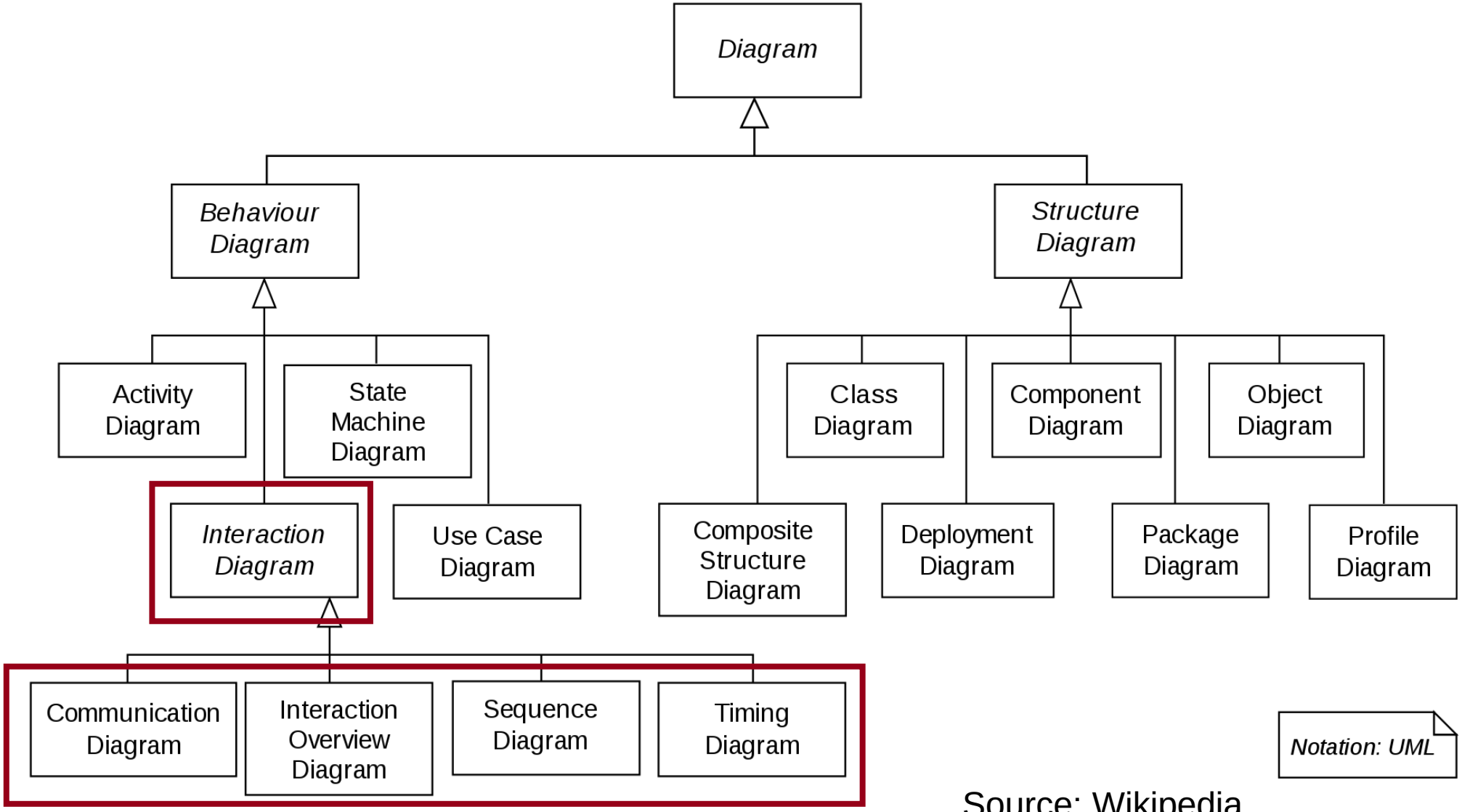
UML Interactions

HUSZERL Gábor
huszerl@mit.bme.hu

MŰEGYETEM 1782
BME VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
Méréstechnika és Információs Rendszerek Tanszék
ftsrg Critical Systems Research Group

UML Diagram Types



Source: Wikipedia

Modelling Scenarios

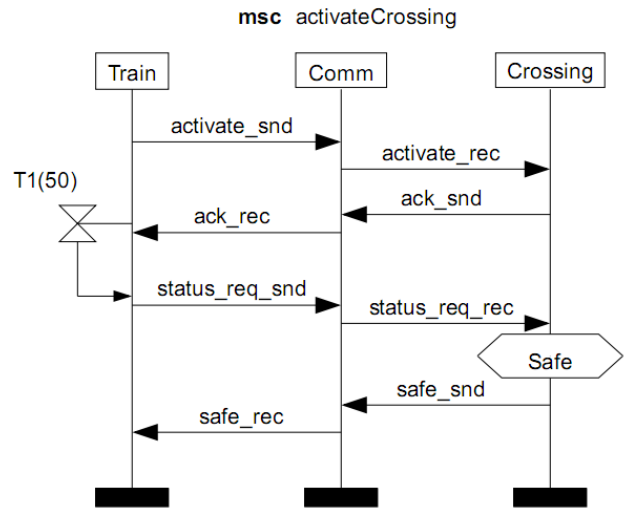
> „**scenario**: step-by-step description of a series of events that occur concurrently or sequentially

- Introducing Scenarios
  - Communication between the objects/roles
  - The order and kind of the messages is important (not the data)

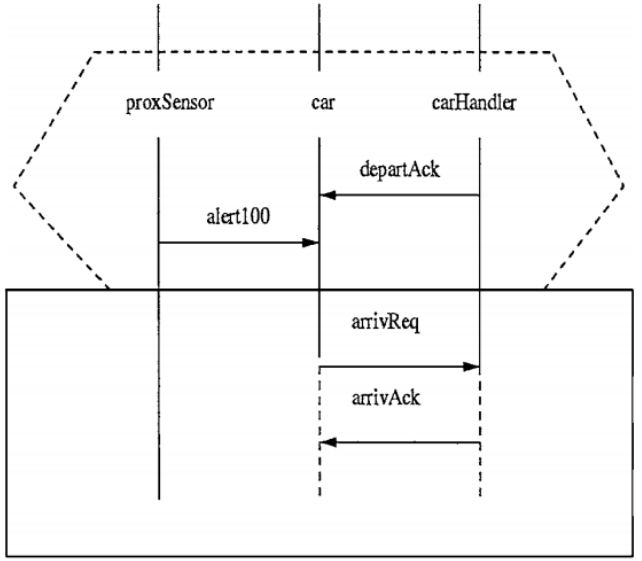
- *“Interactions do not tell the complete story”*
  - Partial behaviour, not complete
  - Describing just a few more important examples or test cases

Scenario Describing Languages

- Scenario Describing Languages
  - Message Sequence Chart (MSC) (1993-)
  - Live Sequence Chart (LSC)
  - …



- Can be used for various purposes:
  - Method invocation chain inside a program
  - Messaging between components
  - Messages of the nodes of a distributed system
  - Network protocols



How They Can Be Used in Software Systems?

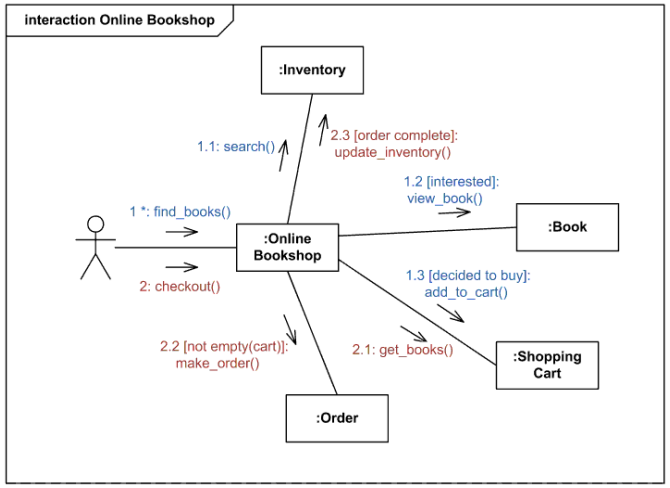
- Refining use cases
  - Presenting typical and alternative traces (Actor ↔ System)
- Describing protocols
  - What can be the order of messages on the interfaces
  - May contain simple logic/control, too
- Defining test cases
  - What is the expected/prohibited behaviour? What is to be checked?
- Describing internal invocation chains
  - While delivering a functionality, who invokes whom, who accesses whom, how
- Visualising traces (debug, understanding, …)

Types of Interaction Diagrams

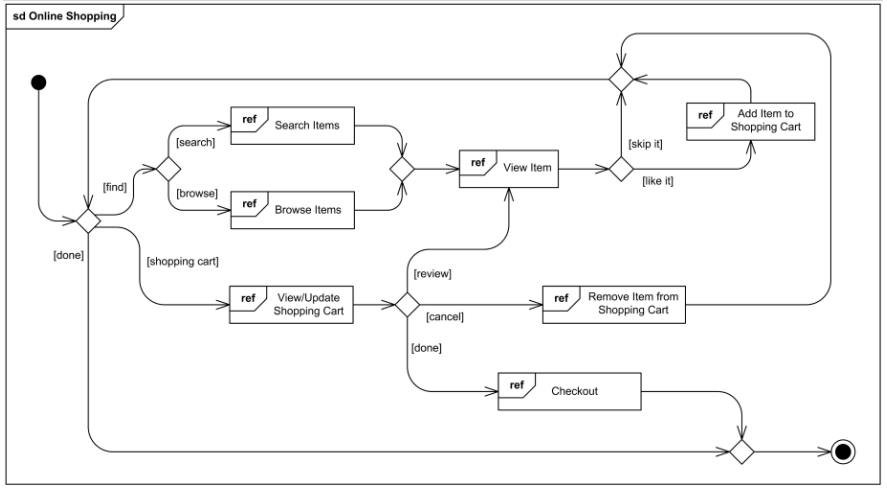
- **Sequence Diagram**
  - Sequences of messages between roles
  - May be extended by decisions, sequencing, time, …
- **Communication Diagram**
  - Following a specific, simple message call
- **Interaction Overview Diagram**
  - Control flow between the different interactions (high-level)
- **Timing Diagram**
  - Relations between timing, messages and state transitions

Types of Interaction Diagrams

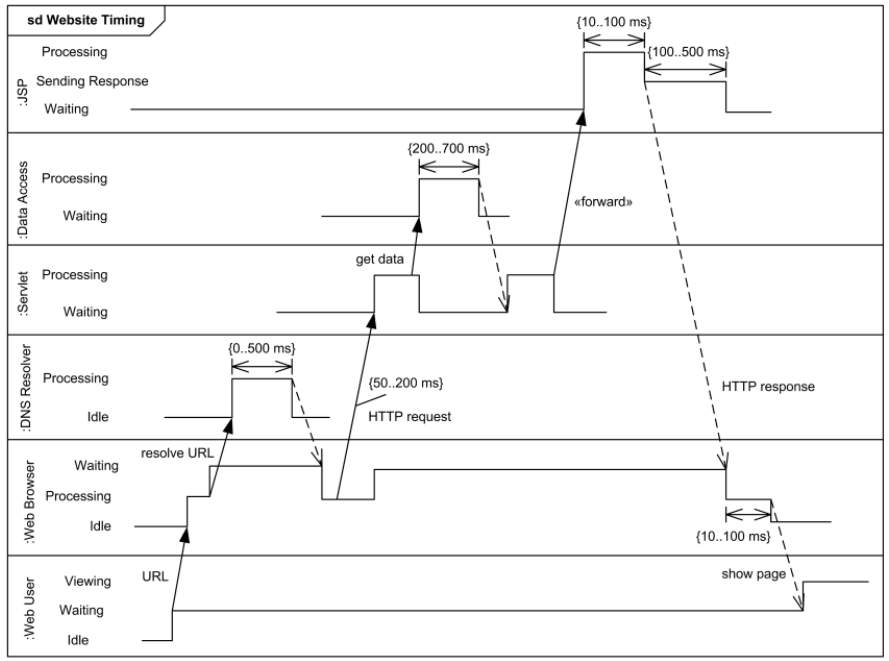
Communication



Interaction Overview



Timing

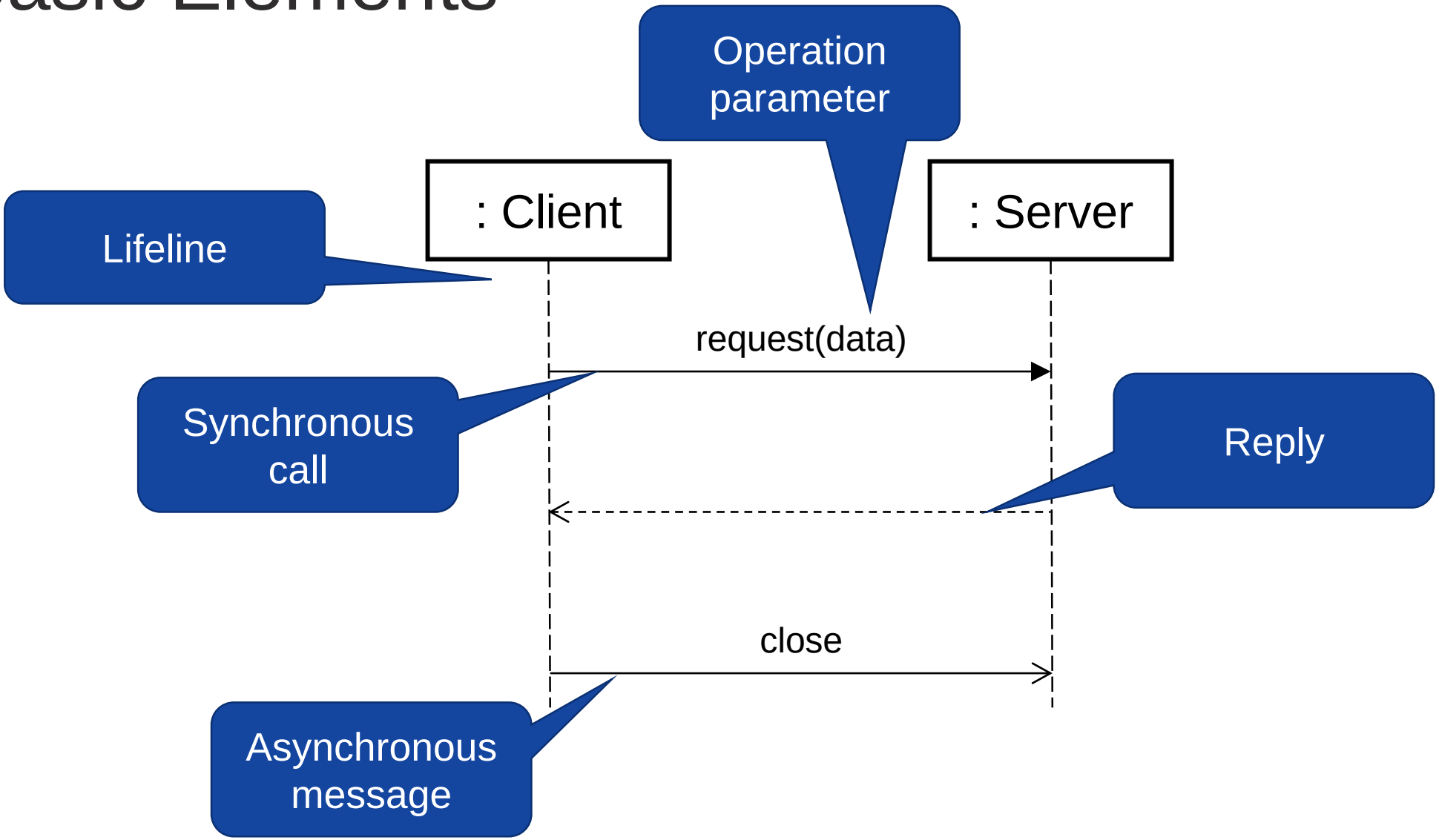


Source: http://www.uml-diagrams.org

Set of Elements

Message, Lifeline, CombinedFragment

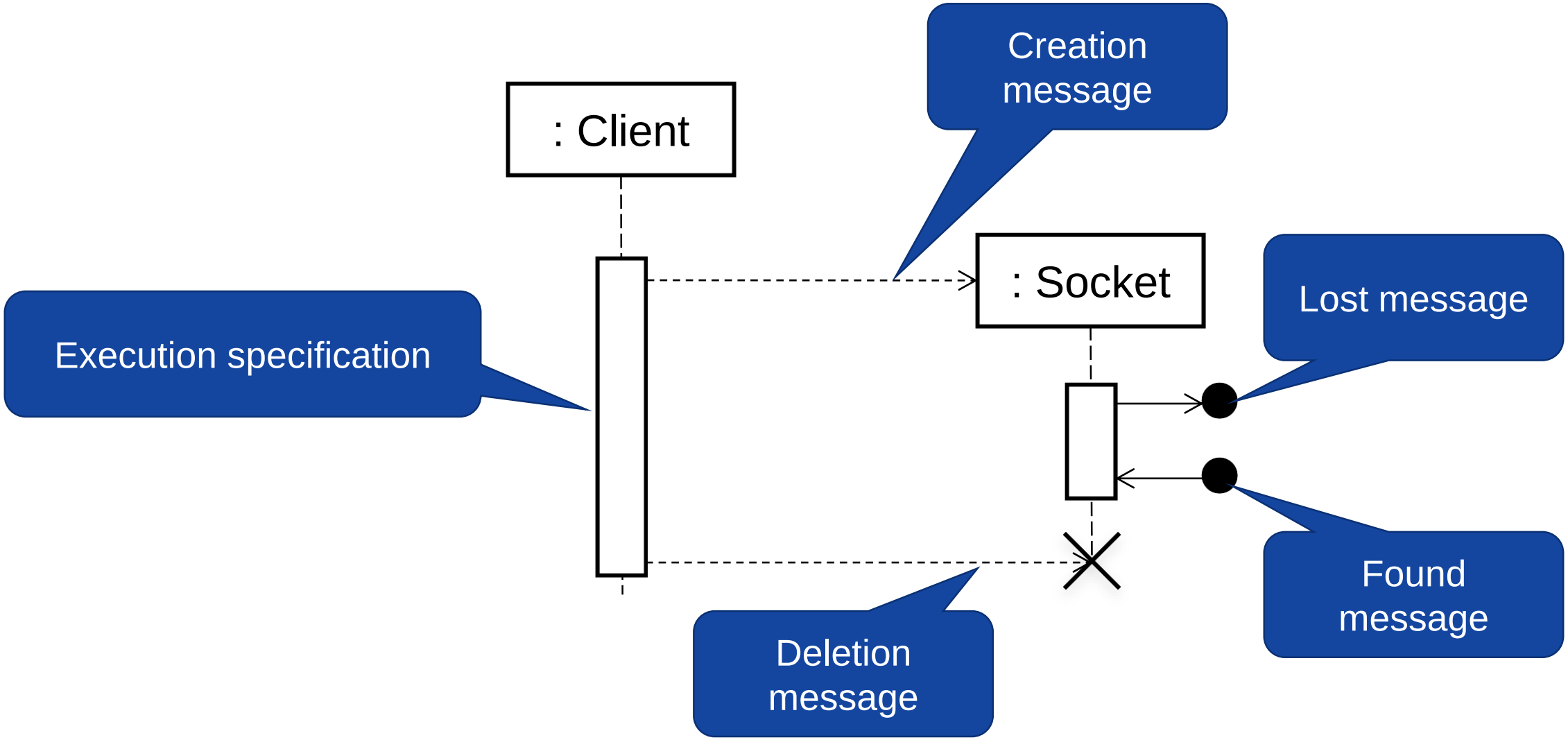
Basic Elements



Basic Elements

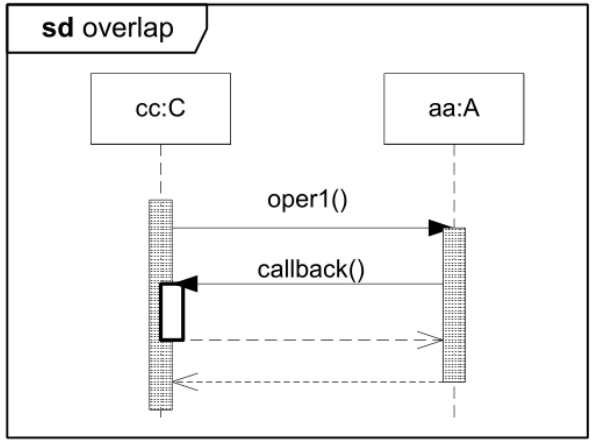
- **Lifeline**
  - ~~**Instances**~~ **roles** in the individual interactions
  - Represents a so called *ConnectableElement* (not an object!)
  - May have name and/or type (but at least one of them)
- **Message**
  - **Synchronous** calls
    - Blocks the caller, may have a reply message
  - **Asynchronous** messages (async. calls or signals)
  - Messages may have arguments
    - A dash (“–”) represents an arbitrary value (wildcard)
    - (Describing arguments is optional, they are often out of focus)

Life Cycle and Message Types



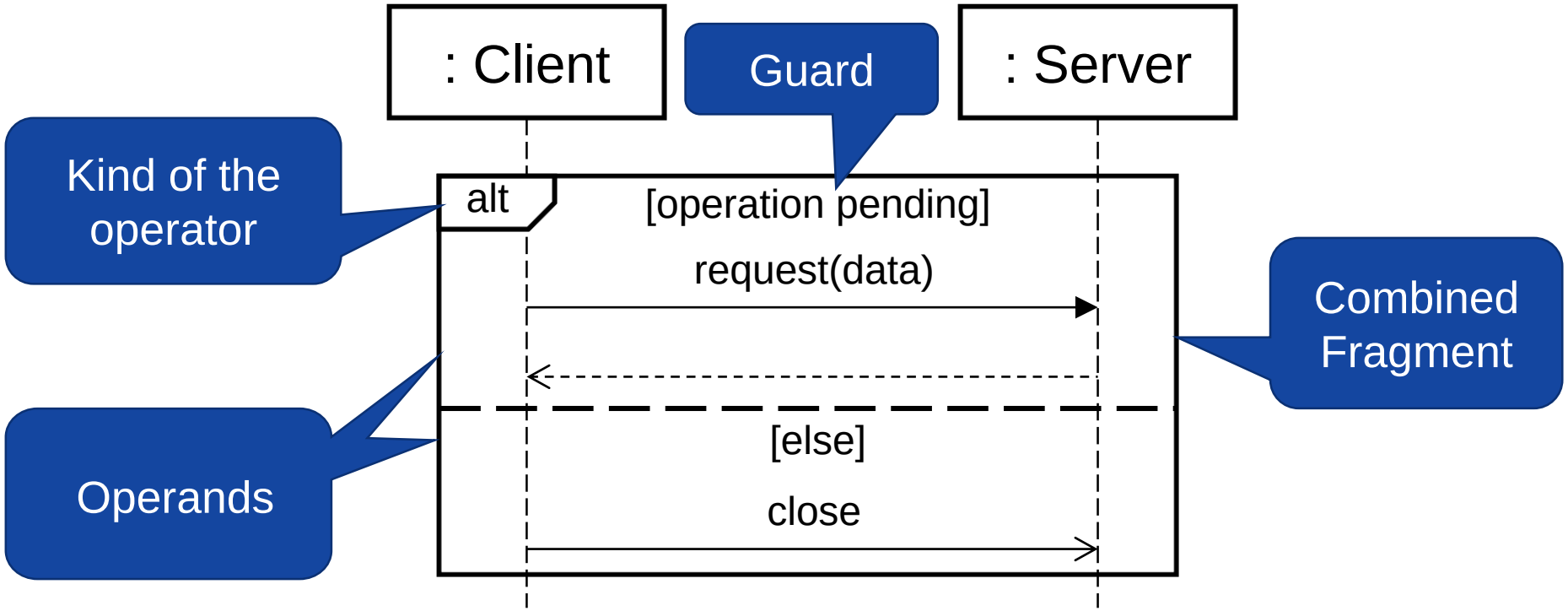
Life Cycle and Message Types

- **Create** and **delete** messages
  - Creating or deleting a different actor
- **Execution specification**
  - Shows when is an actor **active**
    - Does something or wait for a reply
  - Not mandatory, but many tools uses it by default
    - Useful for a single thread, misleading in case of multiple actors
  - Can be bound to an action or behaviour
- **Lost & found messages**
  - Source and target is not know or **not relevant**



Combined Fragments

- Operators for combined scenarios
  - May have one or more operands
  - Each operand may have a guard

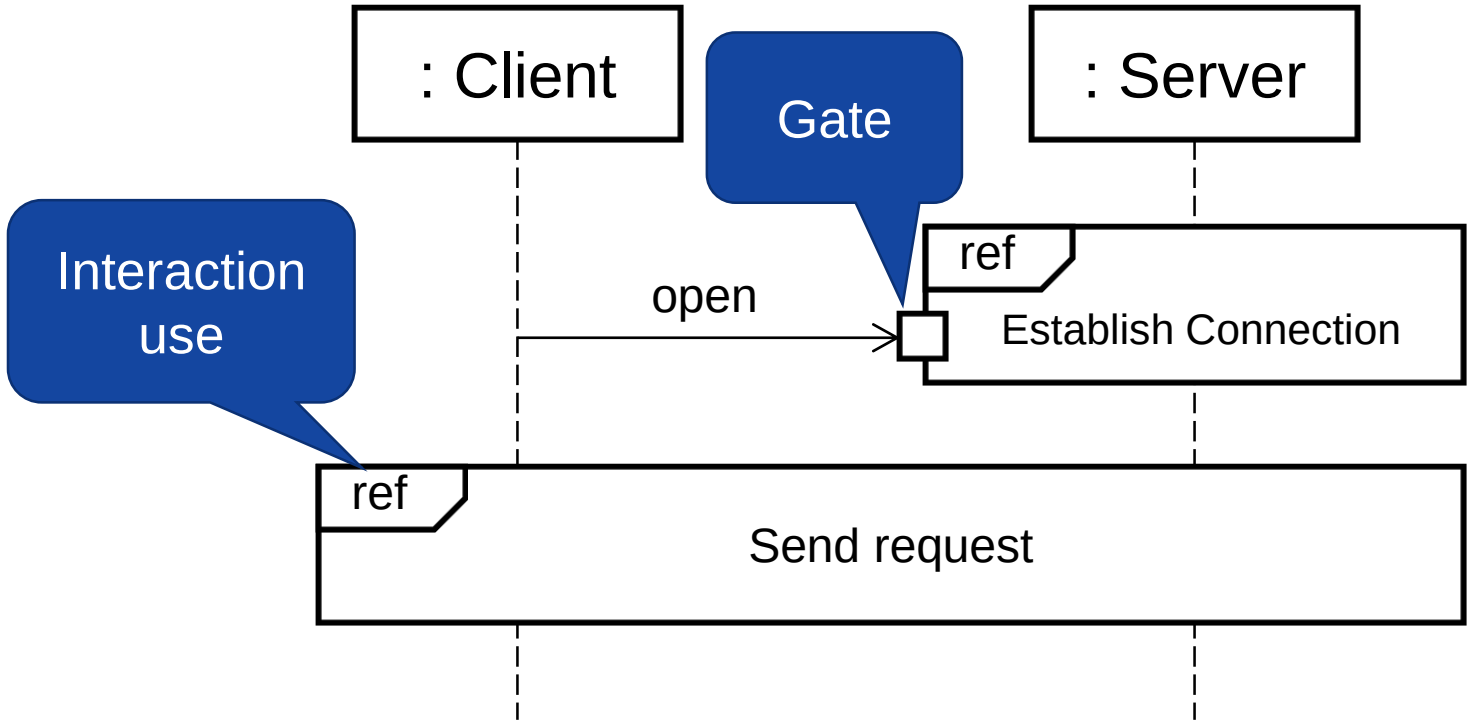


Combined Fragments

- Operators for choice and iteration
  - **alt**: choses from its operands
  - **opt**: either its operand or an empty trace
  - **loop**: loop with lower and upper bounds
  - **break**: executes its operand instead of the remaining of the interaction
- Operators for reordering, parallelisation
  - **par**, **strict**, **seq**, **critical**
- Operators for changing the conformance relation
  - **neg**, **assert**, **ignore**, **consider**
- (Detailed semantics will not be discussed here)

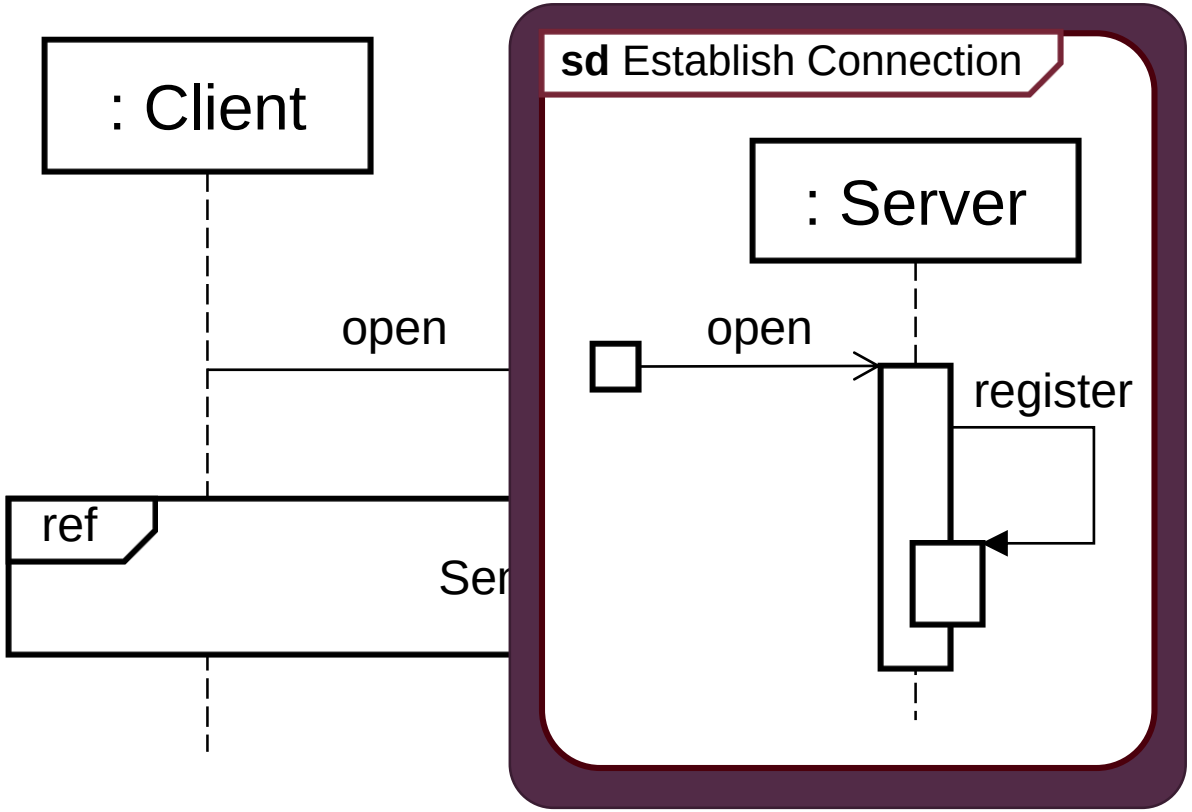
Interaction Use

- Supporting decomposition and reusability

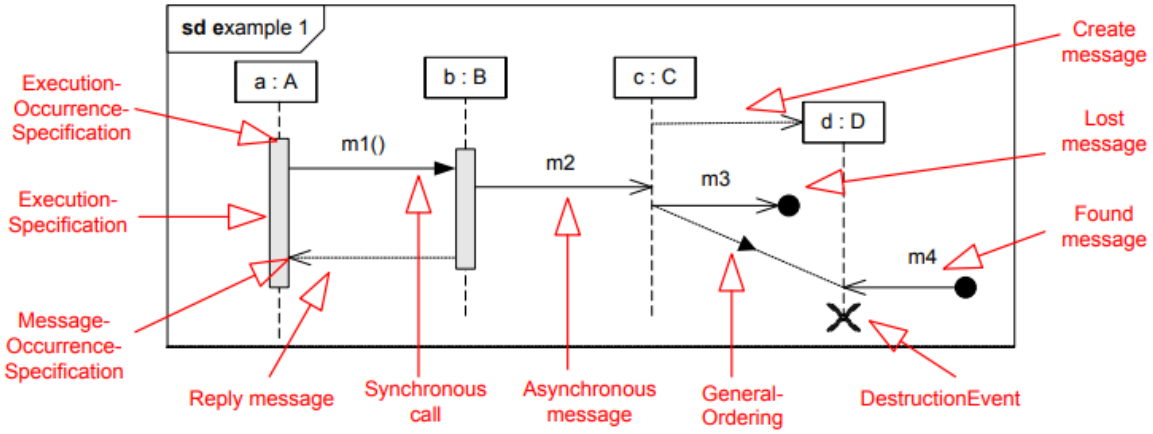


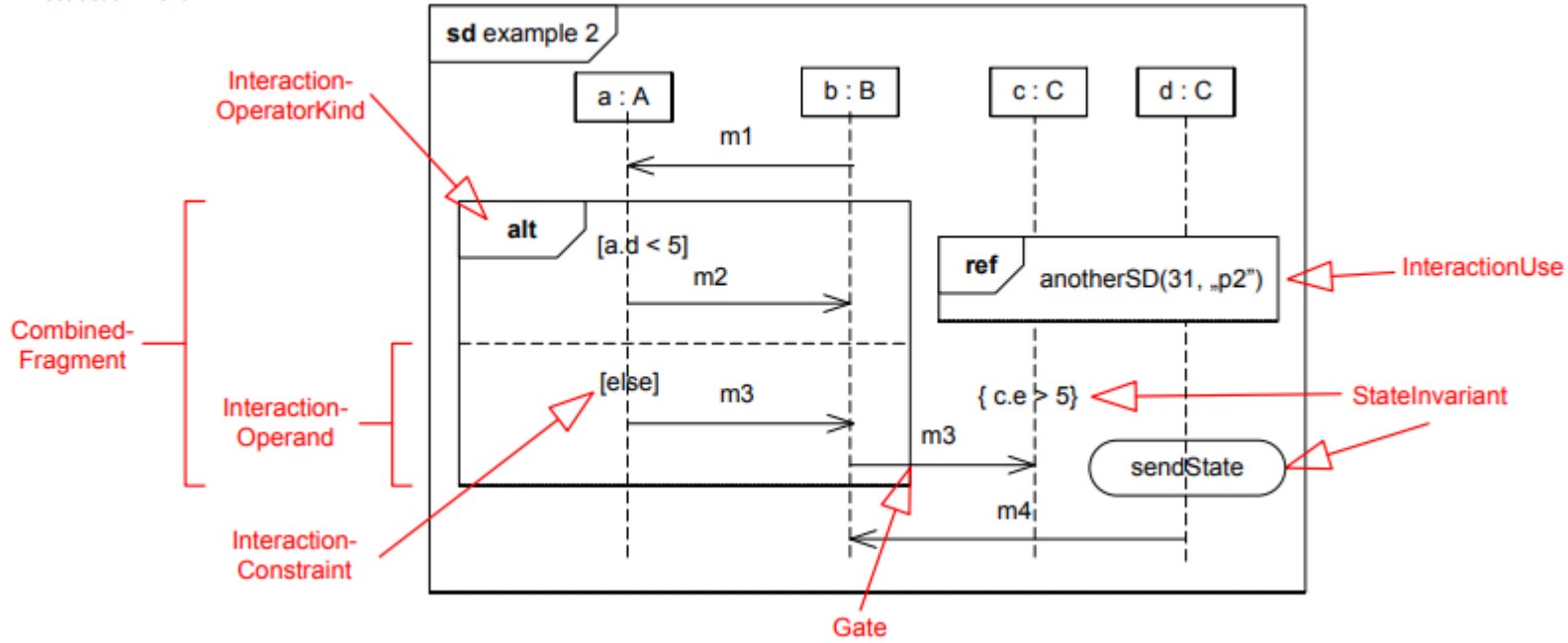
Interaction Use

- Supporting decomposition and reusability

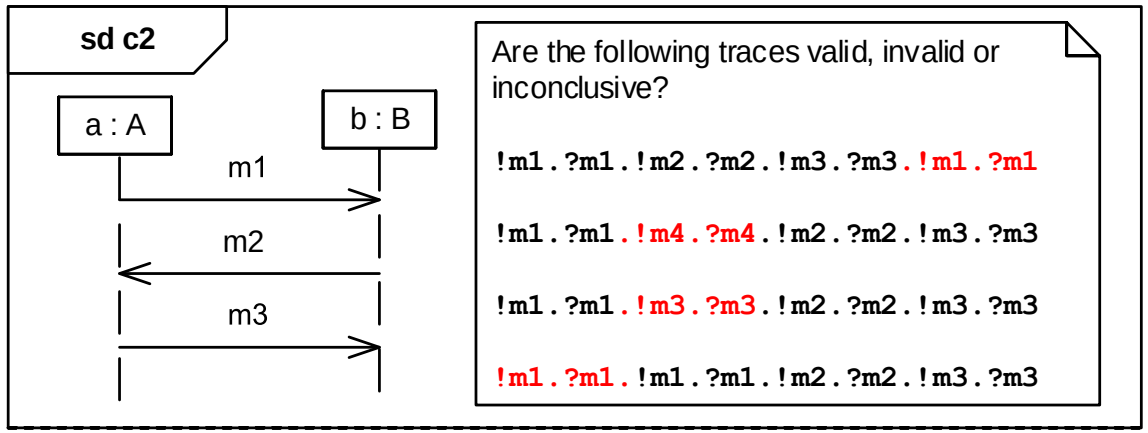


Summary of the Set of Elements





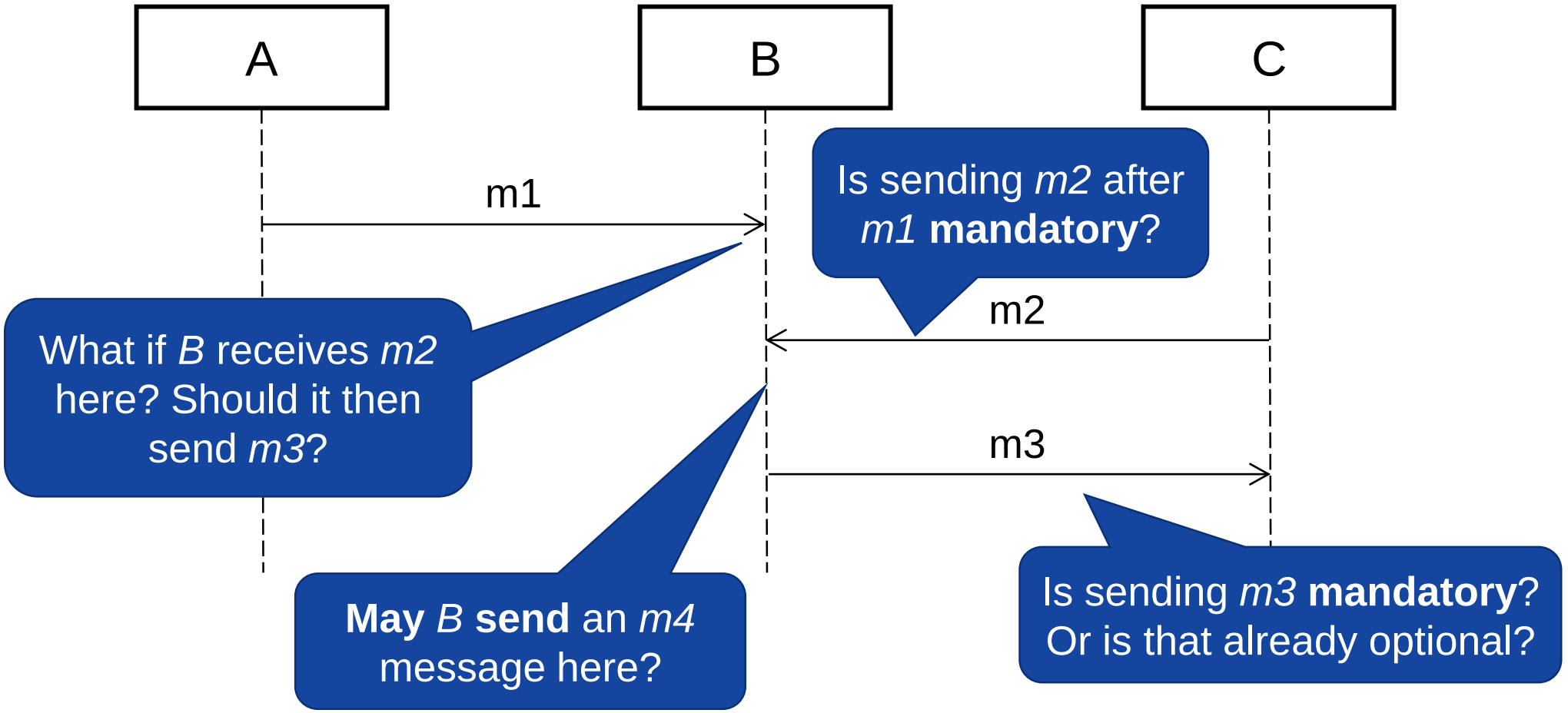
(Simplified) Semantics



Valid, invalid, inconclusive trace

Introduction

Must this whole sequence **always** happen this way?
Or **may** it only **sometimes** happen this way?
Is this the **whole** behaviour of the system?

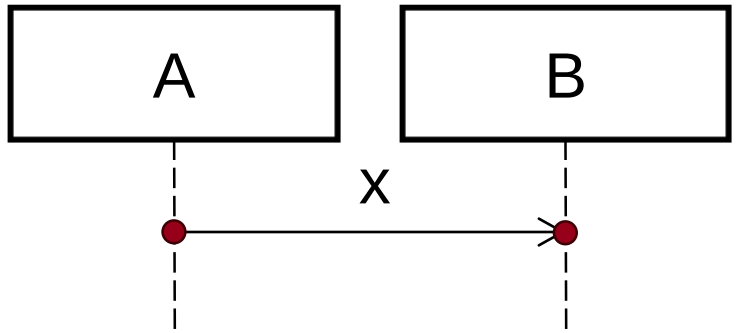


What exactly does this diagram say?

The Basics of the Semantics

- The semantics define **sets of traces** that are either
  - valid, invalid, or inconclusive with regard to the given interaction
- Elements of a trace: **event occurrences**
  - Sending and receiving messages
  - Evaluating StateInvariants
- An interaction defines a **partial order** over the event occurrences
  - Not each pair has a defined order → several traces are possible
  - Even several valid/invalid/inconclusive traces, too

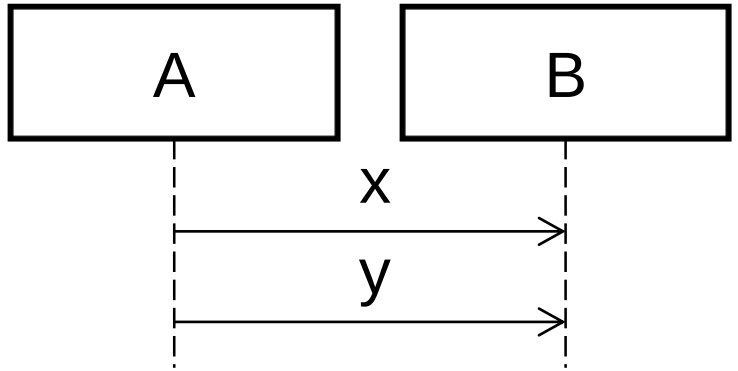
Basic Rules



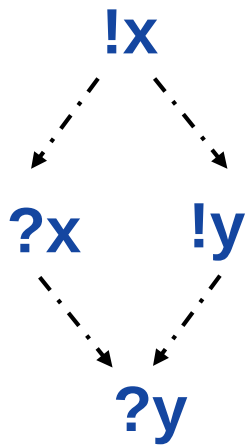
- 2 event occurrences:
  - Sending *x* in A **!x**
  - Receiving *x* in B **?x**
- **Weak (partial) ordering:** „happens-before”
  1. The occurrences **on the same lifeline** are all **ordered**
  2. Receiving is always **after** sending the same message (*causality*)
- Valid trace: { ⟨**!x, ?x**⟩ }

Any other traces are **inconclusive**

Weak Sequencing (Default)



Partial orderings:

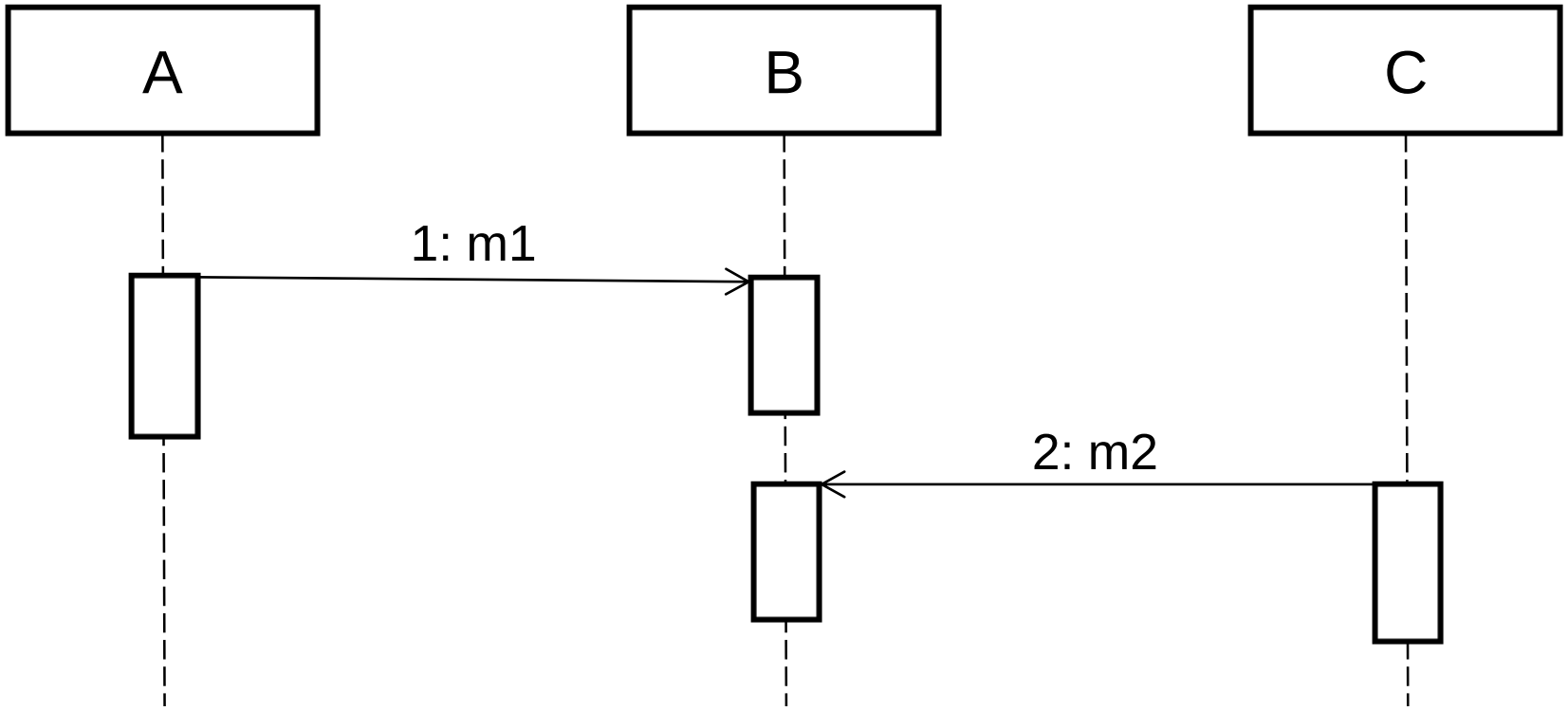


- Weak sequencing: ⟨!x, ?x⟩ **seq** ⟨!y, ?y⟩
  - Inside a specific operand, ordering is kept
  - The occurrences are ordered **only on the same lifeline**
    - Ordering follows the ordering of the operands of the sequencing
    - The ordering of **?x** and **!y** is **not defined**
- Set of valid traces:

  { ⟨!x, ?x, !y, ?y⟩ , ⟨!x, !y, ?x, ?y⟩ }

Caution: Numbering of the Messages

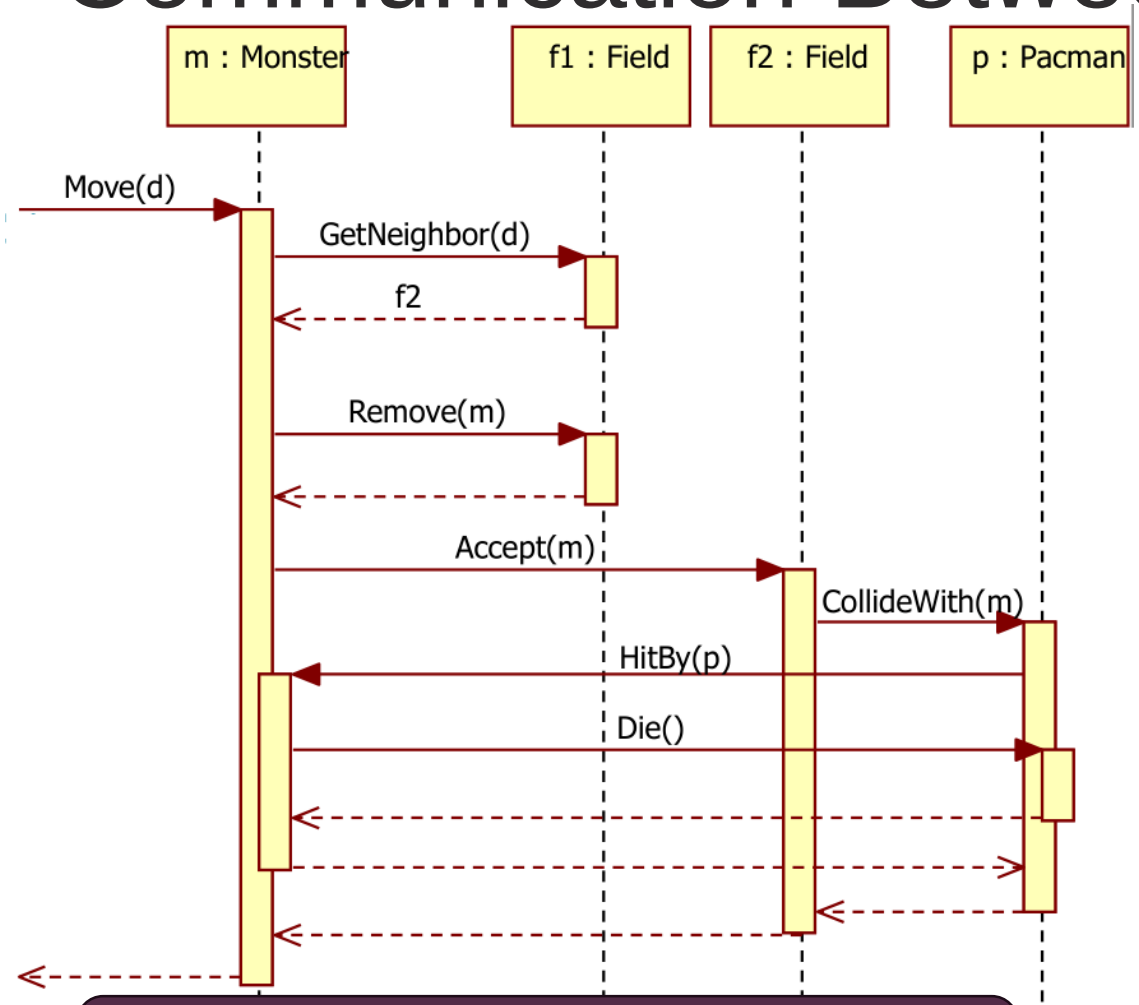
Tools often use numbering for sequencing:



Why be careful with this?

Modelling distributed systems is a theoretically difficult task!

Call Chain in a Single-Threaded Program vs. Communication Between Independent Actors



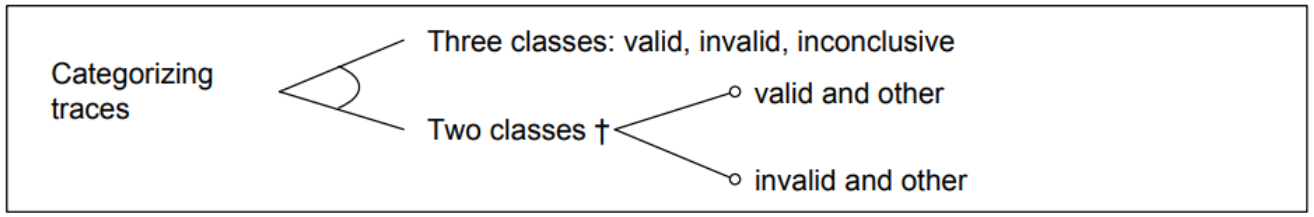
vs.

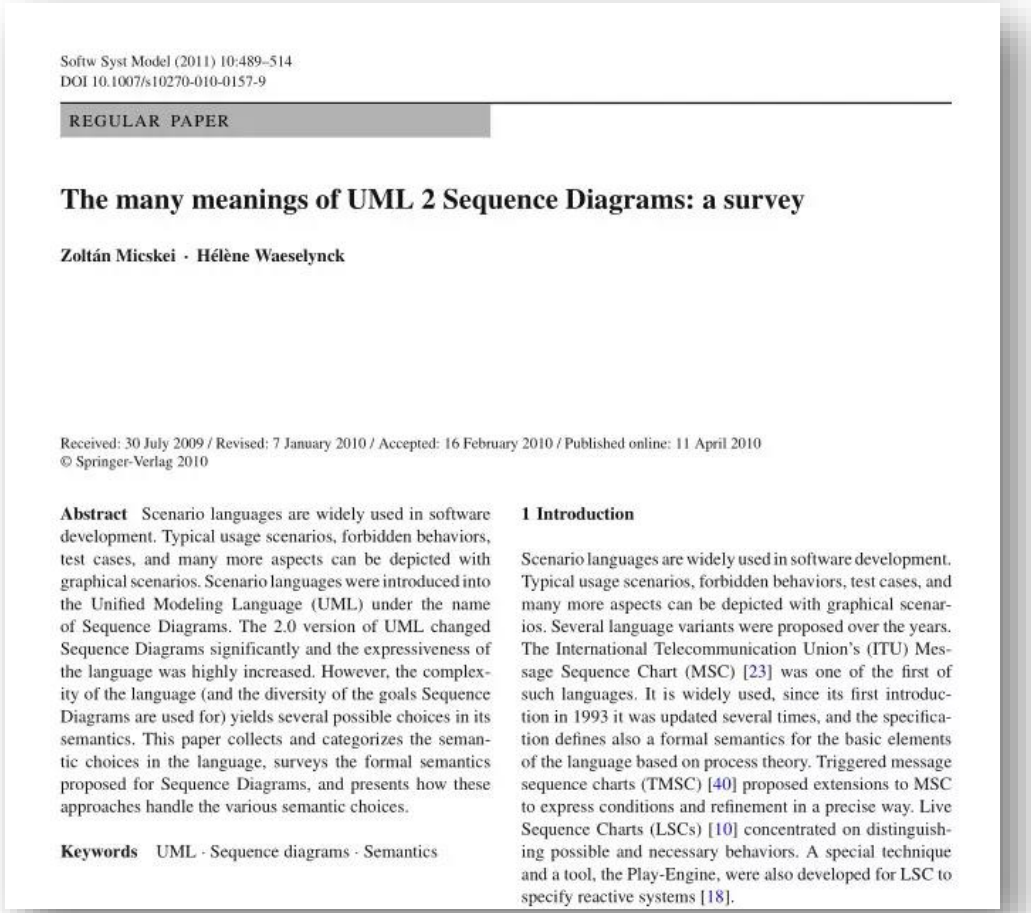
Modelling a sequential execution

Distributed system with several independent actors

Interpretation and Application

- The meaning of the further elements is even more complex
- May depend on the goal of application
  - Initial, uncertain plans?
  - Description of test cases?
- Invalid and inconclusive traces
  - Many different interpretations
- Specify the intention of the model





Softw Syst Model (2011) 10:489–514
DOI 10.1007/s10270-010-0157-9

REGULAR PAPER

The many meanings of UML 2 Sequence Diagrams: a survey

Zoltán Micskei · Hélène Waeselynck

Received: 30 July 2009 / Revised: 7 January 2010 / Accepted: 16 February 2010 / Published online: 11 April 2010
© Springer-Verlag 2010

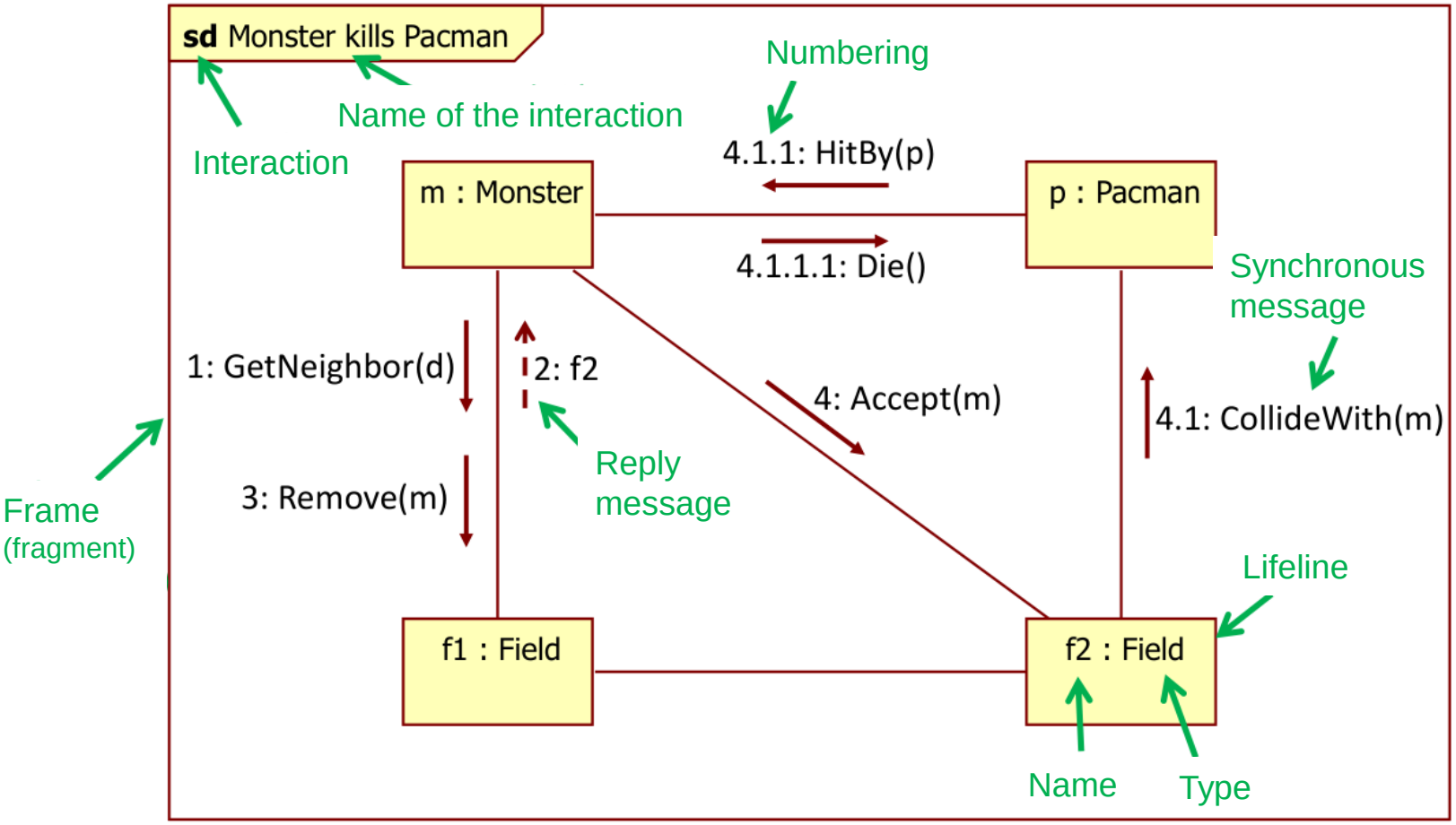
Abstract Scenario languages are widely used in software development. Typical usage scenarios, forbidden behaviors, test cases, and many more aspects can be depicted with graphical scenarios. Scenario languages were introduced into the Unified Modeling Language (UML) under the name of Sequence Diagrams. The 2.0 version of UML changed Sequence Diagrams significantly and the expressiveness of the language was highly increased. However, the complexity of the language (and the diversity of the goals Sequence Diagrams are used for) yields several possible choices in its semantics. This paper collects and categorizes the semantic choices in the language, surveys the formal semantics proposed for Sequence Diagrams, and presents how these approaches handle the various semantic choices.

Keywords UML · Sequence diagrams · Semantics

1 Introduction

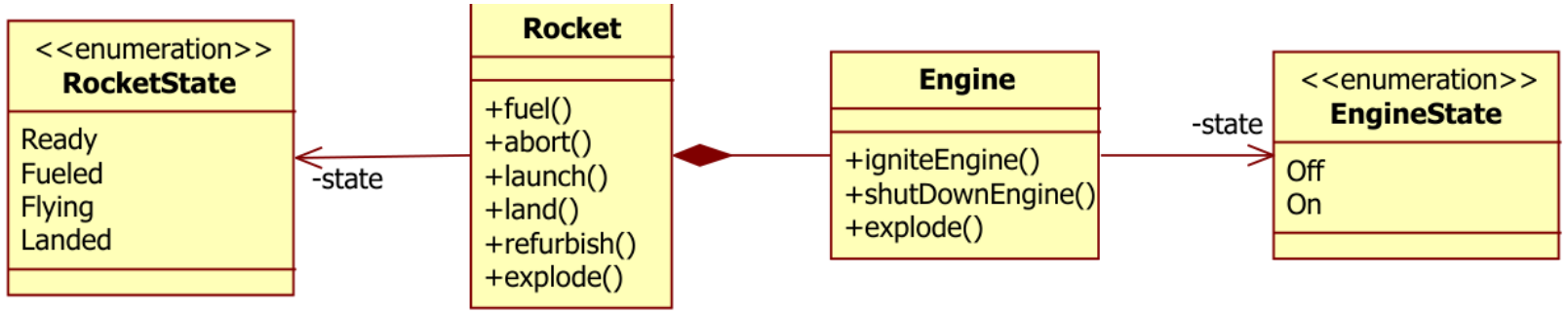
Scenario languages are widely used in software development. Typical usage scenarios, forbidden behaviors, test cases, and many more aspects can be depicted with graphical scenarios. Several language variants were proposed over the years. The International Telecommunication Union's (ITU) Message Sequence Chart (MSC) [23] was one of the first of such languages. It is widely used, since its first introduction in 1993 it was updated several times, and the specification defines also a formal semantics for the basic elements of the language based on process theory. Triggered message sequence charts (TMSC) [40] proposed extensions to MSC to express conditions and refinement in a precise way. Live Sequence Charts (LSCs) [10] concentrated on distinguishing possible and necessary behaviors. A special technique and a tool, the Play-Engine, were also developed for LSC to specify reactive systems [18].

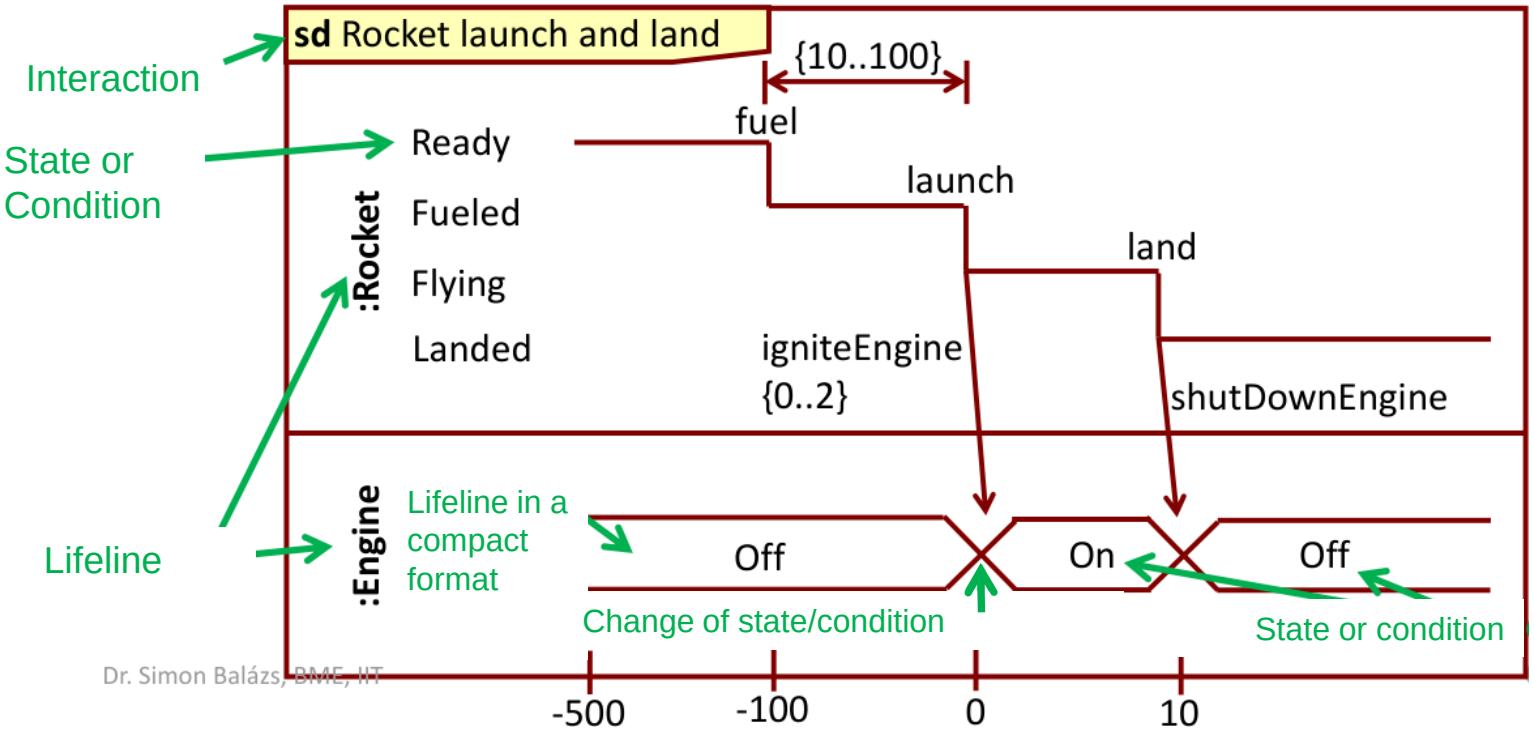
Further Diagrams (1)



- Communication diagram
- Description of a single trace
- Advisable in the case of „single-threaded“ systems

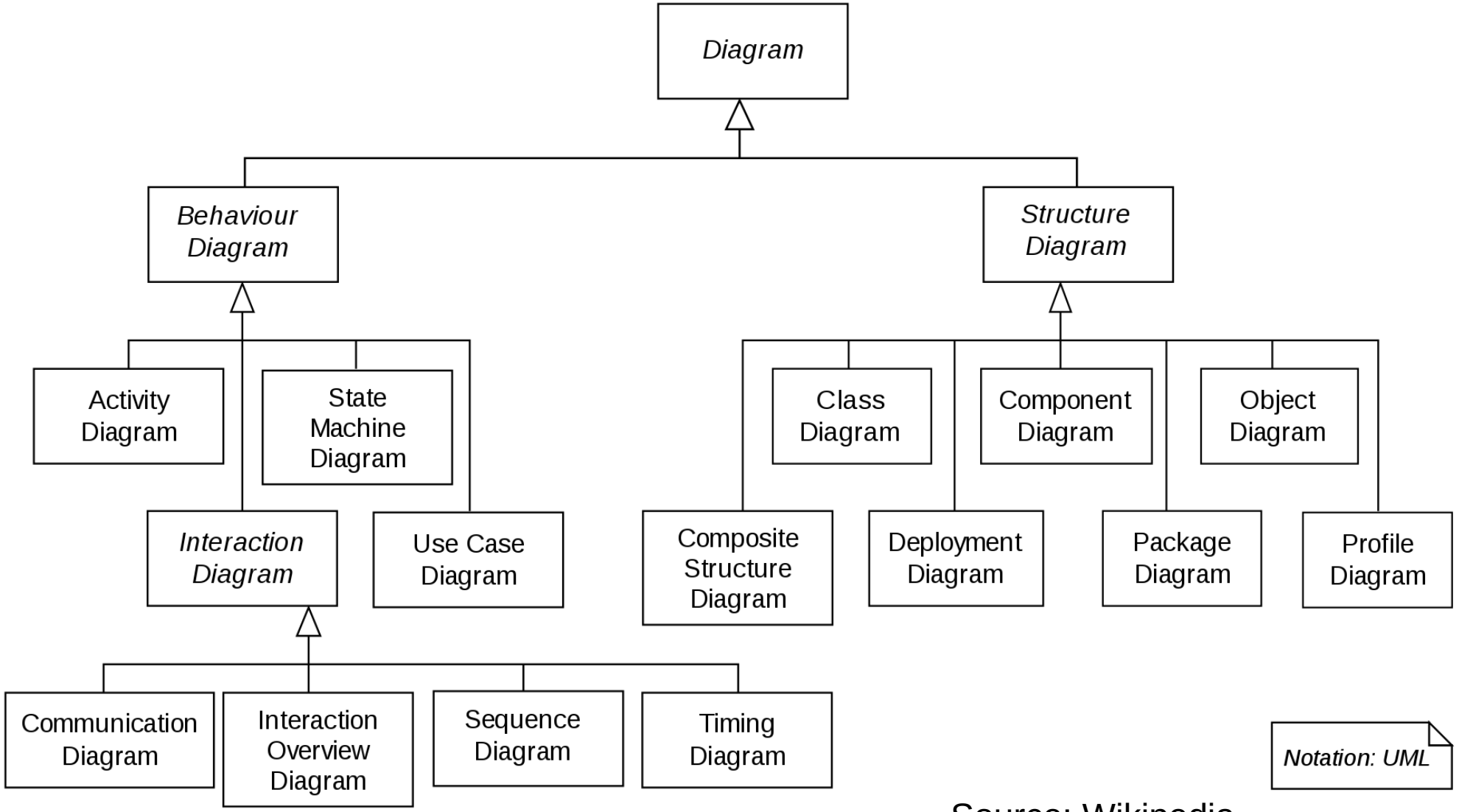
Further Diagrams (2)





- Time diagram
- Time axis
- State changes and messages

Summary of UML Diagram Types

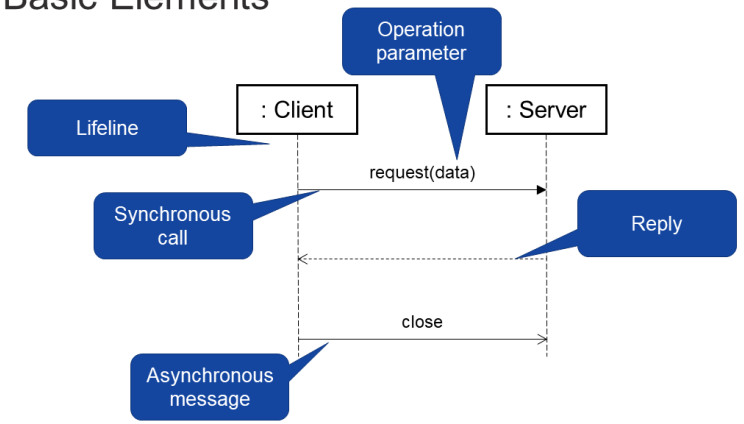


Source: Wikipedia

Summary

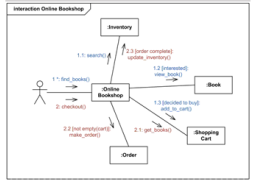
Summary

Basic Elements

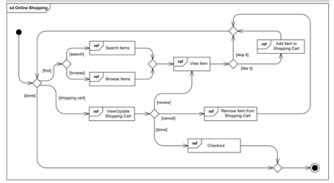


Types of Interaction Diagrams

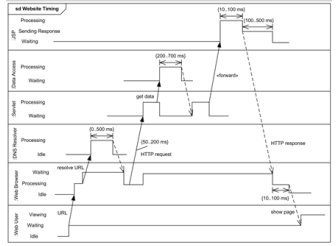
Communication



Interaction Overview

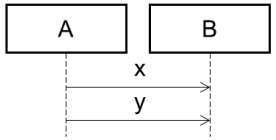


Timing

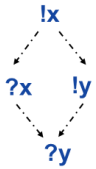


Source: http://www.uml-diagrams.org

Weak Sequencing (Default)

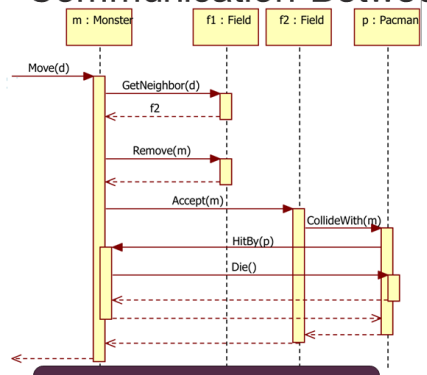


Partial orderings:



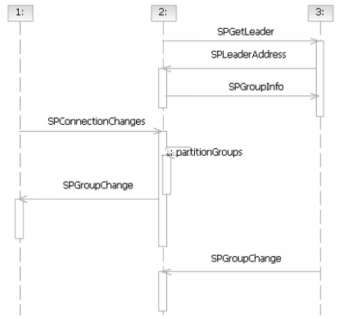
- Weak sequencing: ⟨!x, ?x⟩ **seq** ⟨!y, ?y⟩
  - Inside a specific operand, ordering is kept
  - The occurrences are ordered **only on the same lifeline**
    - Ordering follows the ordering of the operands of the sequencing
    - The ordering of ?x and !y is **not defined**
- Set of valid traces:
  { ⟨!x, ?x, !y, ?y⟩ , ⟨!x, !y, ?x, ?y⟩ }

Call Chain in a Single-Threaded Program vs. Communication Between Independent Actors



Modelling a sequential execution

vs.



Distributed system with several independent actors