Modelling II

Sequence Diagrams

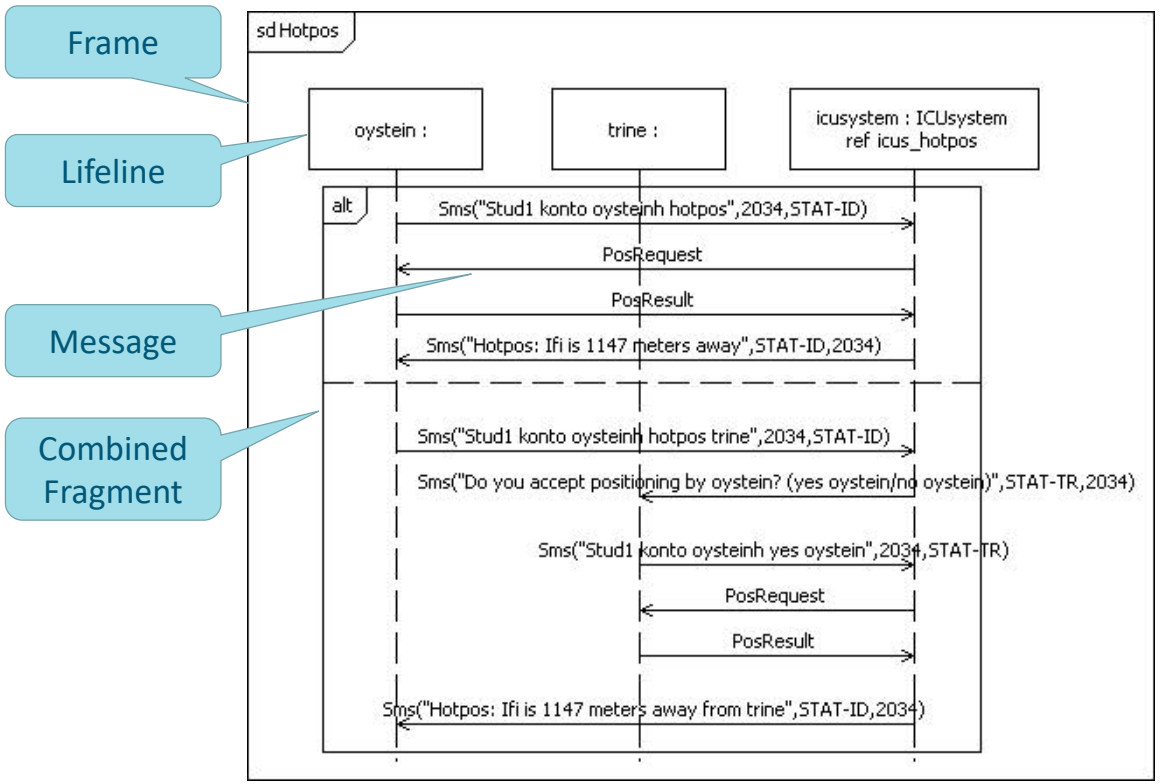
Ketil Stølen

Partly based on slides prepared by Prof. Øystein Haugen, HiØ & SINTEF

Overview of lecture

- Sequence Diagrams
  - What are they intended for?
  - Where in the software engineering process are they used?
- Basic sequence diagrams
- Interaction Fragments – structuring mechanisms

This is a Sequence Diagram



Exercise: What makes sequence diagrams *fundamentally different* from program code?

Sequence Diagrams in a nutshell

- Sequence Diagrams are
  - simple
  - powerful
  - readable
- Emphasizes the interaction between objects when interplay is the most important aspect
  - Often only a small portion of the total variety of behavior is described improve the individual understanding of an interaction problem

Sequence Diagrams are used to ...

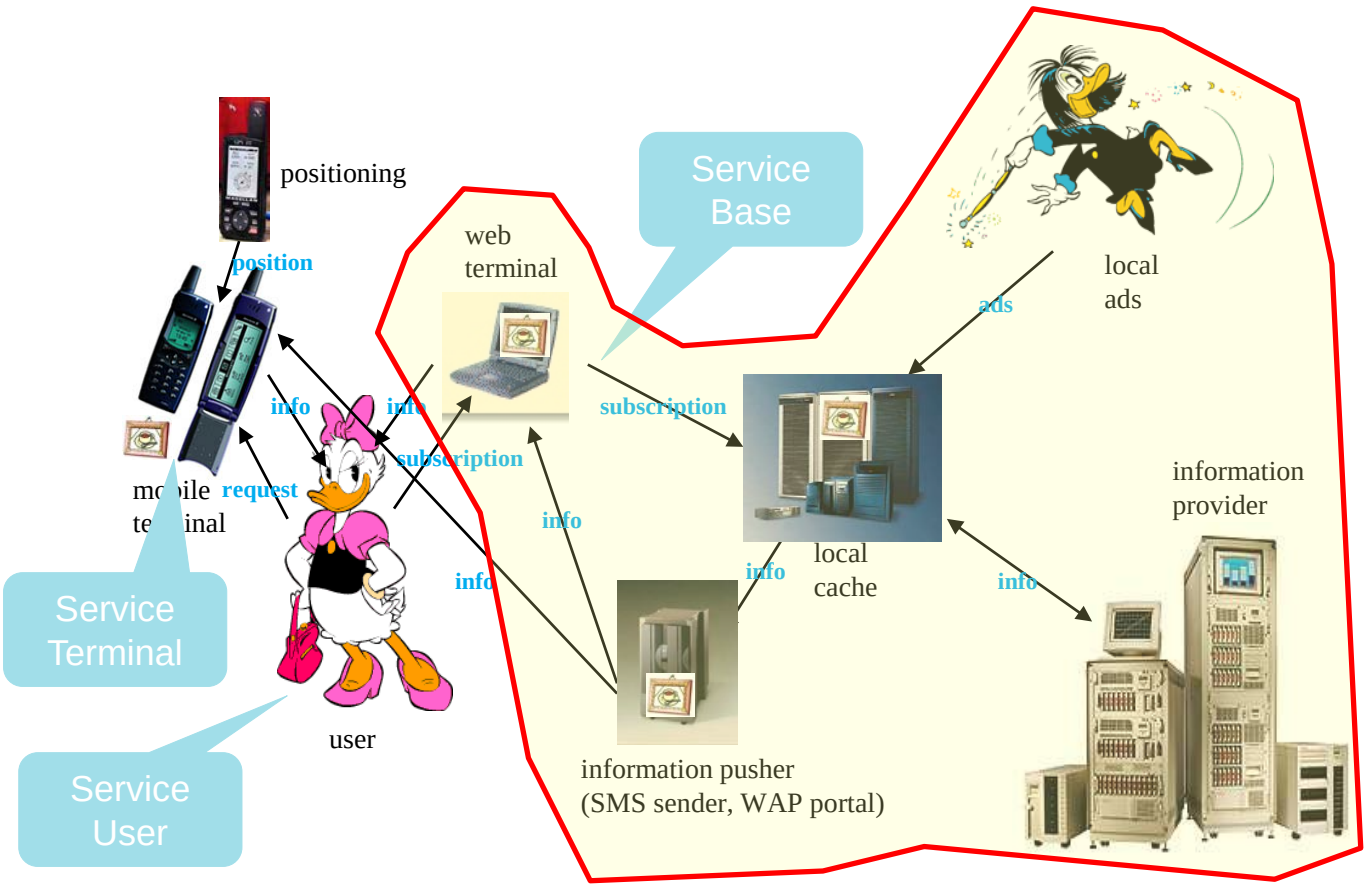
- document protocol situations,
- exemplify behavior situations,
- verify interaction properties relative to a specification,
- describe test cases,
- document simulation traces.

The example context: Dolly Goes To Town

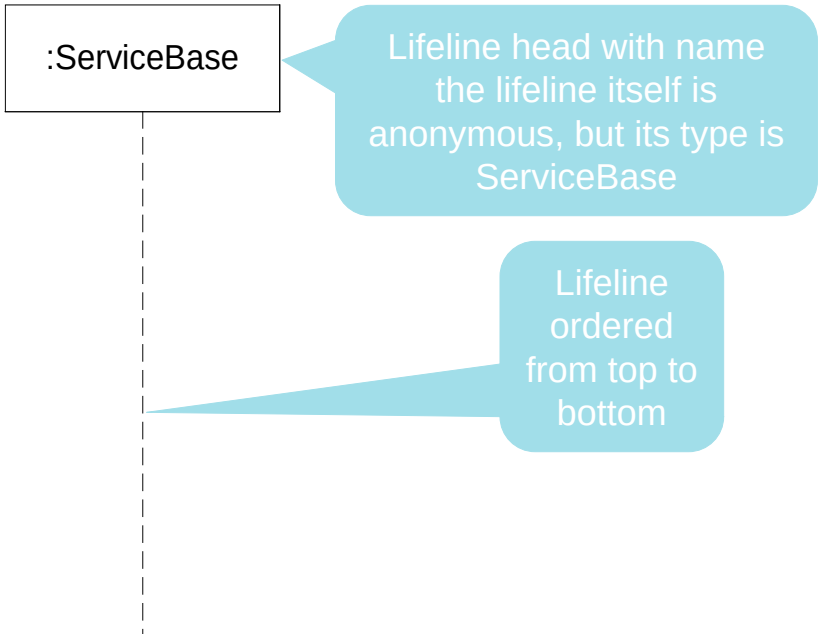
- Dolly is going to town and
  - wants to subscribe for bus schedules back home
  - given her current position
  - and the time of day;

- The service should not come in effect until a given time in the evening

The informal architecture

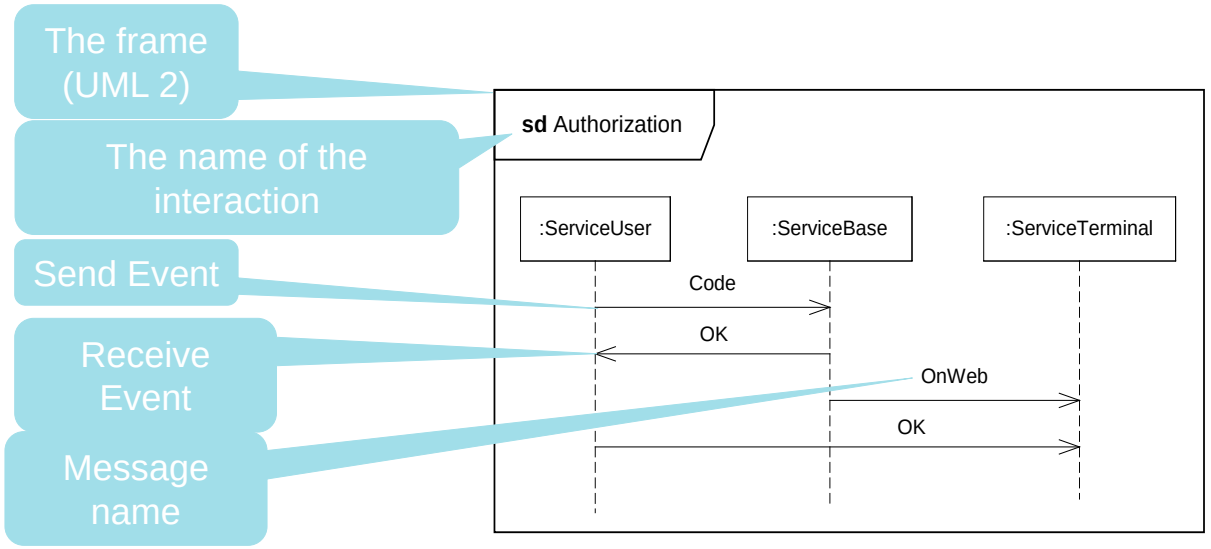


Lifeline – the “doers”



(Simple) Sequence Diagram

- Messages have one send event, and one receive event.
  - The send event must occur before the receive event.
- Events are strictly ordered along a lifeline from top to bottom

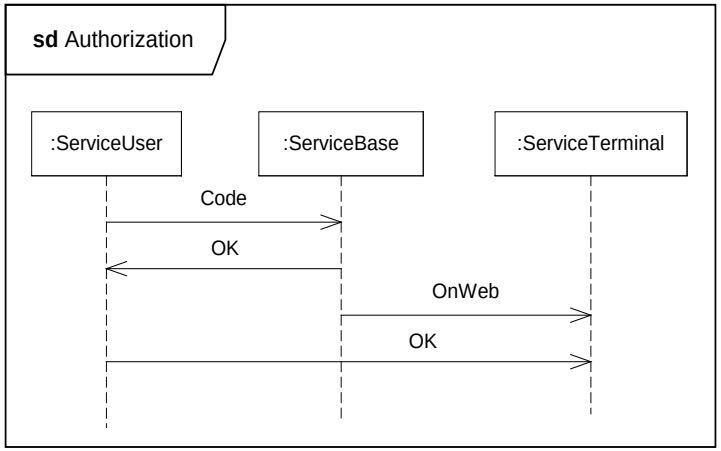


Exercise: How many *global traces* are there in this diagram?

- The only invariants:
  - Messages have one send event, and one receive event. The send event must occur before the receive event.
  - Events are strictly ordered along lifeline

How many?

- 1, 2, 3, 4, 5, 6,..?



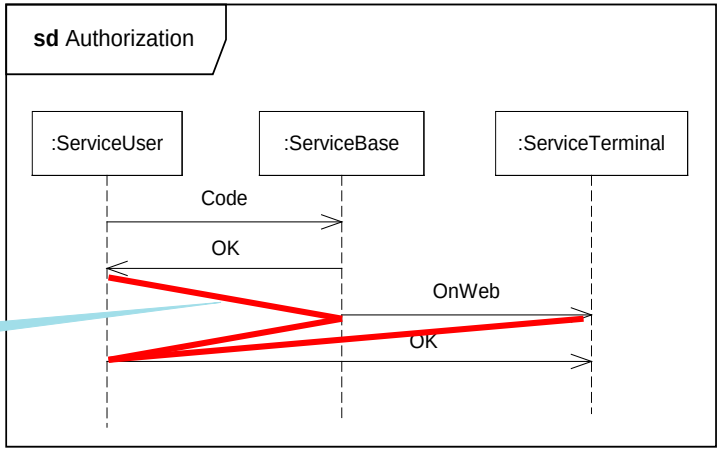
How many global traces are there in this diagram?

- The only invariants:
  - Messages have one send event, and one receive event. The send event must occur before the receive event.
  - Events are strictly ordered along lifeline

How many?

- 1, 2, 3, 4, 5, 6,..?

independent!

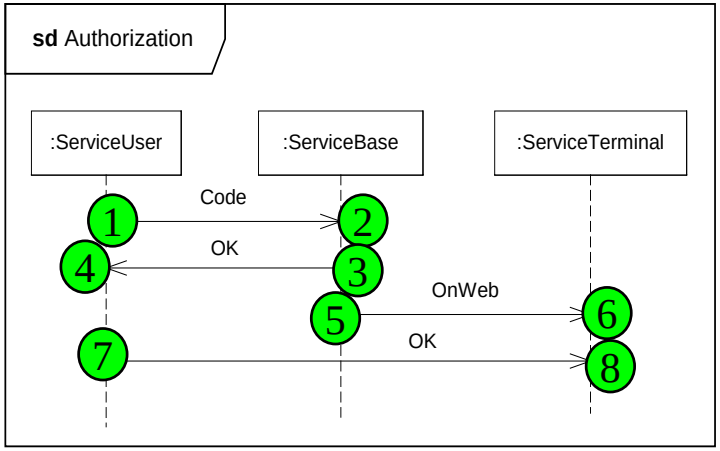


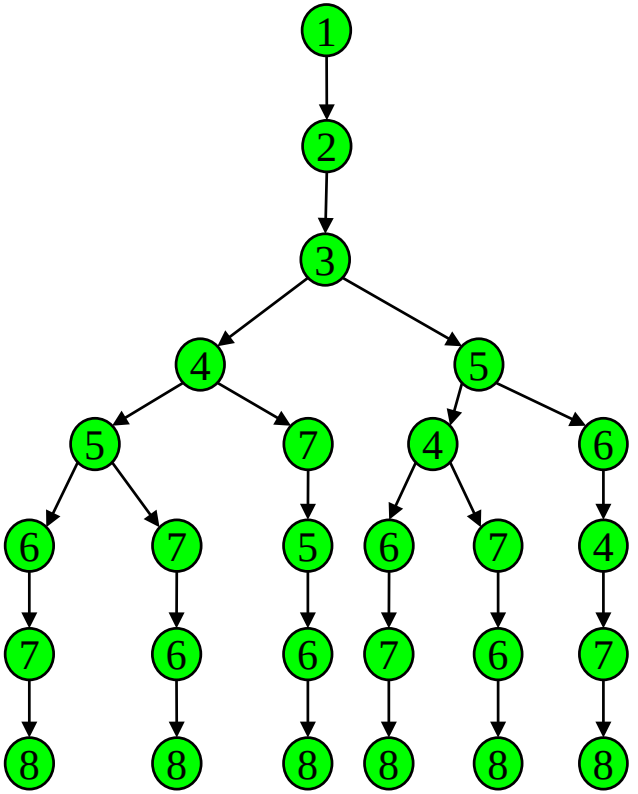
Causality and weak sequencing

- Causality:
  - a message can never be received before it has been transmitted
  - the transmission event for a message is therefore always ordered before the reception event for the same message

- Weak sequencing:
  - events from the same lifeline are ordered in the trace in the same order as on the lifeline

Really counting the traces ...

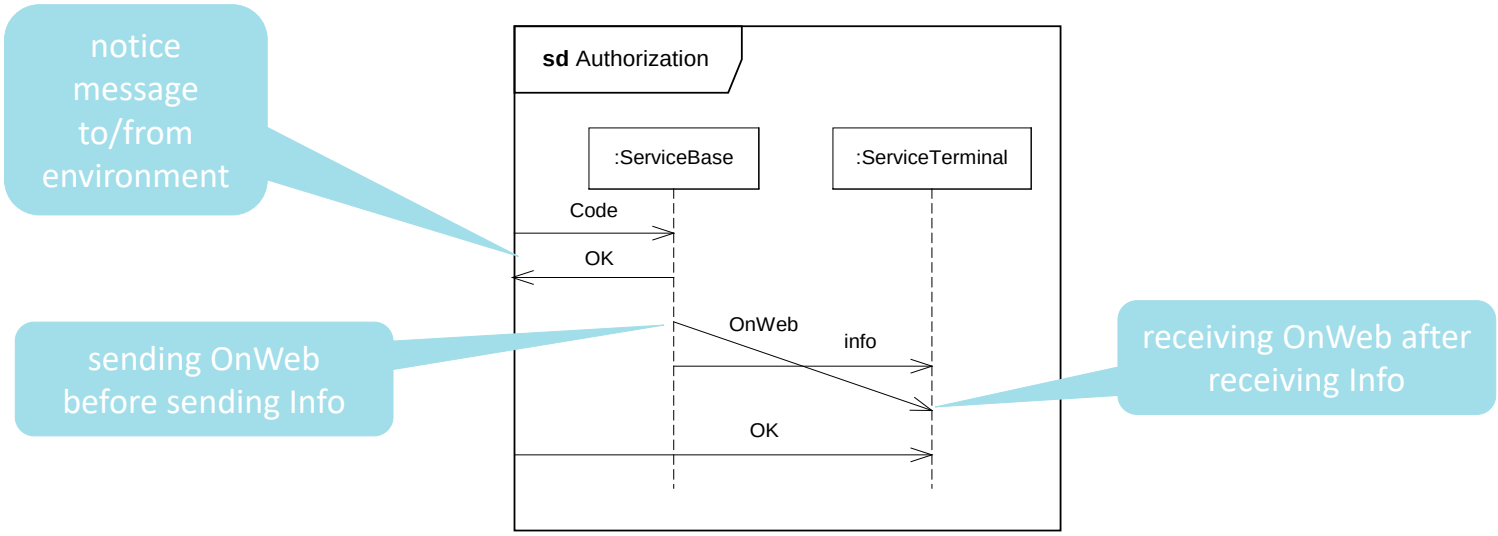




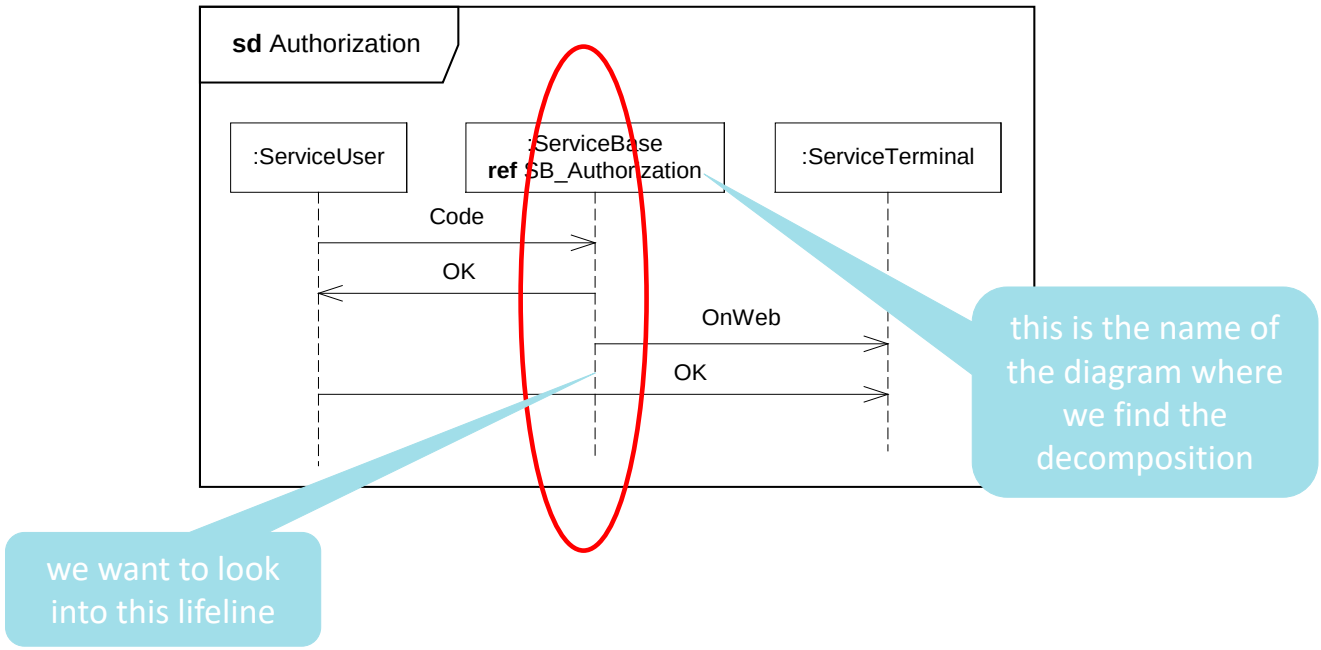
Exercise: Explain how the number of traces in the previous diagram can be *reduced* by adding messages

Asynchronous messages: Message Overtaking

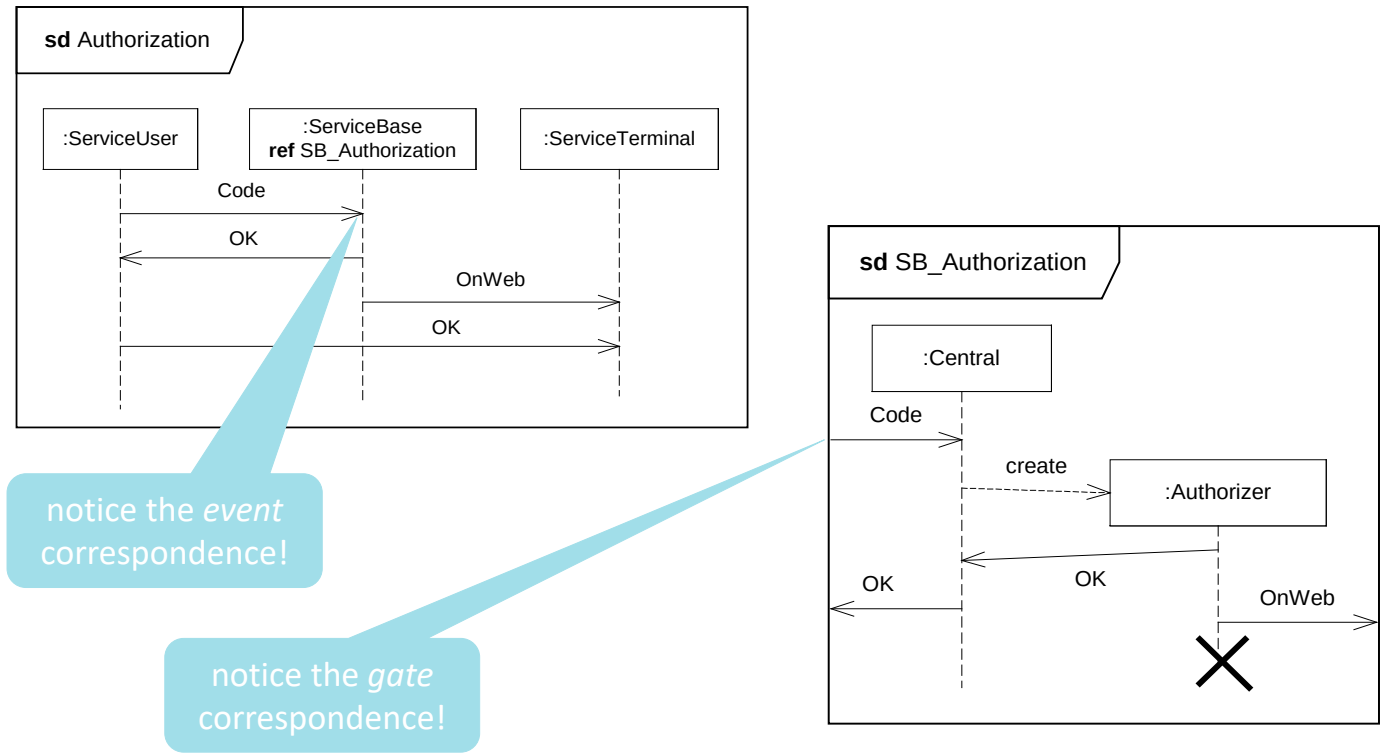
- asynchronous communication = when the sender does not wait for the reply of the message sent
- Reception is normally interpreted as consumption of the message.
- When messages are asynchronous, it is important to be able to describe message overtaking.



Decomposing a Lifeline relative to an Interaction

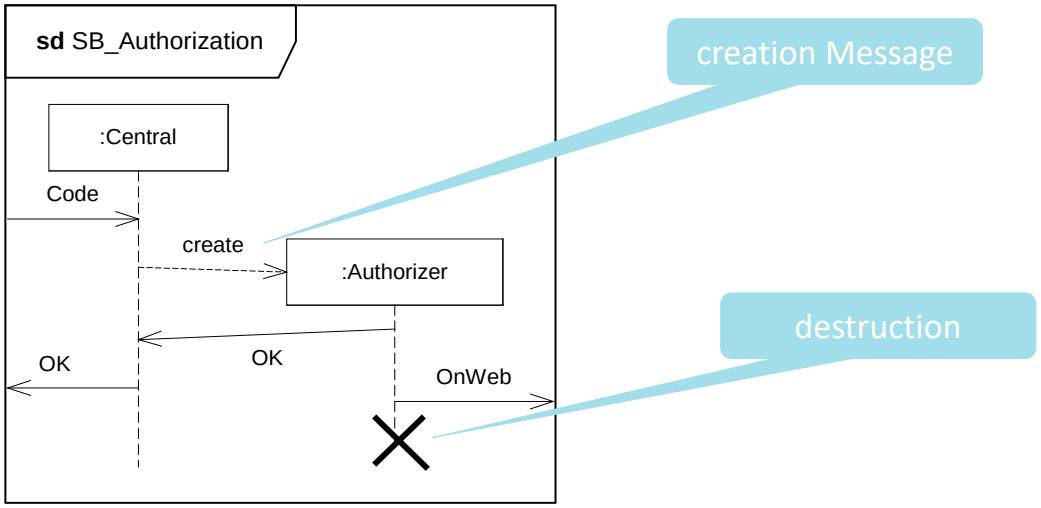


The Decomposition



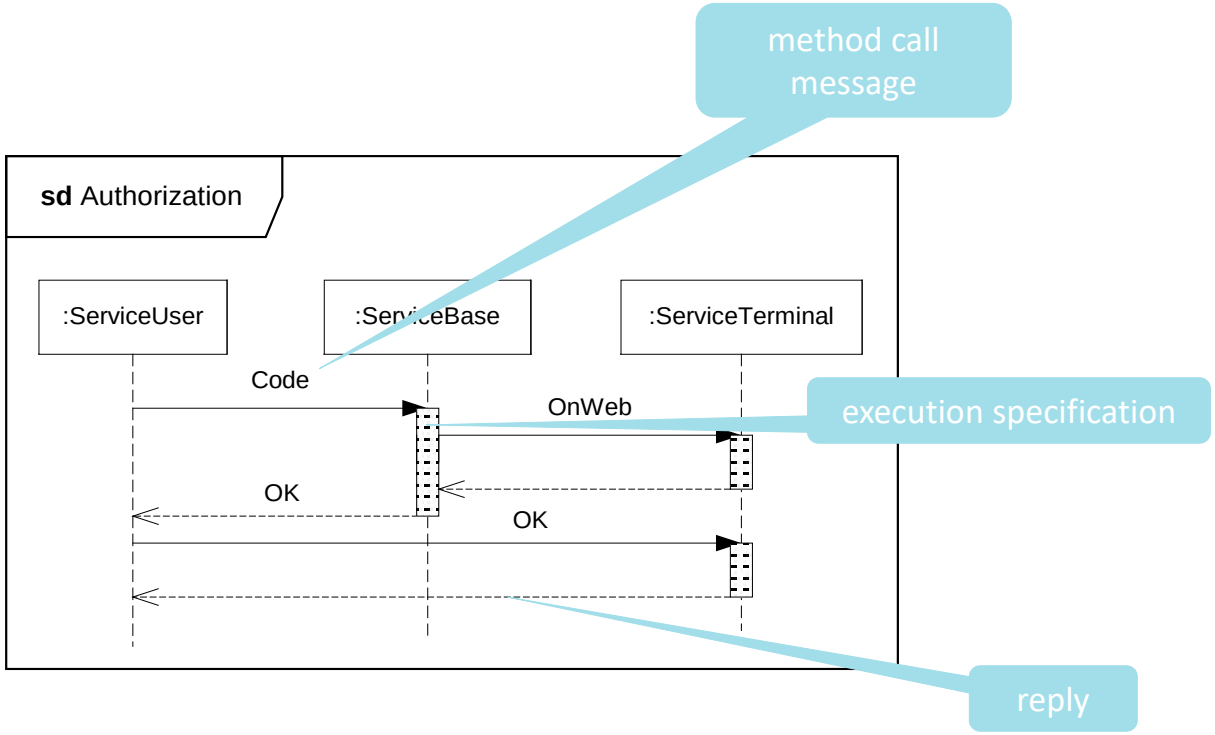
Lifeline creation and destruction

- We would like to describe Lifeline creation and destruction
- The idea here (though rather far fetched) is that the ServiceBase needs to create a new process in the big mainframe computer to perform the task of authorizing the received Code. We see a situation where several Authorizers work in parallel



Exercise: How many global traces are there in the *decomposed* diagram?

Synchronizing interaction



Basic Sequence Diagrams Summary

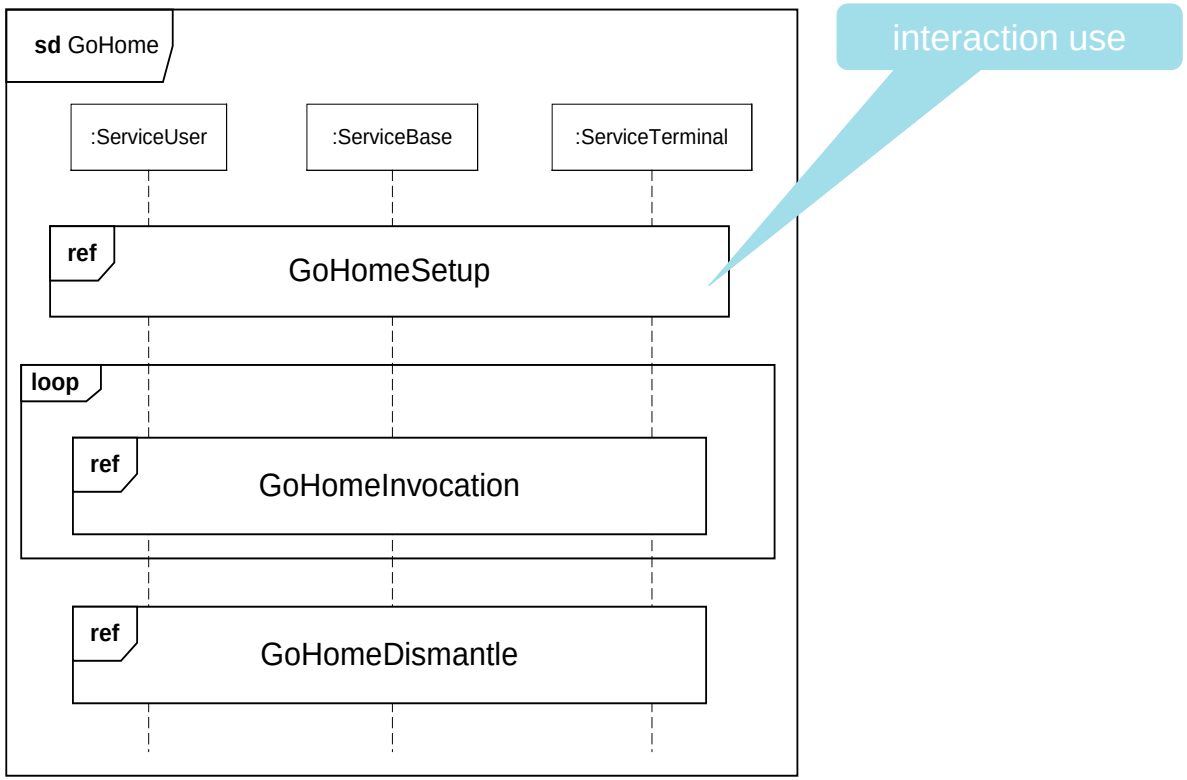
- We consider mostly messages that are asynchronous, the sending of one message must come before the corresponding reception
- UML has traditionally described synchronizing method calls rather than asynchronous communication
- The events on a lifeline are strictly ordered
- The distance between events is not significant.
- The context of Interactions are classifiers
- A lifeline (within an interaction) may be detailed in a decomposition
- Dynamic creation and destruction of lifelines

More structure

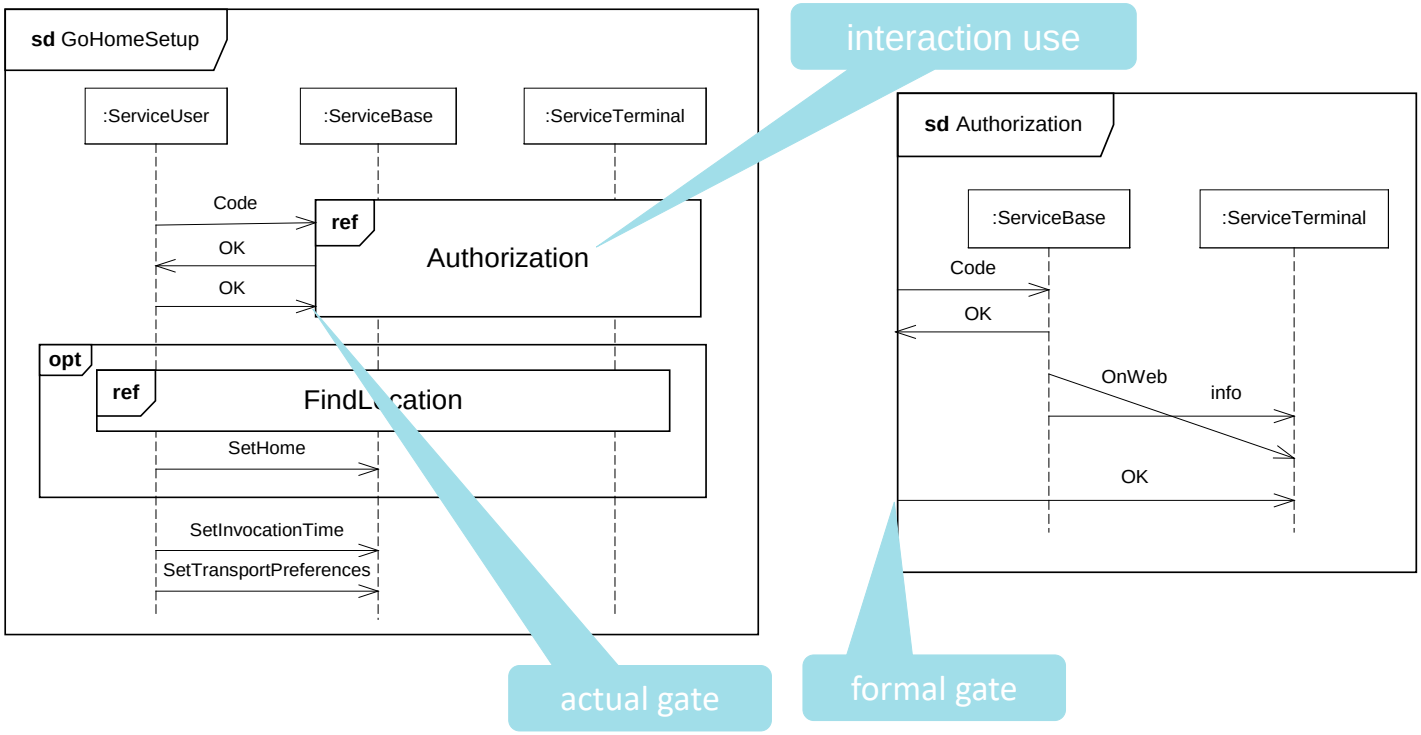
- interaction uses – such that Interactions may be referenced within other Interactions
- combined fragments – combining Interaction fragments to express alternatives, parallel merge and loops

- better overview of combinations – High level Interactions where Lifelines and individual Messages are hidden
  - Not so useful since no tools support this
- gates – flexible connection points between references/expressions and their surroundings
  - we have looked at this in the context of decomposition, but gates are also on InteractionUse and CombinedFragments

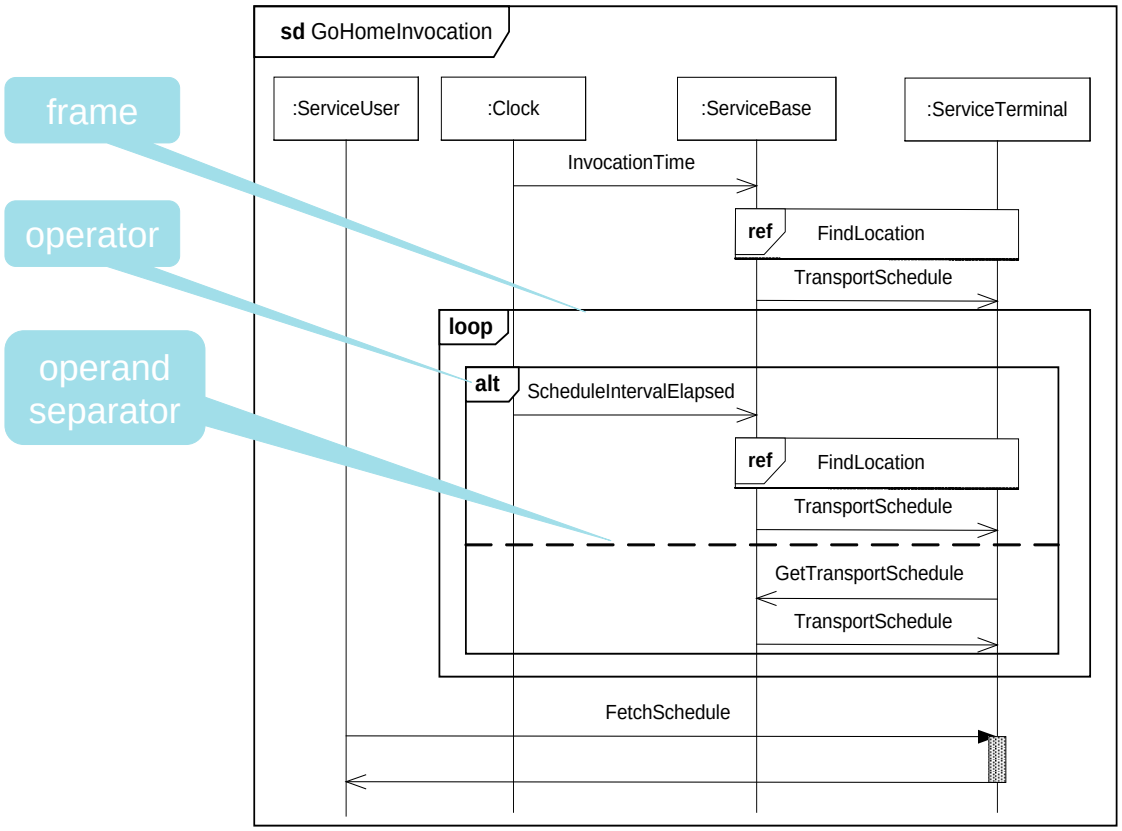
References


sd GoHome
:ServiceUser
:ServiceBase
:ServiceTerminal
ref
GoHomeSetup
loop
ref
GoHomeInvocation
ref
GoHomeDismantle
interaction use

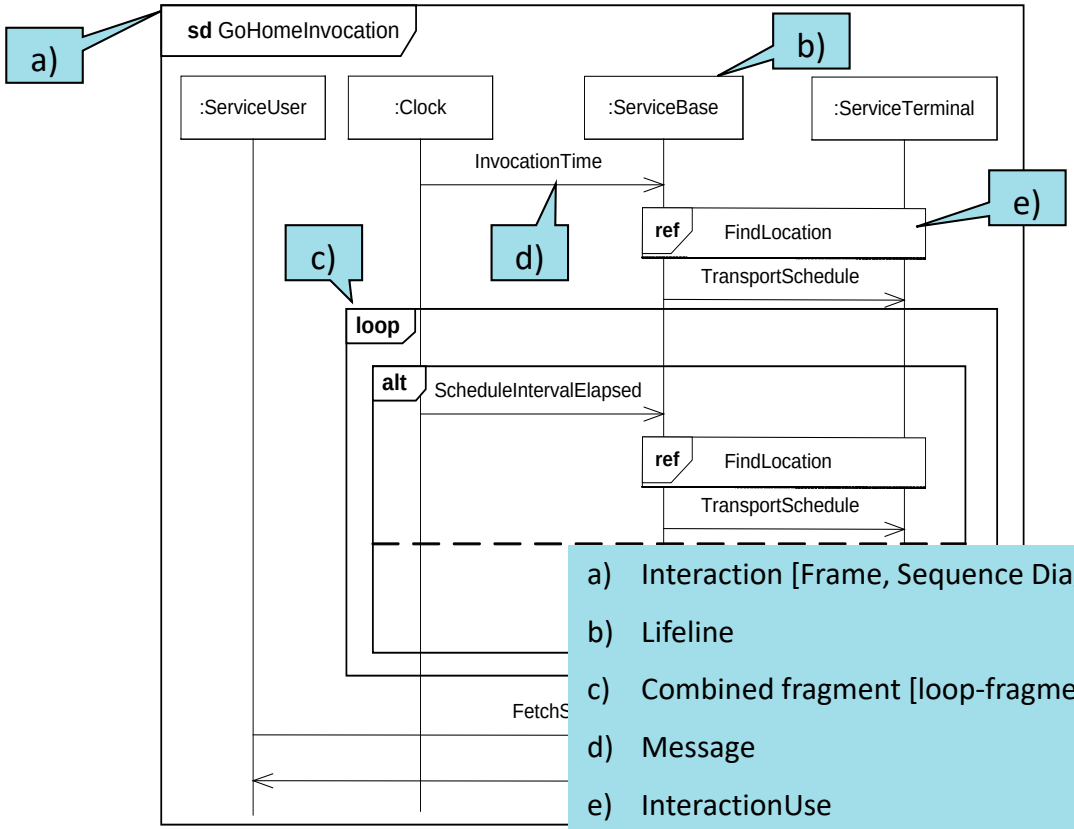
Gates


sd GoHomeSetup
:ServiceUser
:ServiceBase
:ServiceTerminal
Code
OK
OK
ref
Authorization
opt
ref
FindLocation
SetHome
SetInvocationTime
SetTransportPreferences
interaction use
sd Authorization
:ServiceBase
:ServiceTerminal
Code
OK
OnWeb
info
OK
actual gate
formal gate

Combined fragment example



And now chiefly yourselves !!!



a) Interaction [Frame, Sequence Diagram]

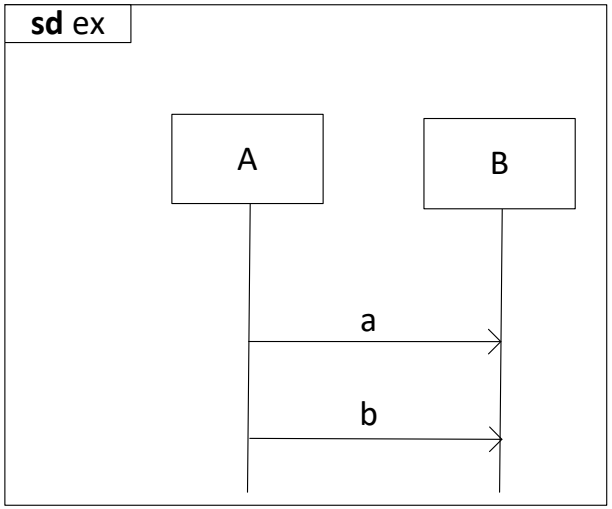
b) Lifeline

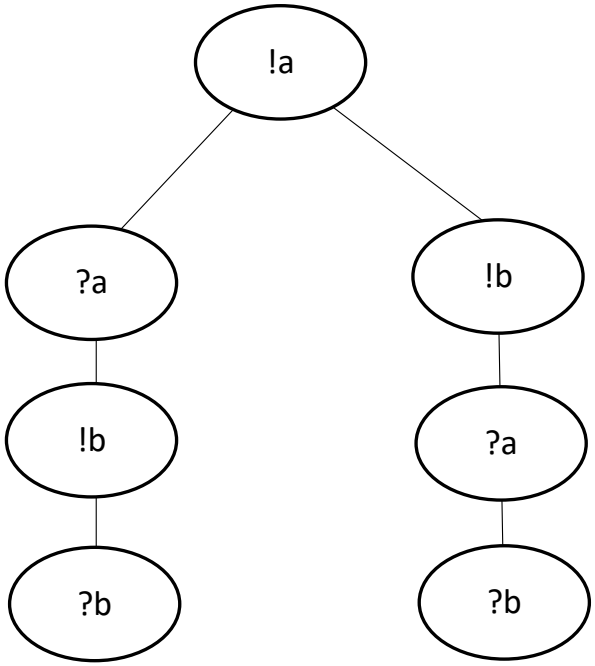
c) Combined fragment [loop-fragment]

d) Message

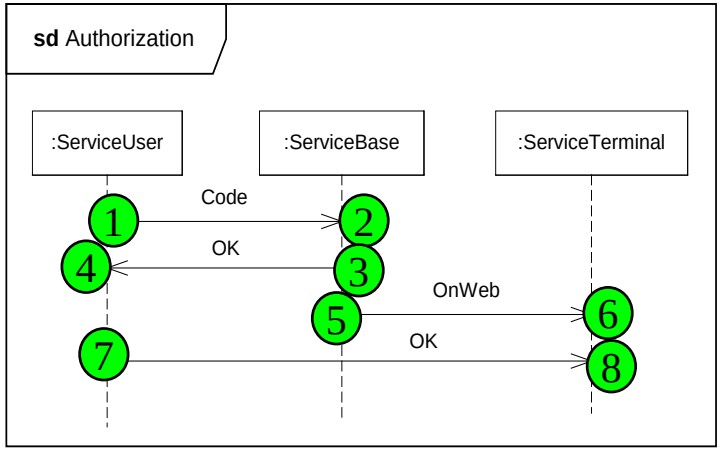
e) InteractionUse

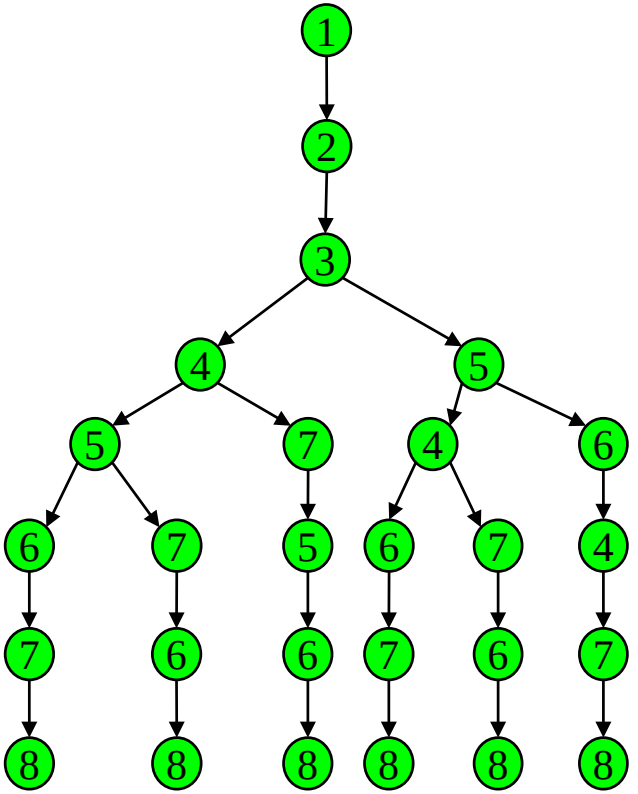
Summary of sequence diagrams – positive behavior I



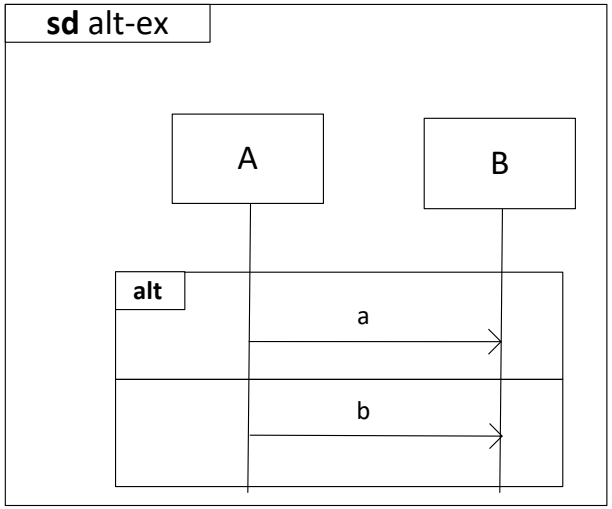


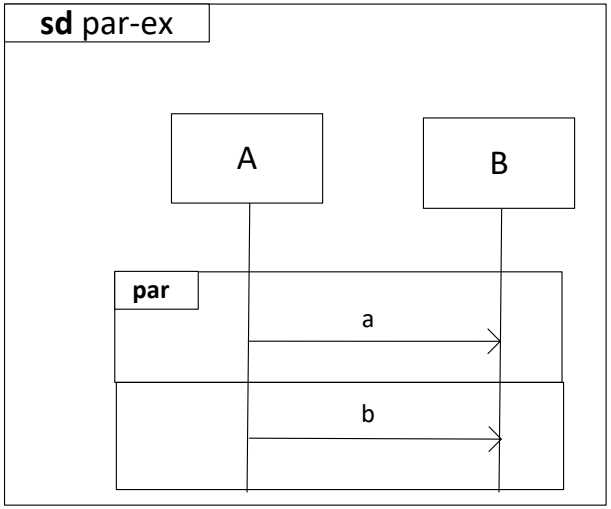
Summary of sequence diagrams – positive behavior II

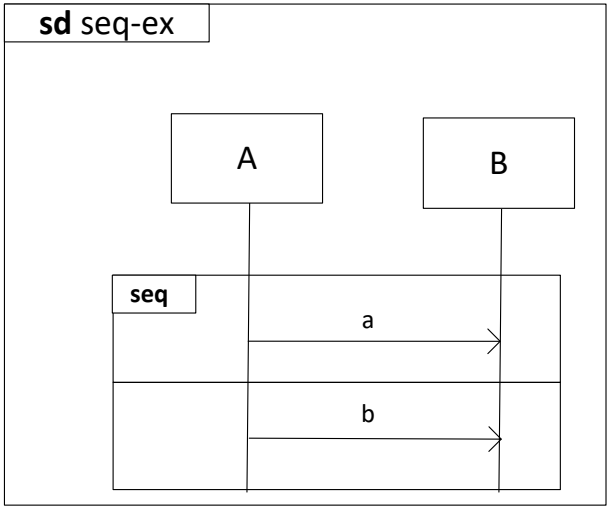




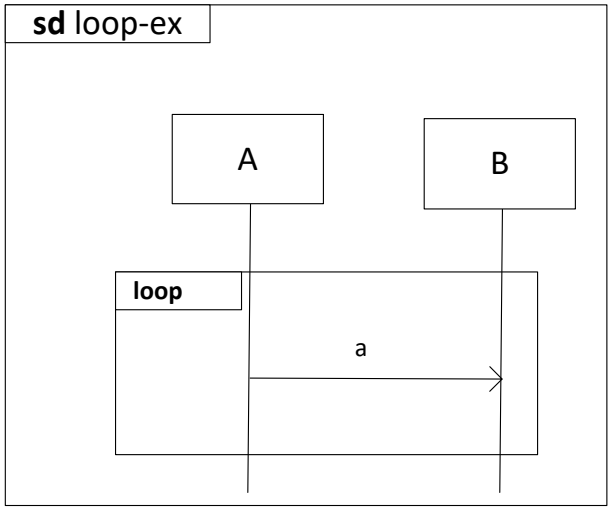
Summary of sequence diagrams – positive behavior III

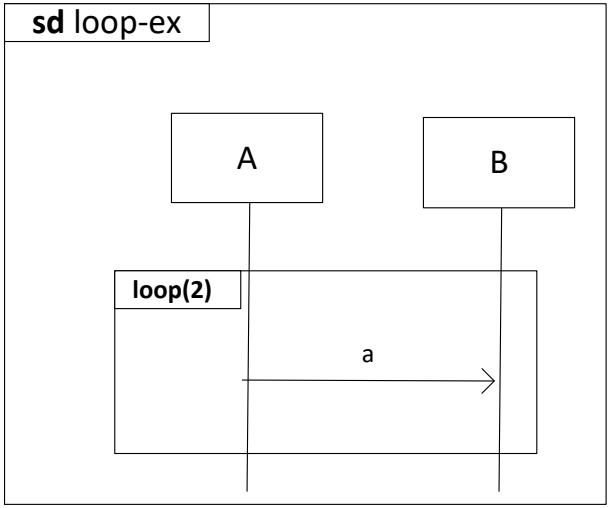


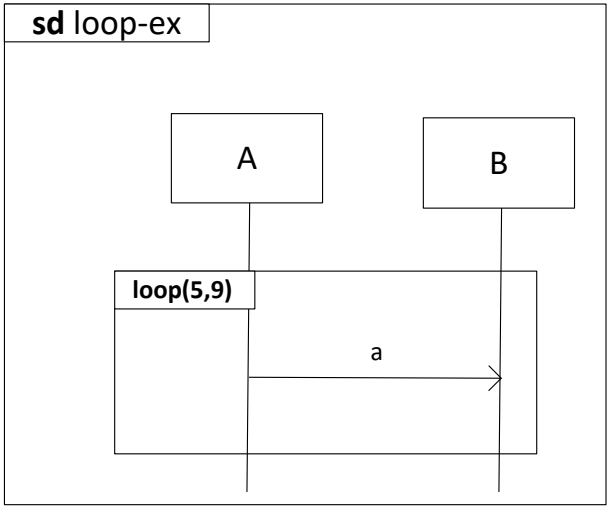




Summary of sequence diagrams – positive behavior IV

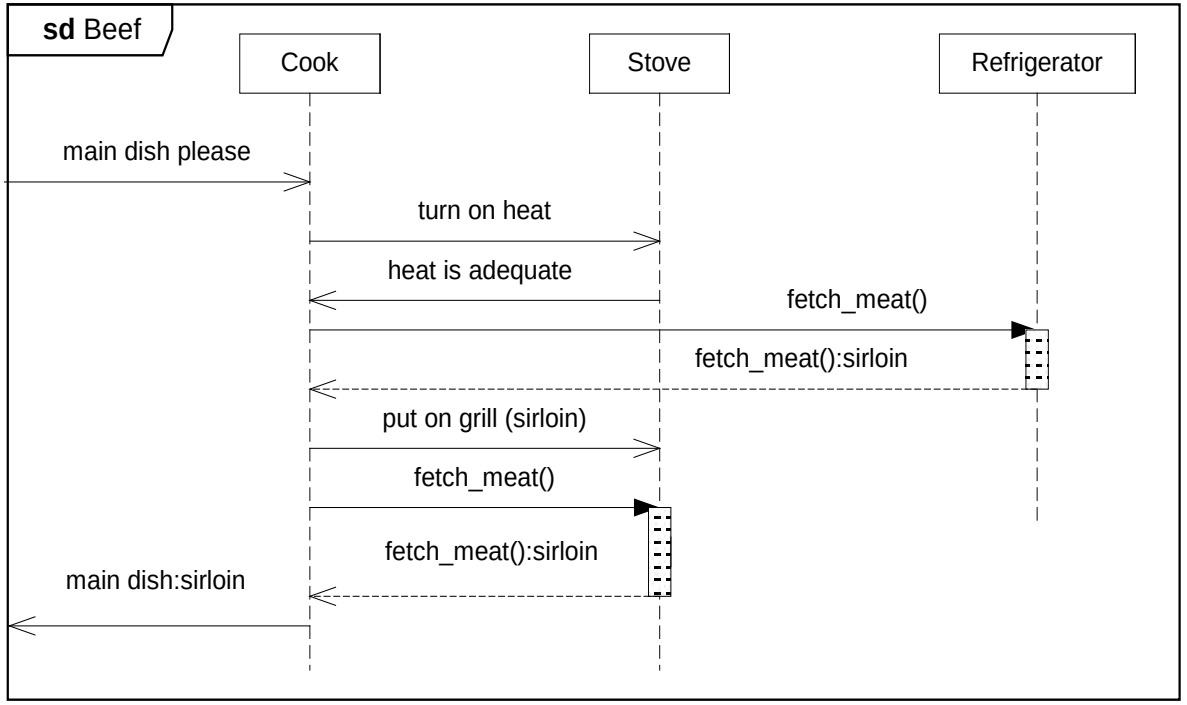




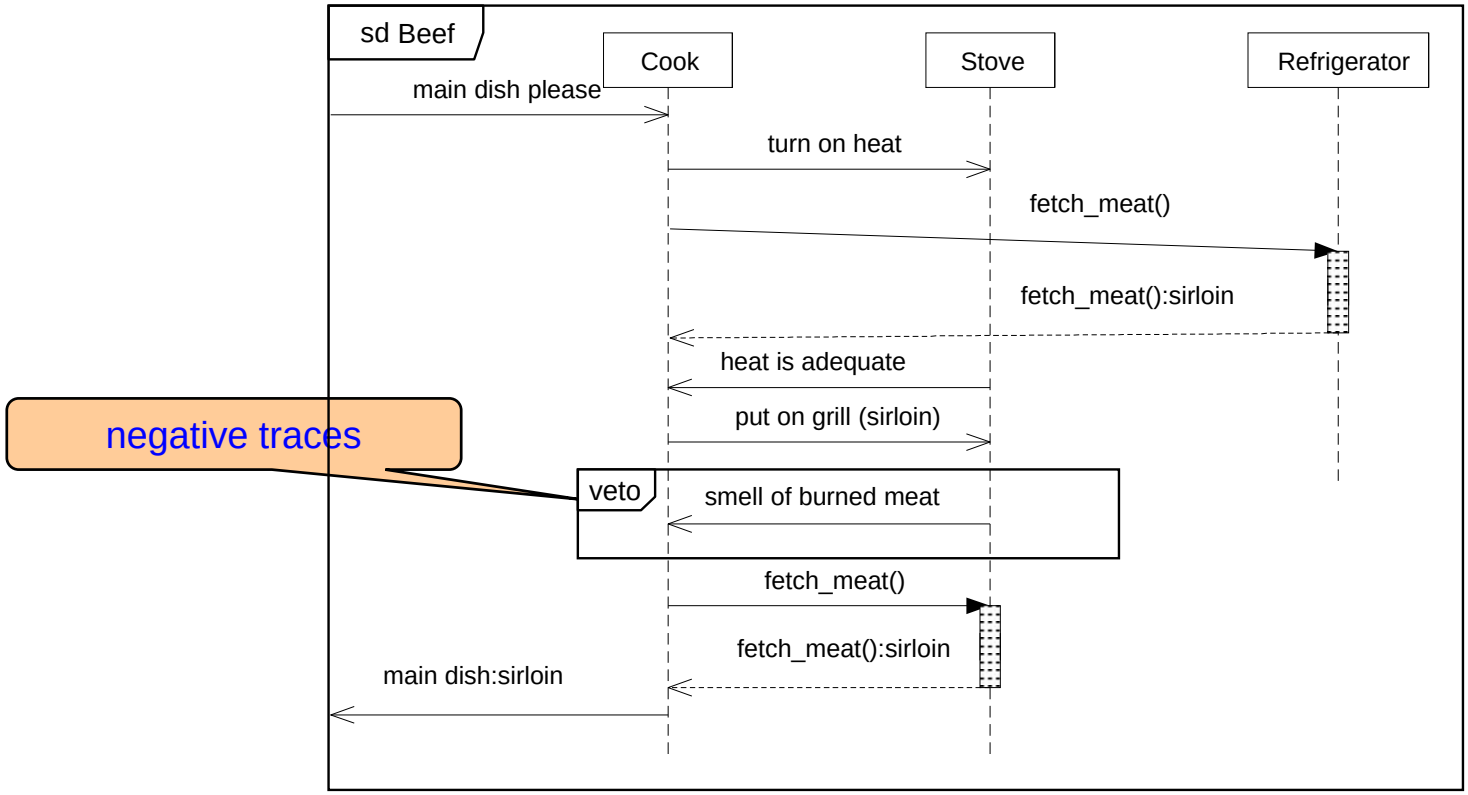


Negative behaviour

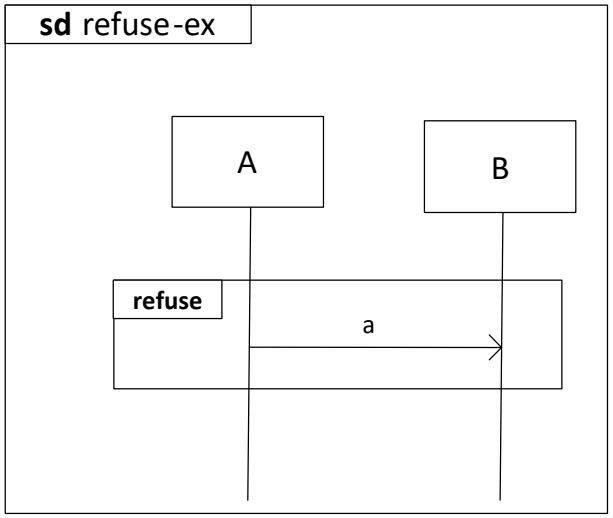
Ordering Beef

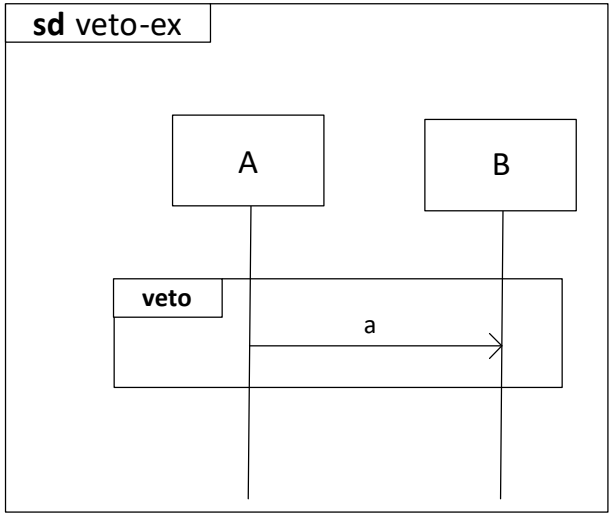


Ordering Beef also including negative behavior



veto and **refuse**





({},{<!a,?a>})

({<>},{<!a,?a>})

Negative behavior due to guards