



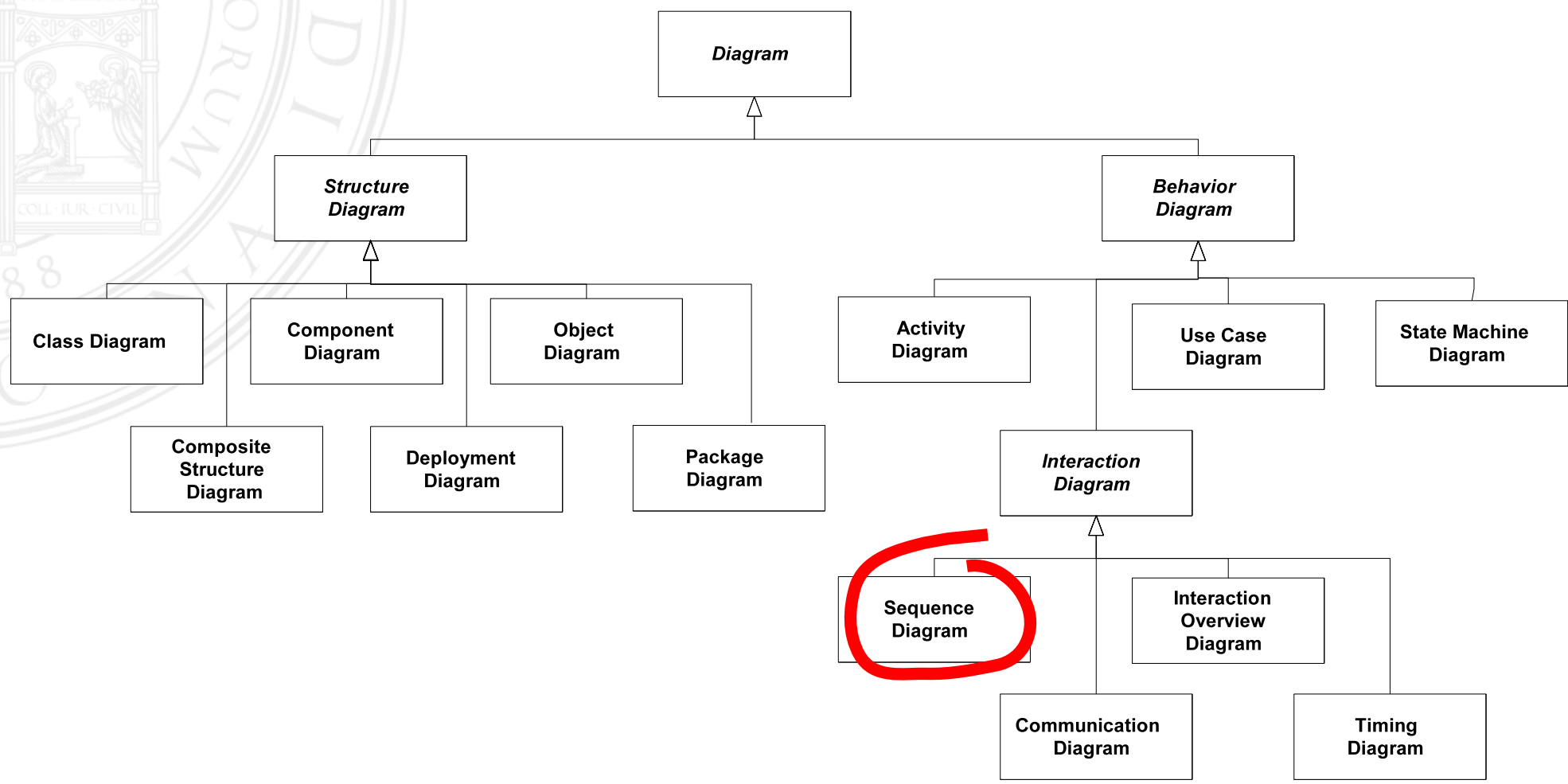
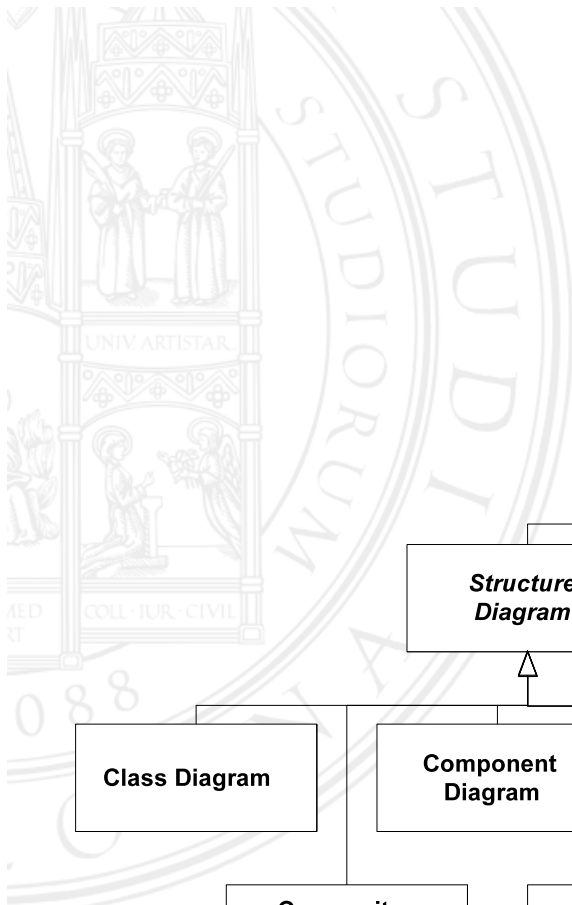
# Ingegneria del Software

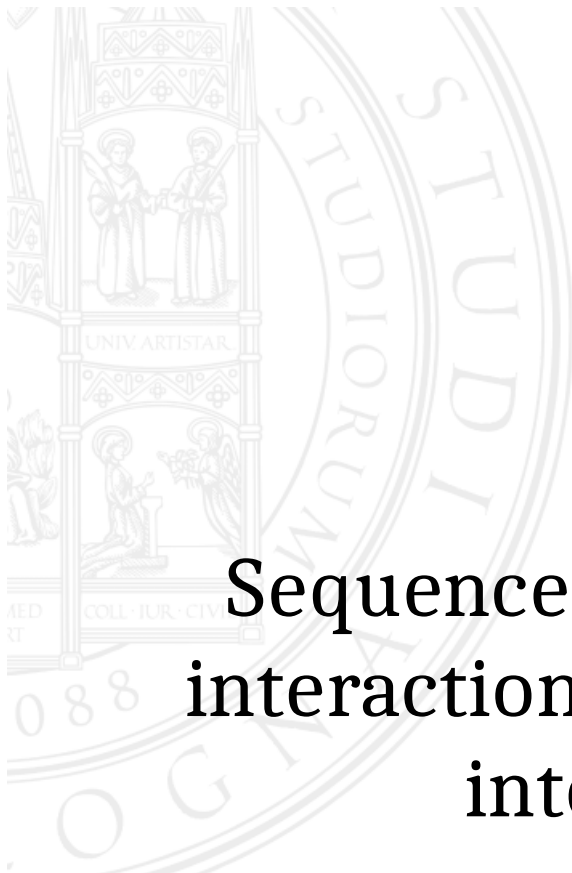
## Corso di Laurea in Informatica per il Management

# UML: Interaction diagrams

**Davide Rossi**  
Dipartimento di Informatica  
Università di Bologna





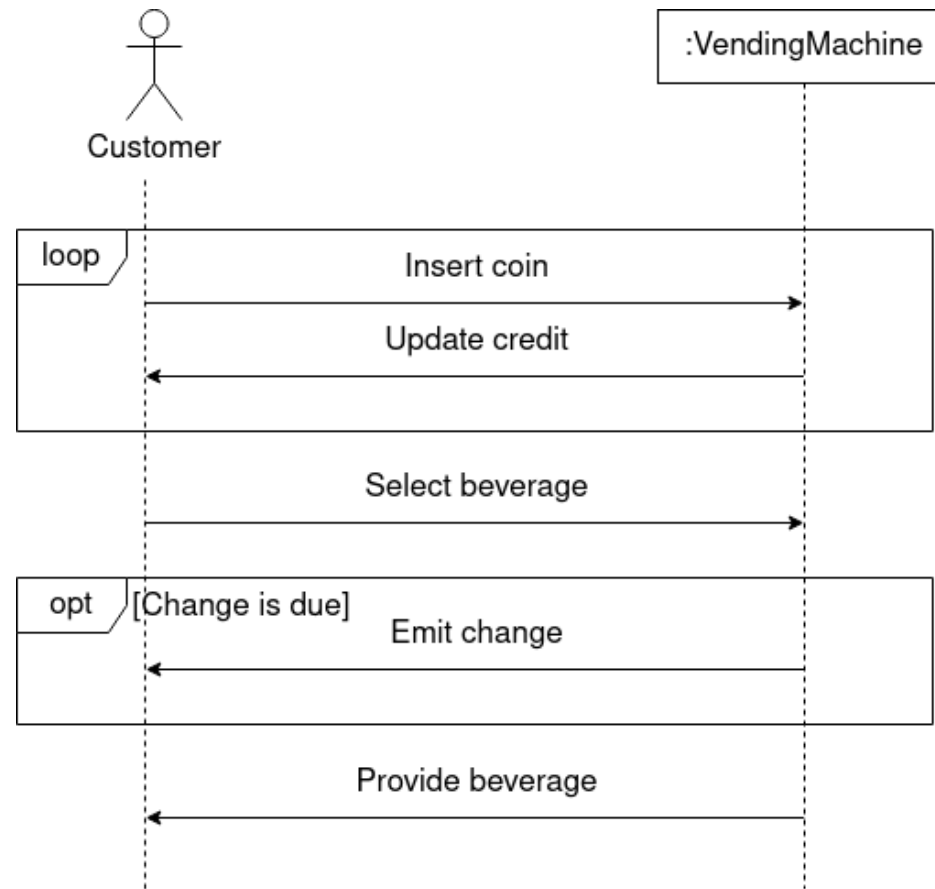


# Sequence diagrams

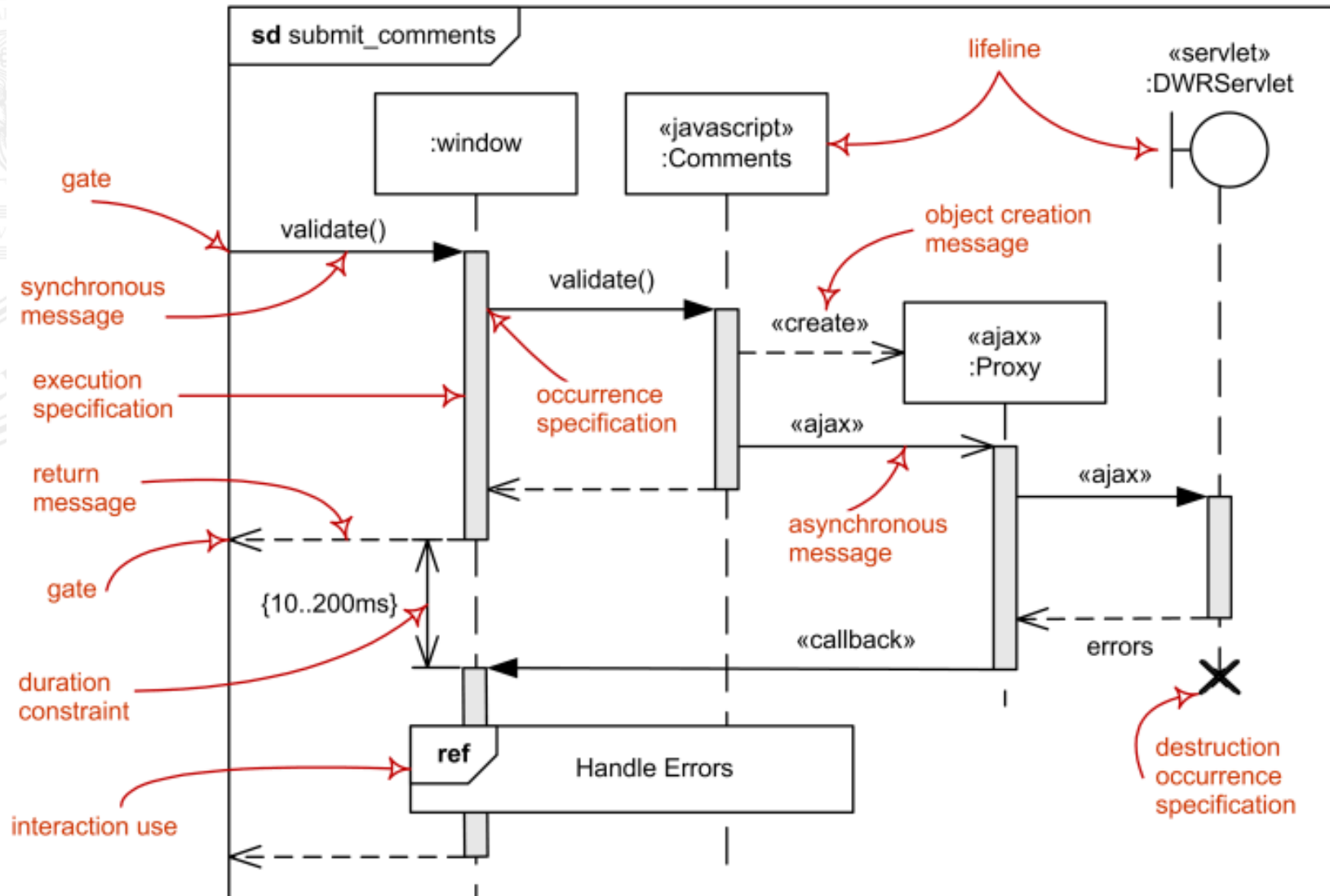
Sequence diagrams are the most common kind of interaction diagrams; they focus on the message interchange between a number of lifelines.

Sequence diagrams describe interactions by focusing on the sequence of messages that are exchanged and their corresponding occurrence specifications on the lifelines.

# Example: a system sequence diagram

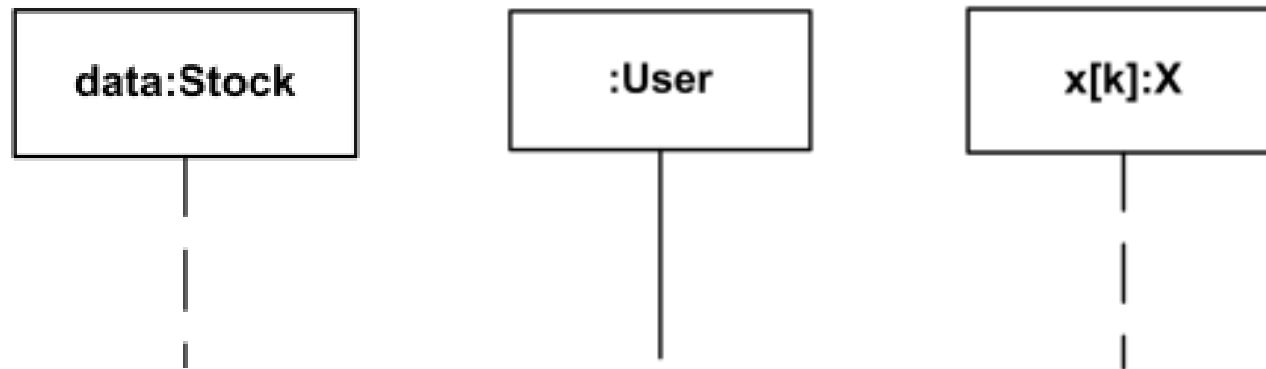


# Elements of sequence diagrams



# Lifeline

A **lifeline** is a named element which represents an individual participant in the interaction. While parts and structural features may have multiplicity greater than 1, lifelines represent only one interacting entity.



# Message

A **message** is a named element that defines one specific kind of communication between lifelines of an interaction. The message specifies not only the kind of communication, but also the sender and the receiver. Sender and receiver are normally two occurrence specifications (messages endpoints).

# Messages by Action Type

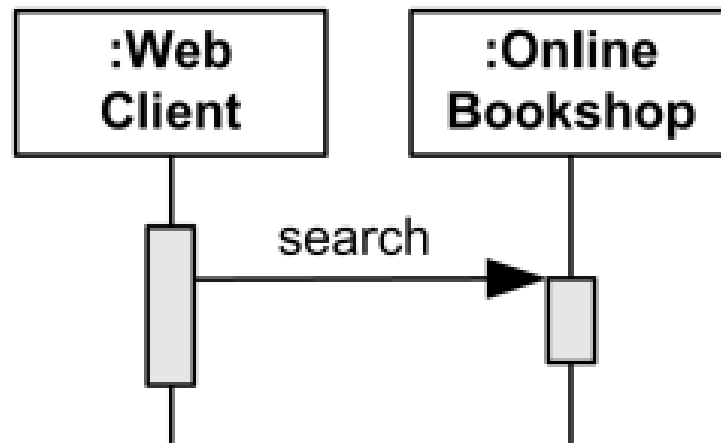
A message reflects either an operation call and a start of execution or a sending and reception of a signal. Depending on the type of action that was used to generate the message, message could be one of:

- synchronous call/asynchronous call
- asynchronous signal
- reply
- create
- delete



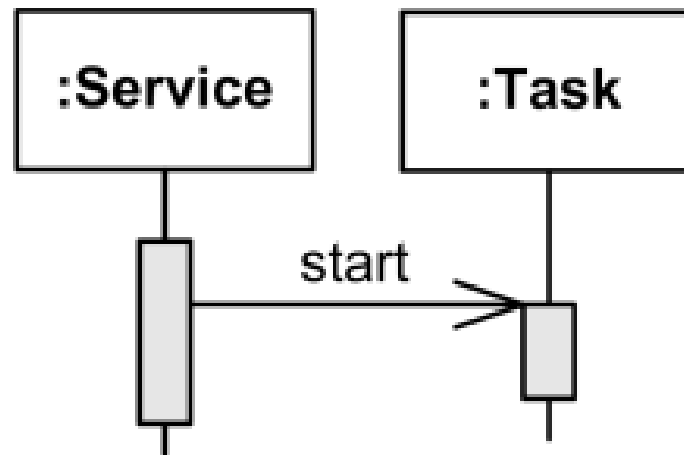
# Synchronous Call

**Synchronous calls** typically represent operation call - send message and suspend execution while waiting for response. Synchronous call messages are shown with filled arrow head.



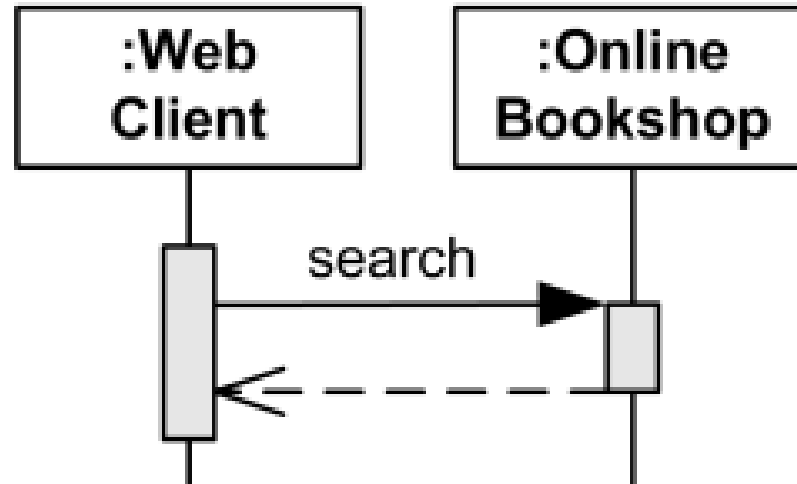
# Asynchronous Call

**Asynchronous calls** - send message and proceed immediately without waiting for return value. Asynchronous messages have an open arrow head.



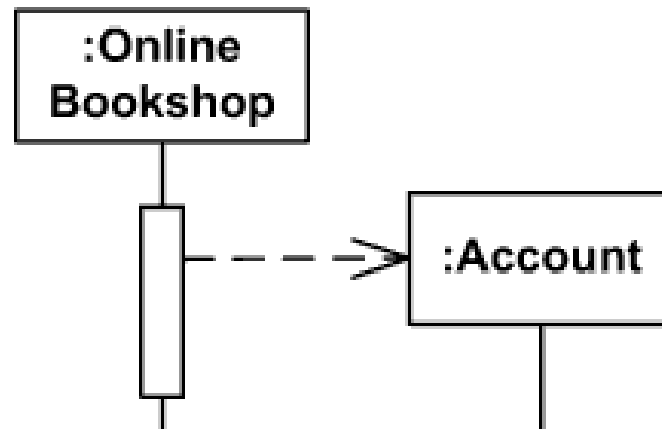
# Reply Message

A **reply message** to an operation call is shown as a dashed line with open arrow head (looks similar to creation message).



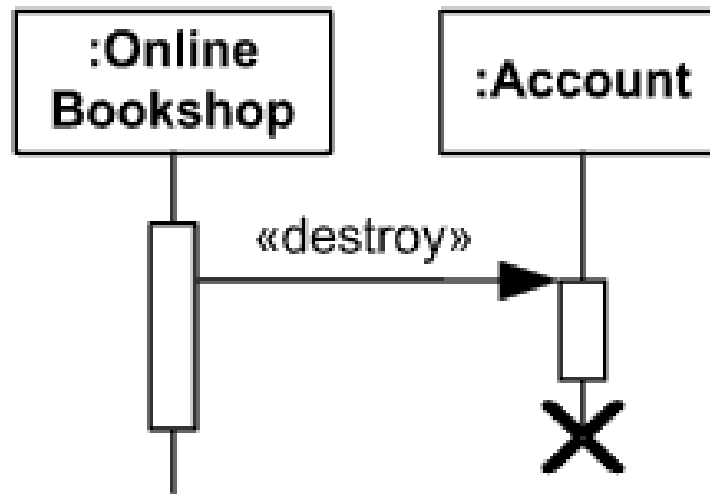
# Create Message

A **create message** is sent to a lifeline to create itself. It is common practice to send a create message to a (still) nonexisting object to create itself.

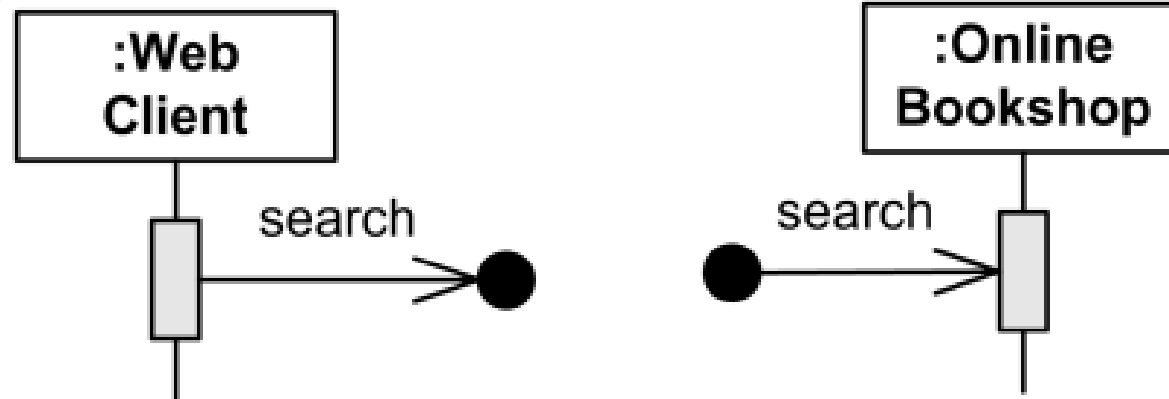


# Delete Message

A **delete message** is sent to terminate another lifeline. The lifeline usually ends with a cross in the form of an X at the bottom denoting a destruction occurrence.

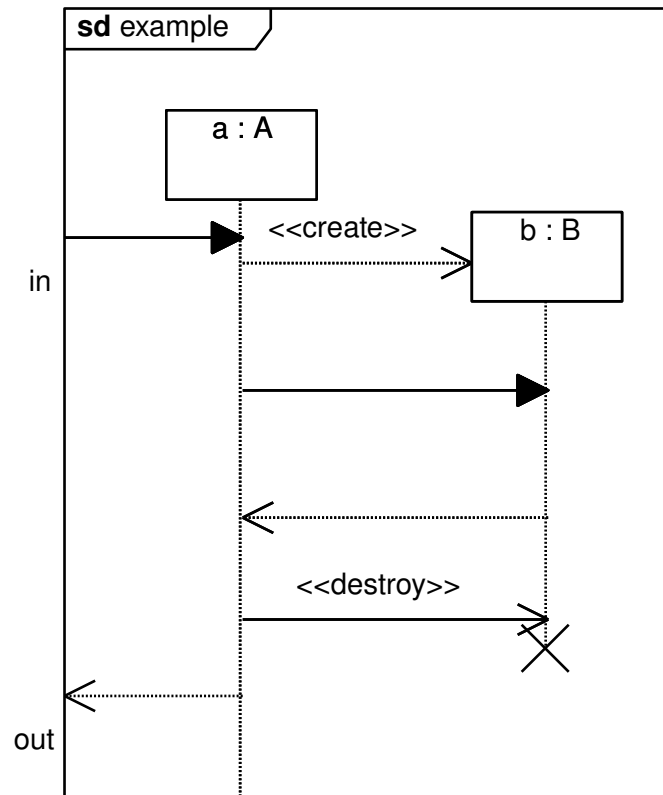


# Lost & founds



# Gate

A **gate** is a message end, a connection point for relating a message outside of an interaction fragment with a message inside the interaction fragment.



# Interaction Fragment

An **interaction fragment** is a named element representing the most general interaction unit. Each interaction fragment is conceptually like an interaction by itself. There is no general notation for an interaction fragment. Its subclasses define their own notation.

Examples of interaction fragments are: occurrence, execution, state invariant, combined fragment, interaction use.





# Occurrence

An **occurrence** (complete name: occurrence specification) is an interaction fragment which represents a moment in time (event) at the beginning or end of a message or at the beginning or end of an execution.

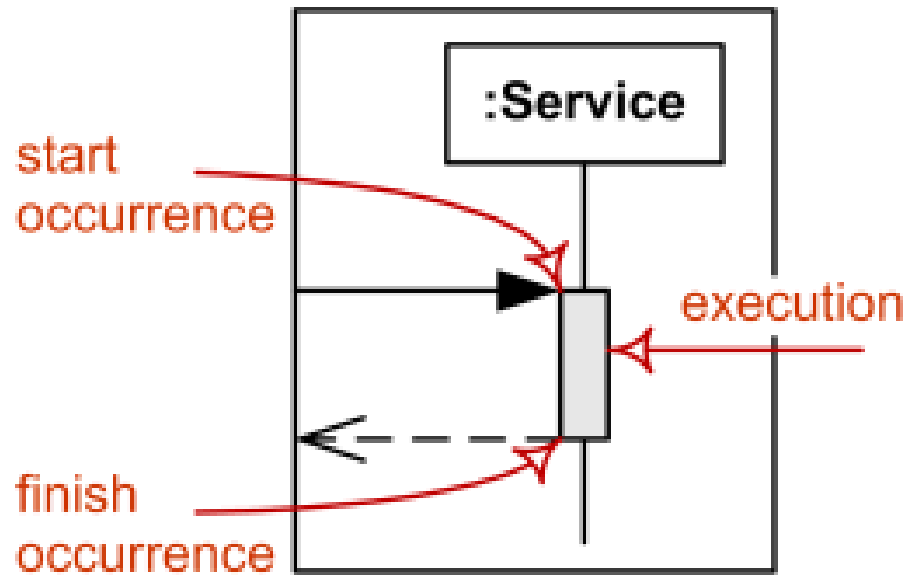
An occurrence specification is one of the basic semantic units of interactions. The meanings of interactions are specified by sequences of occurrences described by occurrence specifications.

# Execution

An **execution** (full name - execution specification, informally called *activation*) is an interaction fragment which represents a period in the participant's lifetime when it is:

- executing a unit of behavior or action within the lifeline,
- sending a signal to another participant,
- waiting for a reply message from another participant

# Execution



# Combined Fragment

A **combined fragment** is an interaction fragment which defines a combination of interaction fragments. A combined fragment is defined by an interaction operator and corresponding interaction operands. Through the use of combined fragments the user will be able to describe a number of traces in a compact and concise manner.

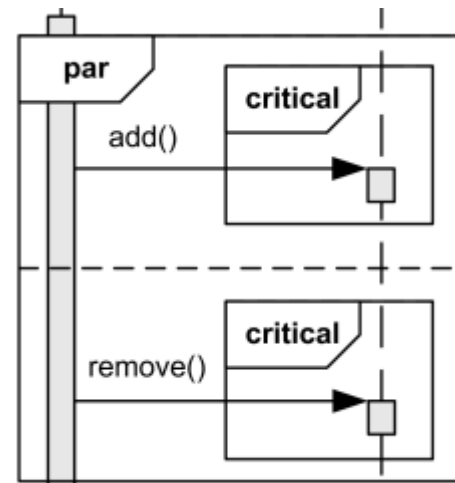
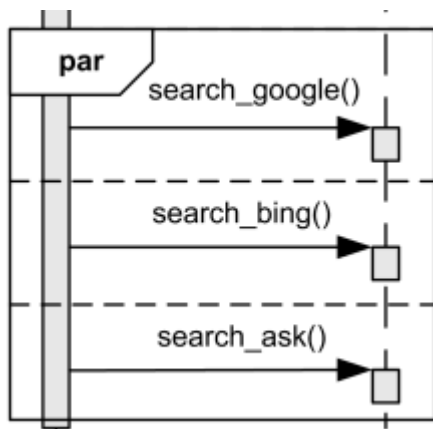
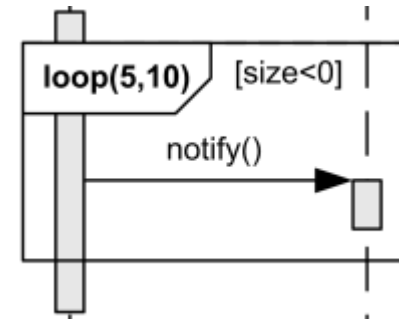
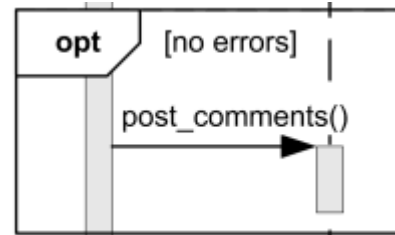
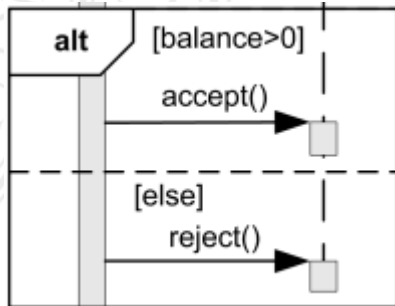
Combined fragment may have interaction constraints (guards).

# Combined Fragment

Interaction operator could be one of:

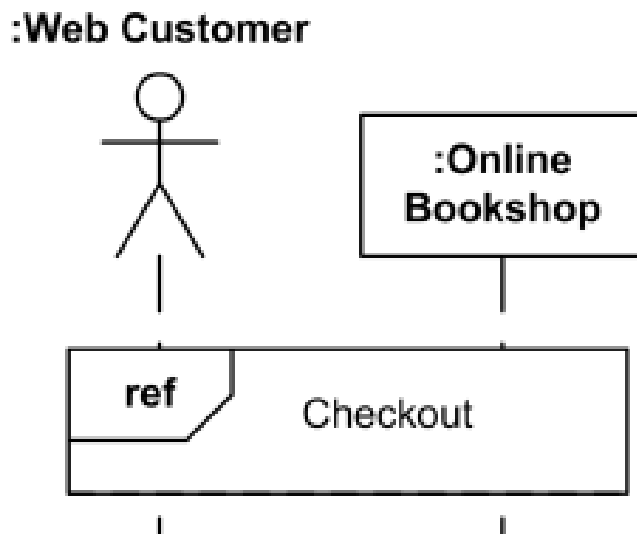
- alt – alternatives
- opt - option
- loop - iteration
- break - break
- par - parallel
- strict – strict sequencing
- seq - weak sequencing
- critical - critical region
- ignore - ignore
- consider - consider
- assert - assertion
- neg - negative

# Combined Fragment



# Interaction Use

An **interaction use** is an interaction fragment which allows to use (or call) another interaction. Large and complex sequence diagrams could be simplified with interaction uses. It is also common reusing some interaction between several other interactions.



# SD do's and don'ts

- Don't over-generalize sequences
- When using SD for analysis
  - A use case can be described by more than one SD
  - Message's action type can be decided at design time
  - No messages between lifelines belonging to elements of the system

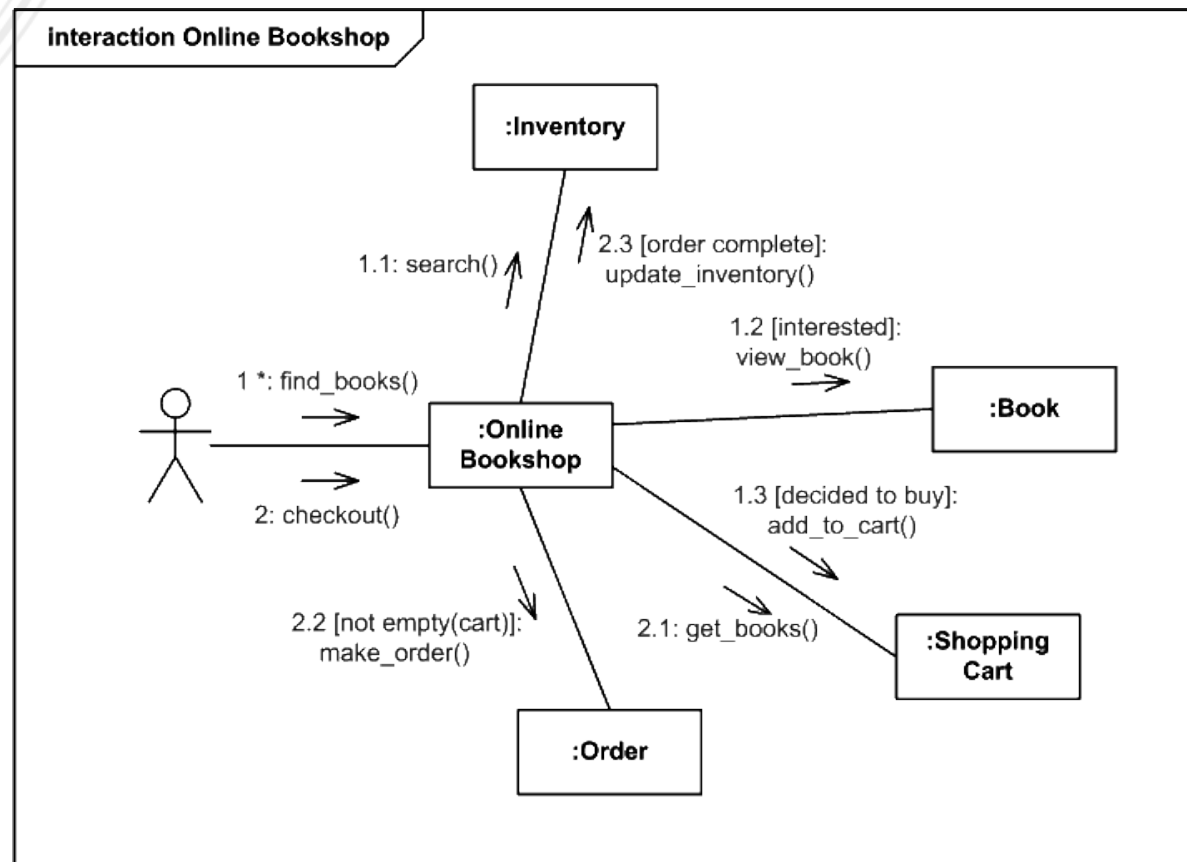


# Communication Diagrams

- A communication diagram shows the interactions between lifelines using a free-form arrangement.
- Communication diagrams can be converted into sequence diagrams (but NOT the other way around in UML 2).
- It is assumed that messages are received in the same order as they are generated.

# Communication Diagrams

- A communication diagram can contain frames, lifelines and messages.



# Messages

- The notation for messages in communication diagrams follow the same rules used in sequence diagrams.
- In order to understand the dynamic evolution of the system, messages have a sequence expression.

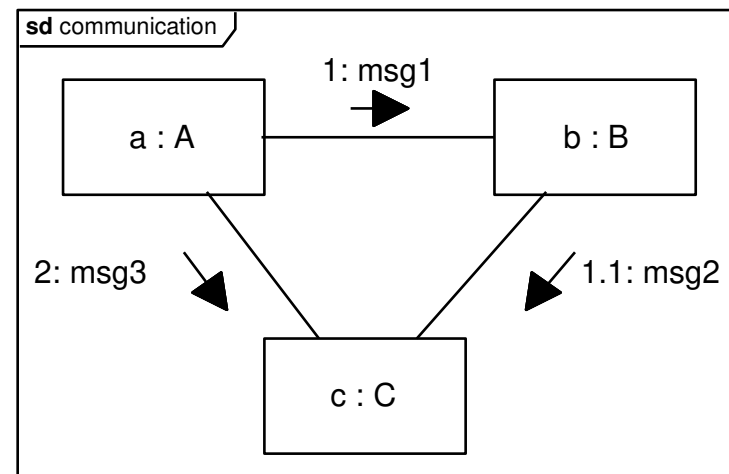
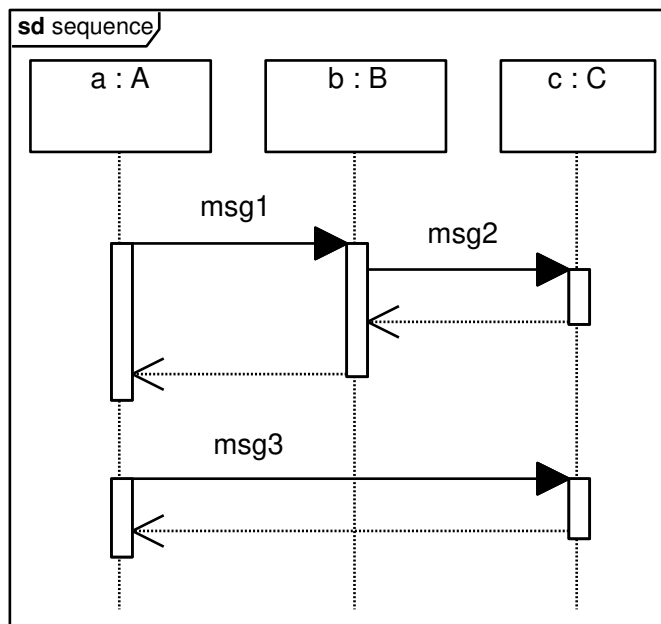
# Sequence expressions

Sequence-expression ::=

sequence-term '.' . . . ':' message-name

Sequence-term ::= [integer[name]][recurrence]

Sequence terms are used to represent the nesting of messages within an interaction.



# Concurrency and recurrence

- Sequence-term ::= [integer[name]]  
[recurrence]

Messages that differ only for the name part are considered concurrent

- recurrence ::= branch | loop

branch ::= '[' guard ']

Guards specify conditions for the message to happen

2.3b [x>y]: draw()

# Acknowledgments

These slides include work by Paolo Ciancarini, Sara Zuppiroli and Gian Piero Favini.

Some diagrams and texts have been taken from <http://www.uml-diagrams.org>.