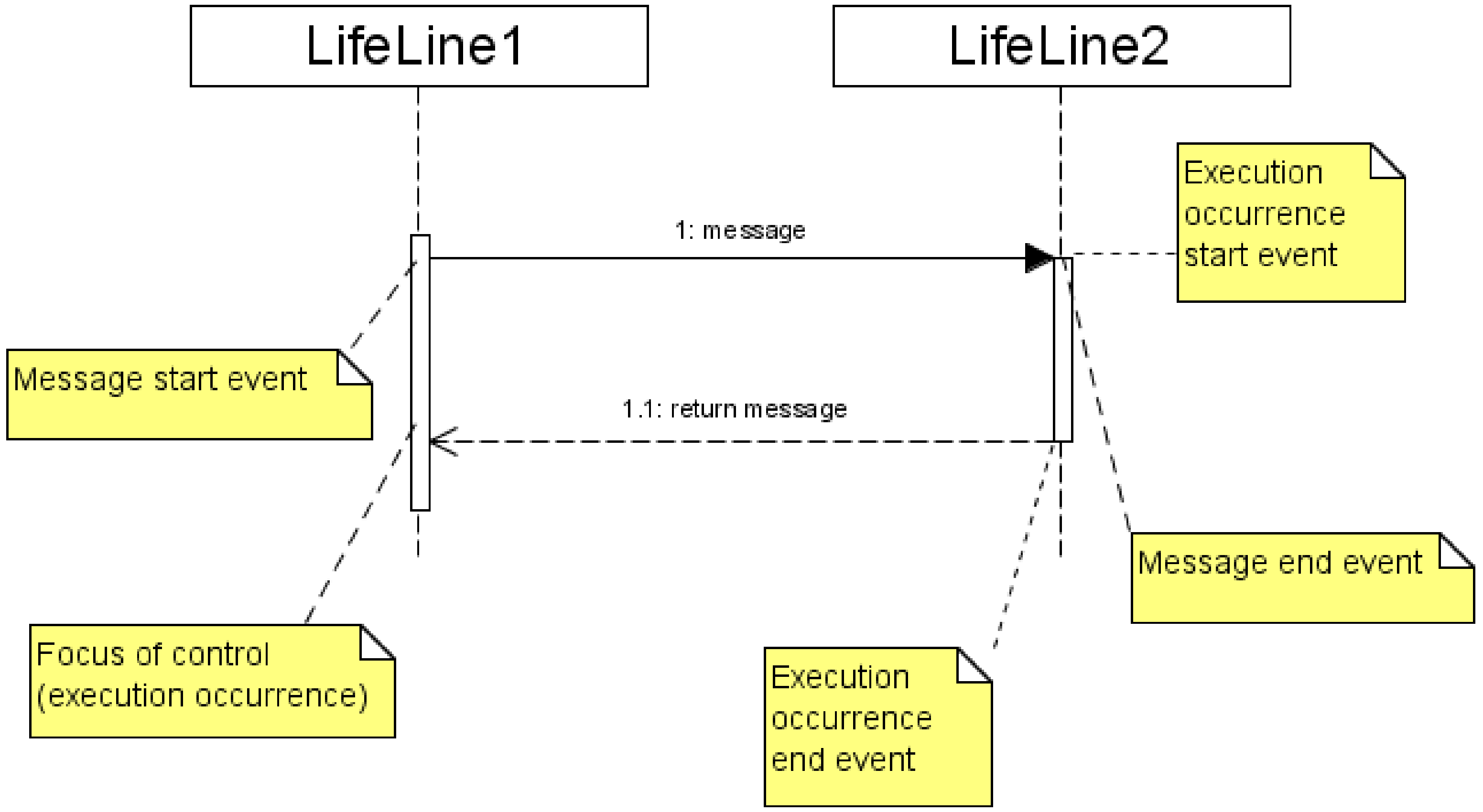
- Sequence diagrams are organised according to time
- Each participant has a corresponding lifeline
- Each vertical dotted line is a lifeline, representing the time that an object exists
- Lifeline name:

  `[connectable-element-name]['['selector']'][:class-name][decomposition]`

Lifelines Names

Syntax	**Explanation**
`seoclecturer`	An object named `secolecturer`
`seoclecturer:Lecturer`	An object named `seoclecturer` of class `Lectuer`
`:Lecturer`	An anonymous object of class `Lecturer`
`lecturer[i]`	The object `lecturer` that is selected by the index value i
`s ref sd3`	A subsystem `s` whose internal interaction is shown in sequence diagram sd3 (decomposition)
`self`	The connectable element that owns the interaction shown in the sequence diagram

Messages and Focus of Control


LifeLine1
LifeLine2
1: message
1.1: return message
Execution occurrence start event
Message start event
Message end event
Focus of control (execution occurrence)
Execution occurrence end event

Slide 8: Messages and Focus of Control

- Focus of control (execution occurrence): an execution occurrence (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.

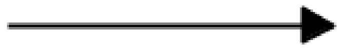
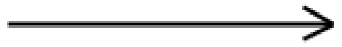
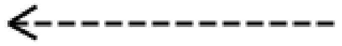
- An Event is any point in an interaction where something occurs.

Messages

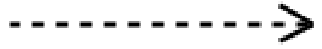
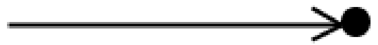
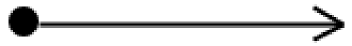
- Messages (or signals) on a sequence diagram are specified using an arrow from the participant (message caller) that wants to pass the message to the participant (message receiver) that is to receive the message

- A Message (or stimulus) is represented as an arrow going from the sender to the top of the focus of control (i.e., execution occurrence) of the message on the receiver's lifeline

Message Type Notations

Message	Description
	**Synchronous:** A synchronous message between active objects indicates wait semantics; the sender waits for the message to be handled before it continues. This typically shows a method call.
	**Asynchronous:** With an asynchronous flow of control, there is no explicit return message to the caller. An asynchronous message between objects indicates no-wait semantics; the sender does not wait for the message before it continues. This allows objects to execute concurrently.
	**Reply:** This shows the return message from another message.

Message Type Notations

Message	Description
	**Create:** This message results in the creation of a new object. The message could call a constructor for a class if you are working with Java, for example.
	**Lost:** A lost message occurs whet the sender of the message is known but there is no reception of the message. This message allows advanced dynamic models to built up by fragments without complete knowledge of all the messages in the system. This also allows the modeler to consider the possible impact of a message's being lost.
	**Found:** A found message indicates that although the receiver of the message is known in the current interaction fragment, the sender of the message is unknown.

Slide 11: Message Type Notations

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 elements lifeline where the destruction is marked with a large $\times$ symbol

Slide 11: Message and Argument Syntax

Message Syntax

```
[attribute=] signal-or-operation-name [(argument)] [:return-value]|*
```

Argument syntax

```
[parameter-name=] argument-value| attribute=out-parameter-name [:argument-value]
| -
```

Slide 11: Types of Communications

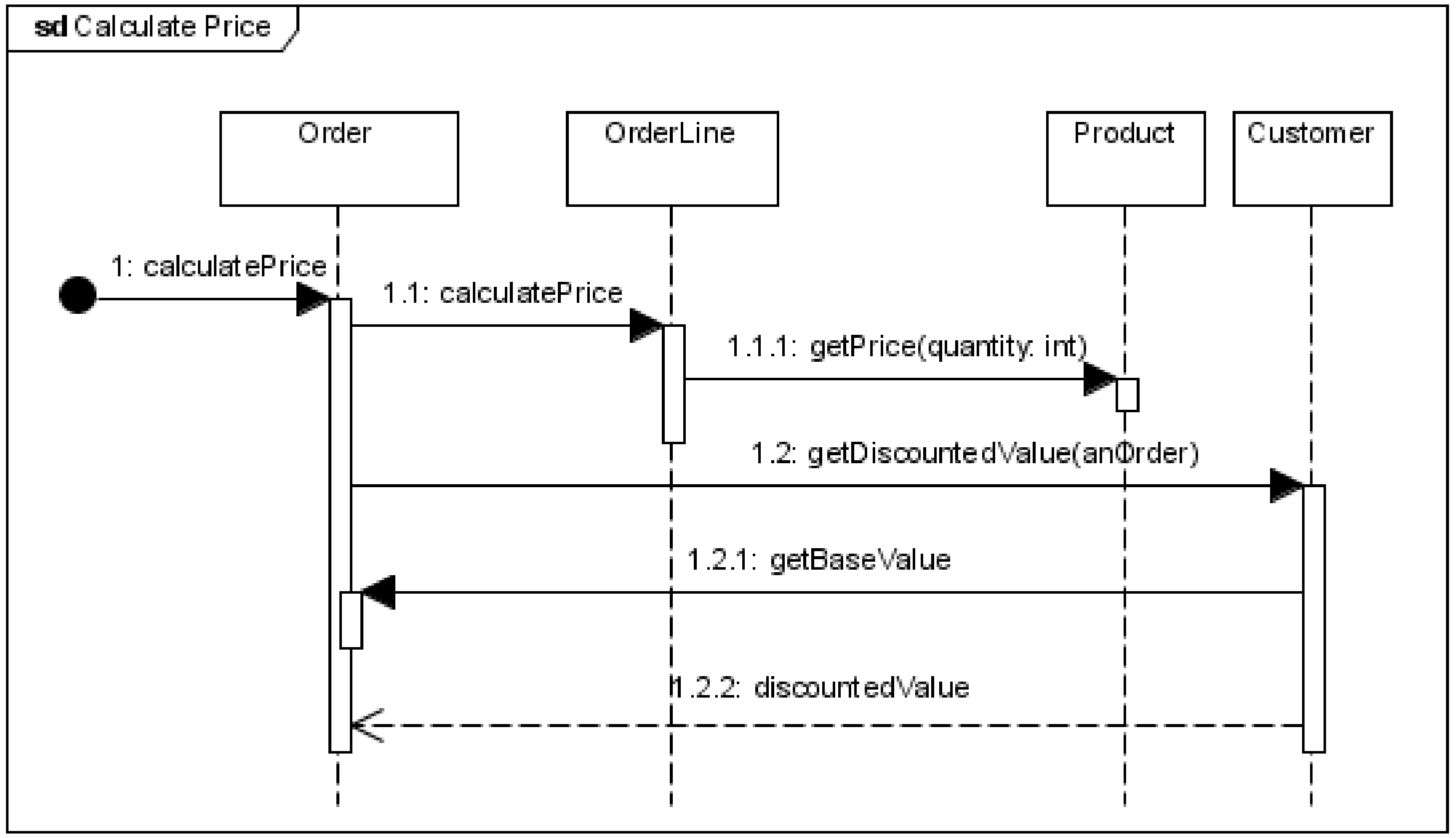
Reflexive Communications: similar to a reflexive association or link, an element may communicate with itself where communication is sent from the element to itself. Sending messages to itself means an object has two activations simultaneously.

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.

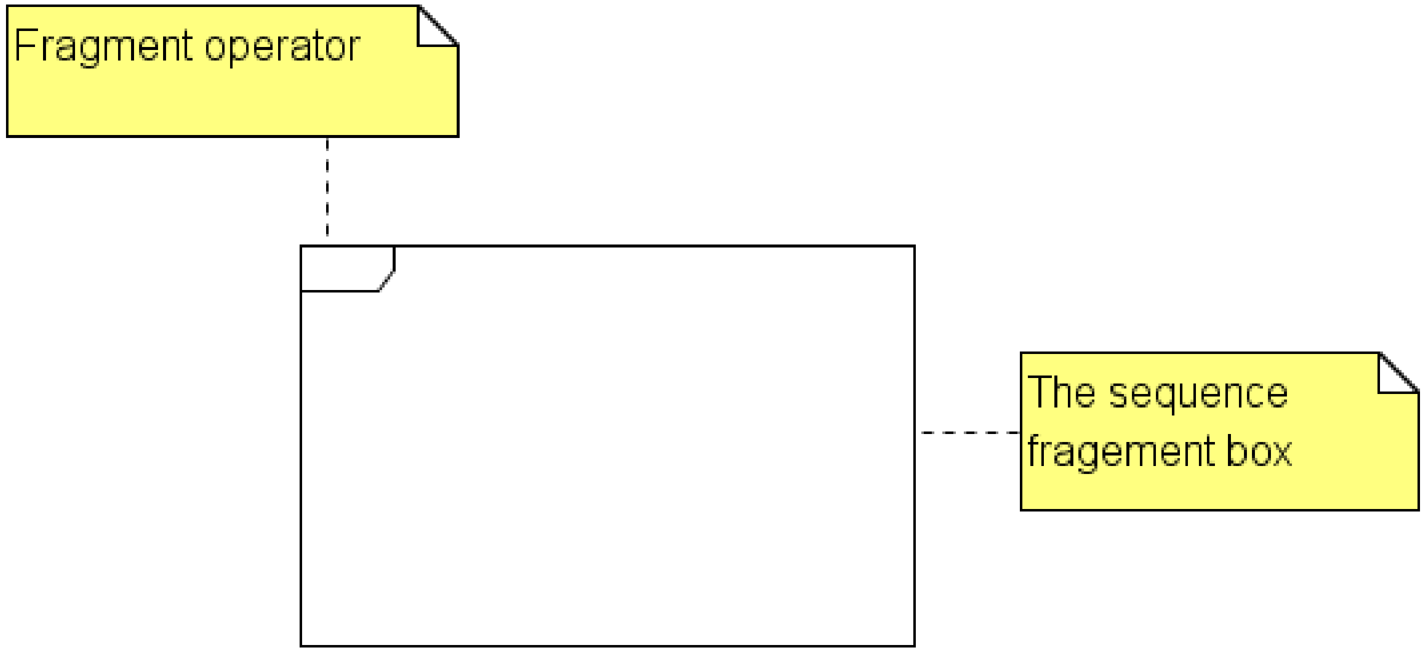
A sequence diagram for distributed control



Sequence Fragments

- UML 2.0 introduces sequence (or interaction) fragments
- Sequence fragments make it easier to create and maintain accurate sequence diagrams
- 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



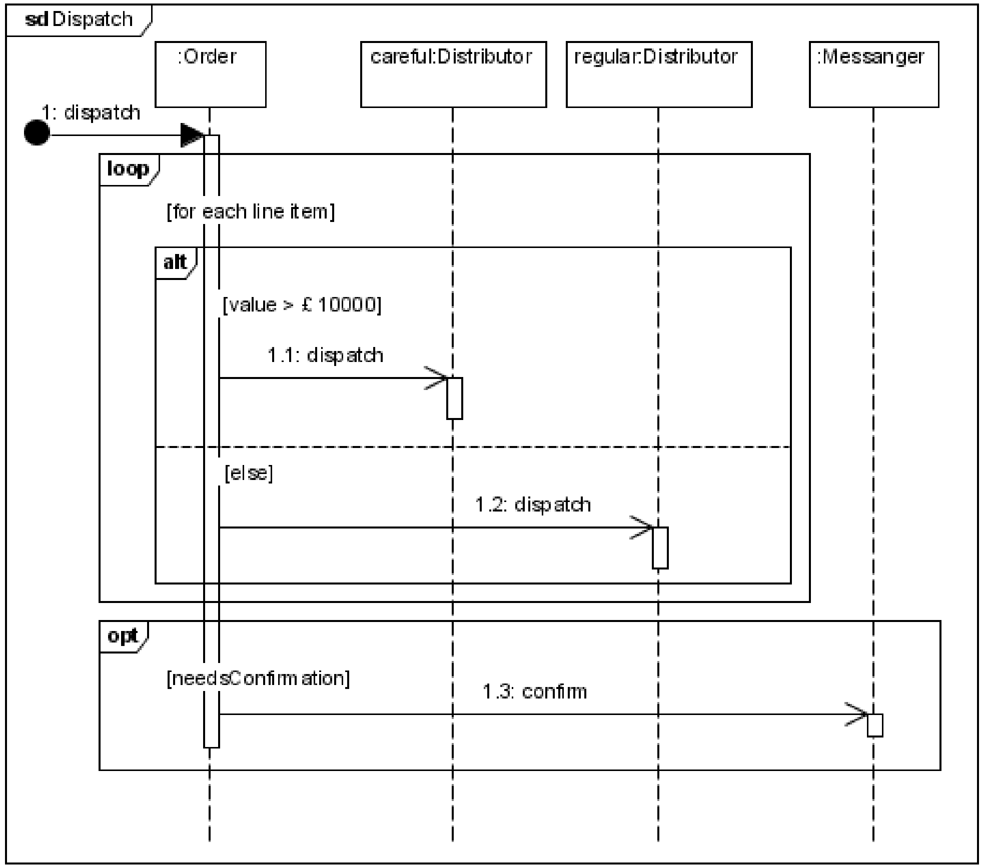
Common Operators for Interaction Frames

Operator	**Meaning**
alt	**Alternative multiple fragments:** only the one whose condition is true will execute.
opt	**Optional:** the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace.
par	**Parallel:** each fragment is run in parallel.
loop	**Loop:** the fragment may execute multiple times, and the guard indicates the basis of iteration.
region	**Critical region:** the fragment can have only one thread executing it at once.
neg	**Negative:** the fragment shows an invalid interaction.
ref	**Reference:** refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value.
sd	**Sequence diagram:** used to surround an entire sequence diagram.

Slide 15: Combined Frames

- It is possible to combine frames in order to capture, e.g., loops or branches.
- Combined fragment keywords: alt, opt, break, par, seq, strict, neg, critical, ignore, consider, assert and loop
- Other ways in UML 2.0 of hiding information are by interaction occurrences and continuations

Interaction Frames



Other Notations

- **States** - it is possible to place states on lifelines (e.g., pre and post conditions)
- **Textual notations** (e.g., comments, time constraints, duration constraints)
- **Tabular notation**

Timing

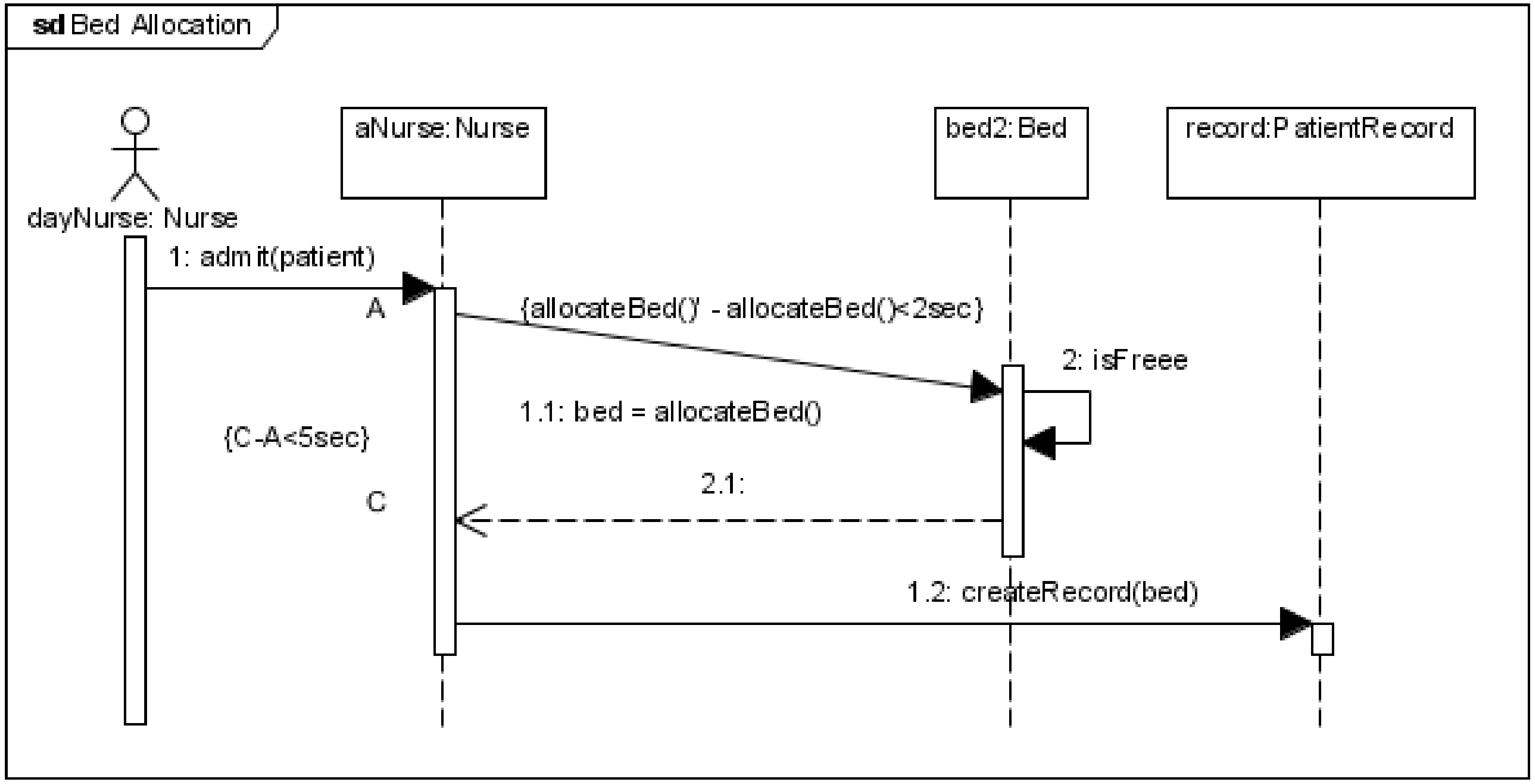
- **Constraints** are usually used to show timing constraints on messages. They can apply to the timing of one message or intervals between messages.
- **Durations** – The duration of activations or the time between messages can be show with construction marks.

Slide 18: Timing

- 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).

- 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

Timing


sd Bed Allocation
dayNurse: Nurse
aNurse: Nurse
bed2: Bed
record:PatientRecord
1: admit(patient)
A
{allocateBed()' - allocateBed()<2sec}
2: isFreee
1.1: bed = allocateBed()
{C-A<5sec}
C
2.1:
1.2: createRecord(bed)

How to Produce Sequence Diagrams

1. Decide on Context: Identify behavior (or use case) to be specified

2. Identify structural elements:

   (a) Model objects (classes)
   (b) Model lifelines
   (c) Model activations
   (d) Model messages
   (e) Model Timing constraints

3. Refine and elaborate as required

When to Use Sequence Diagrams

- You should use sequence diagrams when you want to look at the behaviour of several objects within a single use case.
- Sequence diagrams are good at showing collaborations among the objects.
- They are not so good at precise definition of behaviour.

How do Interaction Diagrams help?

- Check use cases
- Check class can provide an operation – showing how a class realizes some operation by interacting with other objects
- Describe design pattern – parameterising by class provides a scheme for a generic interaction (part of Software Architecture)
- Describe how to use a component – capturing how components can interact

Required Readings

- UML course textbook, Chapter 9 on Interaction Sequence Diagrams

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