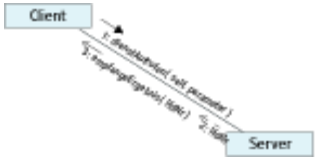
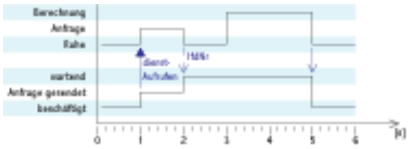
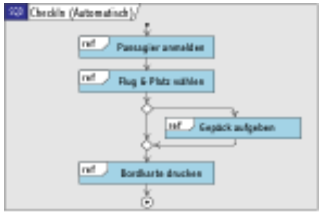
Unified Modeling Language 2

Interactions

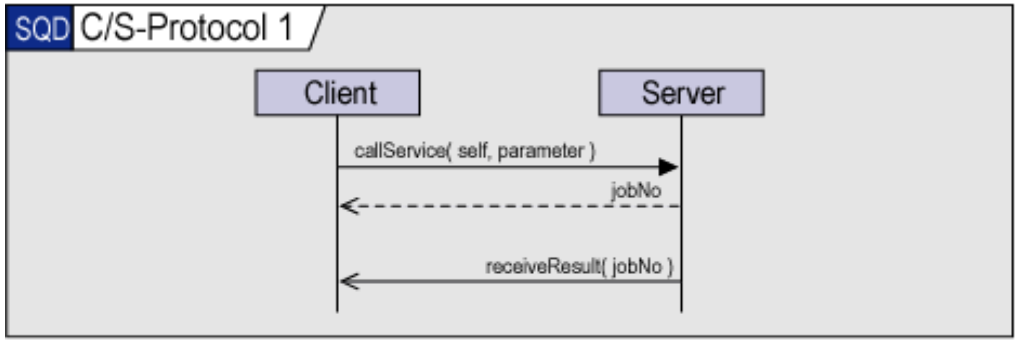




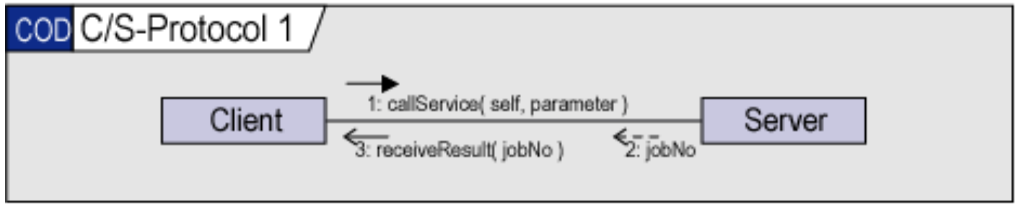


A first glimpse

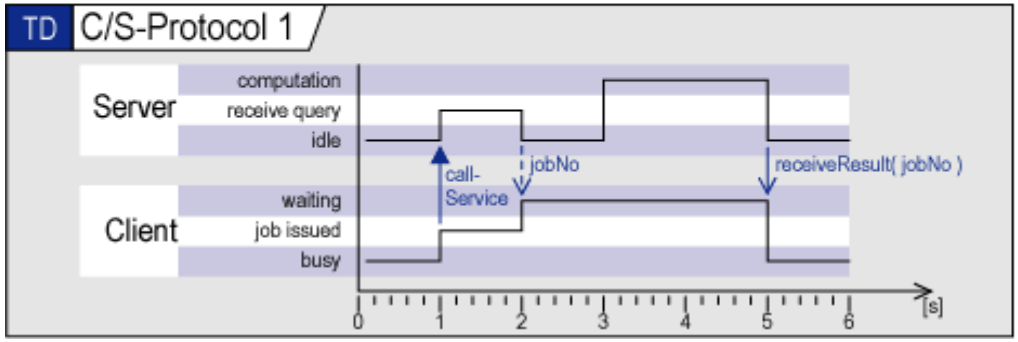
sequence diagram



communication diagram



timing diagram



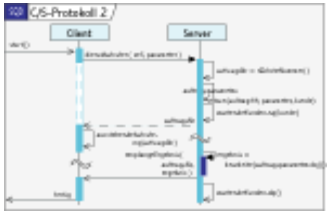
all three are semantically equivalent

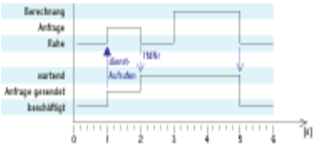
History and predecessors

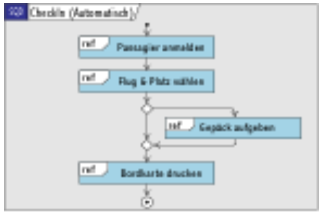
- Simple sequence diagrams
  - 1990‘s
    - Message Sequence Charts (MSCs) used in TelCo-industry
    - several OO-methods use sequence diagrams
- Complex sequence diagrams
  - 1996: Complex MSCs introduced in standard MSC96
  - 1999: Life Sequence Charts (LSCs)
- Communication diagrams
  - 1991: used in Booch method
  - 1994: used in Cook/Daniels: Syntropy
- Timing diagrams
  - traditionally used in electrical engineering
  - 1991: used in Booch method
  - 1993: used in early MSCs
- Interaction overview
  - 1996: high-level MSCs (graphs of MSCs as notational alternative)

Usage scenarios

- Class/object interactions
  - design or document message exchange between objects
  - express synchronous/asynchronous messages, signals and calls, activation, timing constraints
- Use case scenarios
  - illustrate a use case by concrete scenario
  - useful in design/documentation of business processes (i.e. analysis phase and reengineering)
- Test cases
  - describe test cases on all abstraction levels
- Timing specification/documentation
- Interaction overview
  - organize a large number of interactions in a more visual style
  - defined as equivalent to using interaction operators



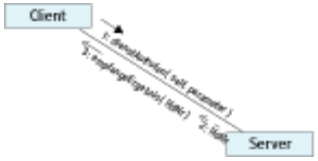




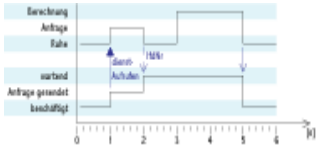
Syntactical variants

- Sequence diagram
  - traditional sequence diagrams + interaction operators
  - focuses on exchanging many messages in complex patterns among few interaction partners

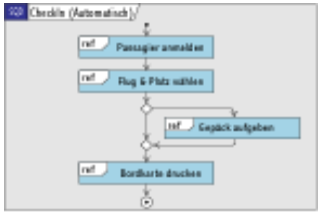
- Communication diagram
  - “collaboration diagram” in UML 1.x
  - focuses on exchanging few messages between (many) interaction partners in complex configuration



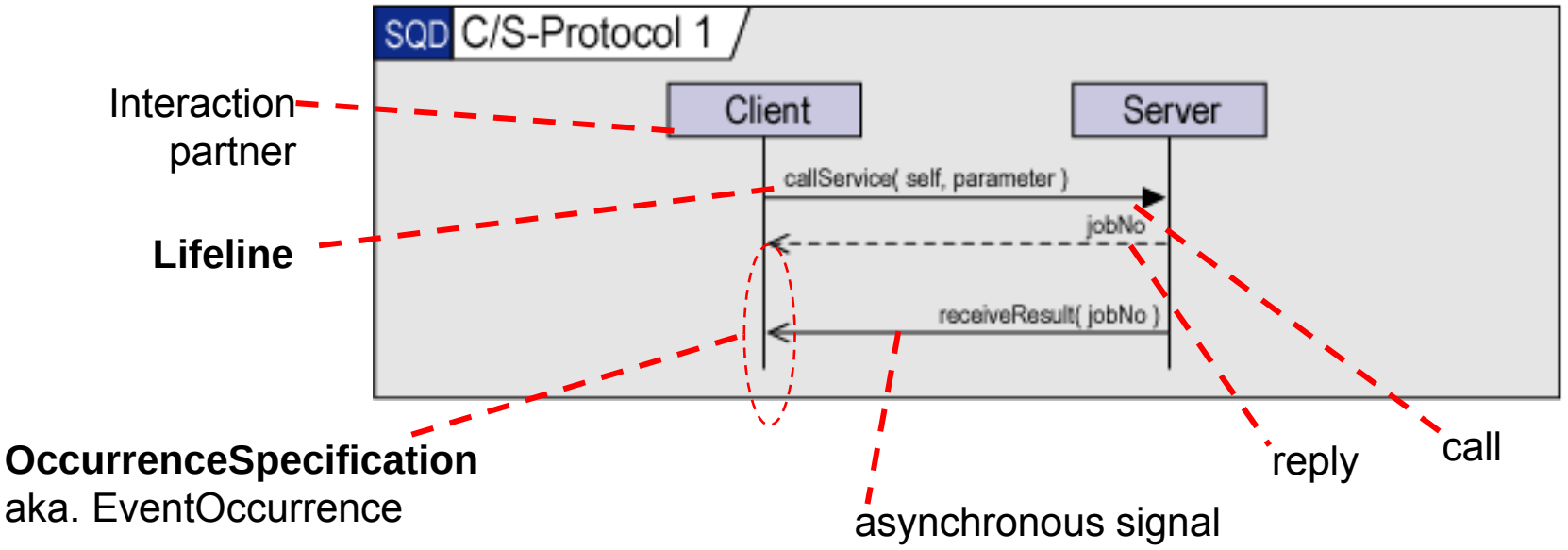
- Timing diagram
  - new in UML 2.0, oscilloscope-type representation, not necessarily metric time
  - focuses on (real) time and coordinated state change of interaction partners over time



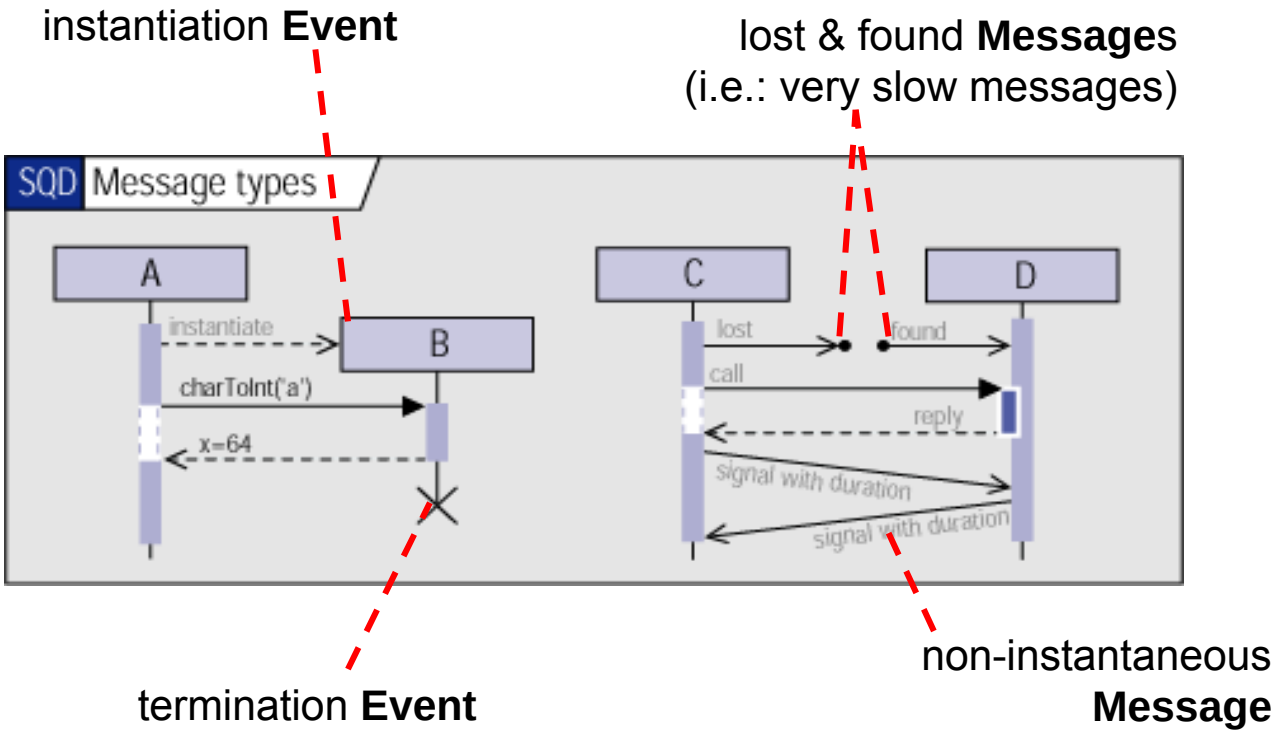
- Interaction overview diagram
  - looks like restricted activity diagram, but isn’t
  - arrange elementary interactions to highlight their interaction



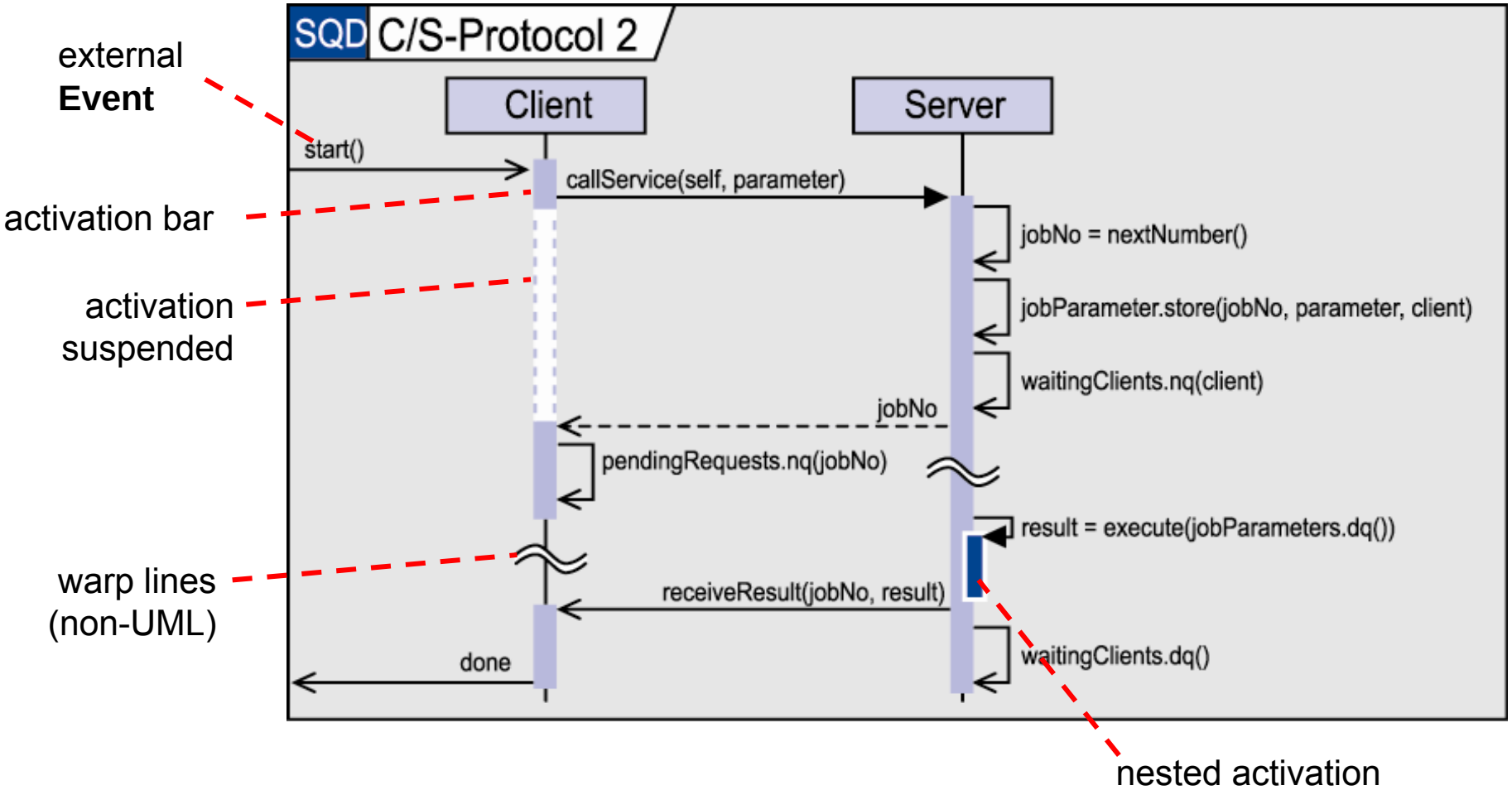
Main concepts



Message types

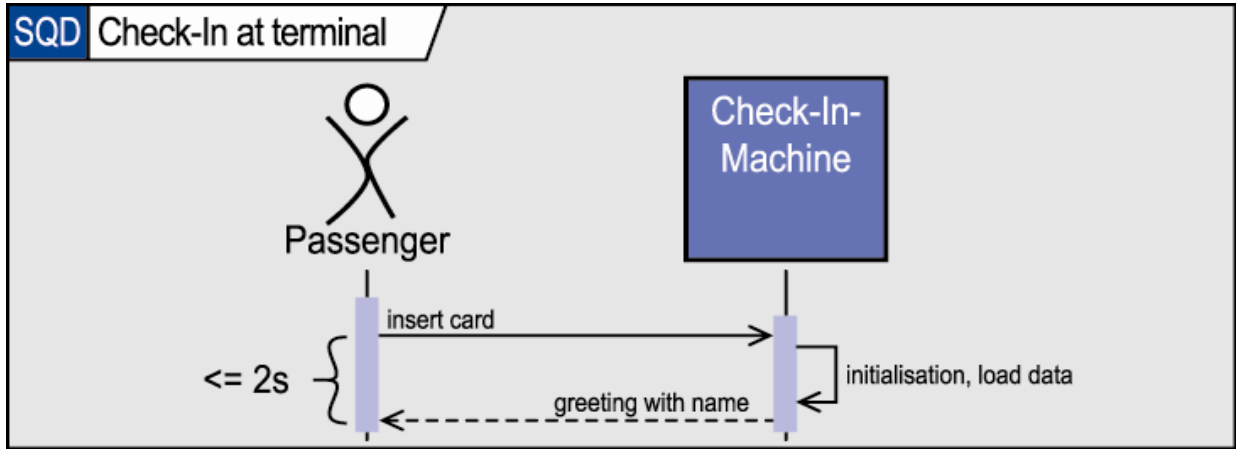


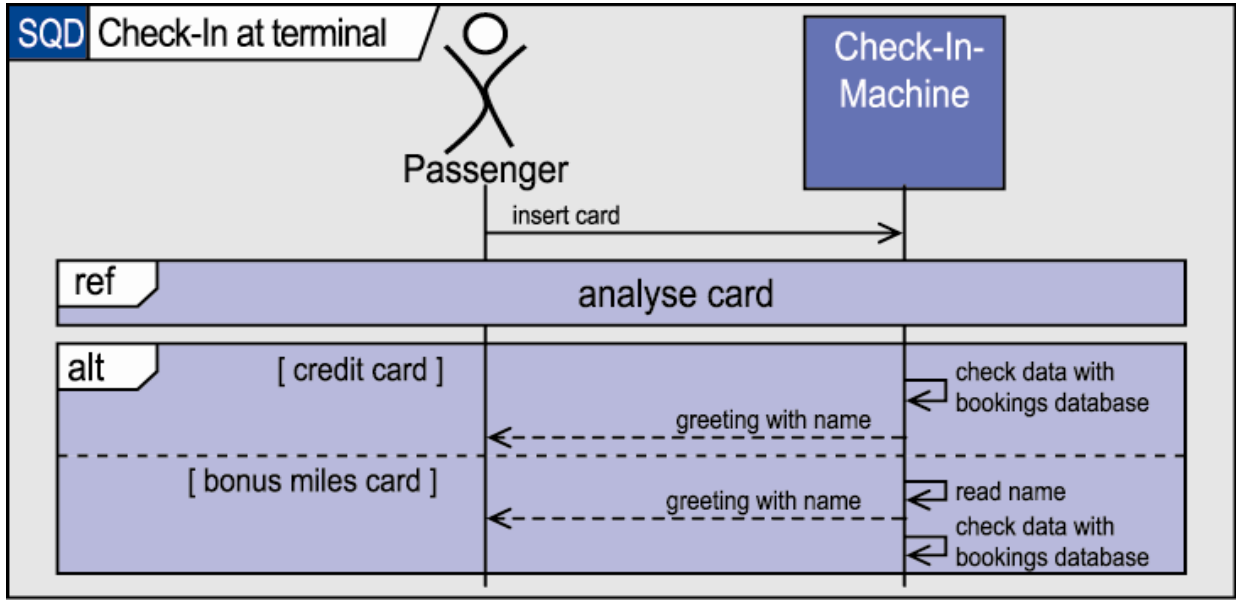
Activation


external
Event
activation bar
activation
suspended
warp lines
(non-UML)
SQD C/S-Protocol 2
Client
Server
start()
callService(self, parameter)
jobNo = nextNumber()
jobParameter.store(jobNo, parameter, client)
waitingClients.nq(client)
jobNo
pendingRequests.nq(jobNo)
result = execute(jobParameters.dq())
receiveResult(jobNo, result)
waitingClients.dq()
done
nested activation

Usage: Use case scenarios

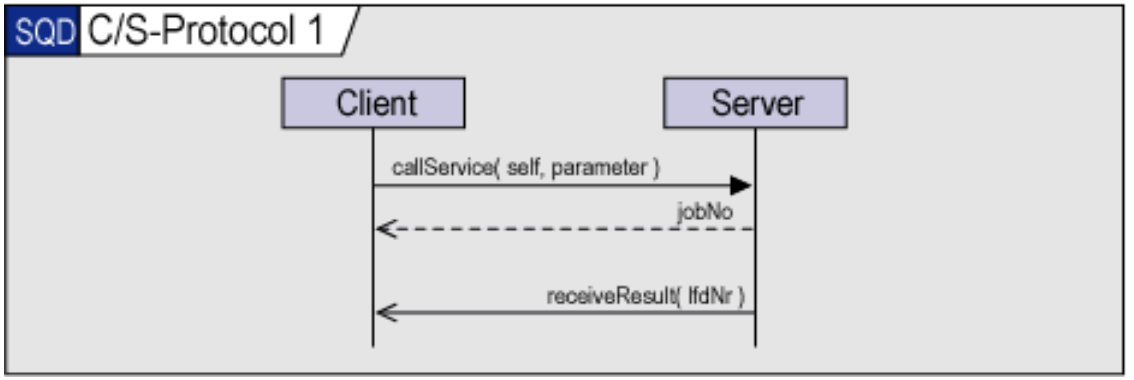
- Interaction **participants** are actors and systems rather than classes and objects.
- May be **refined** successively.
- Useful also for specifying high-level non-functional requirements such as response times.
- All kinds of interaction diagrams may be applied, depending on the circumstances.

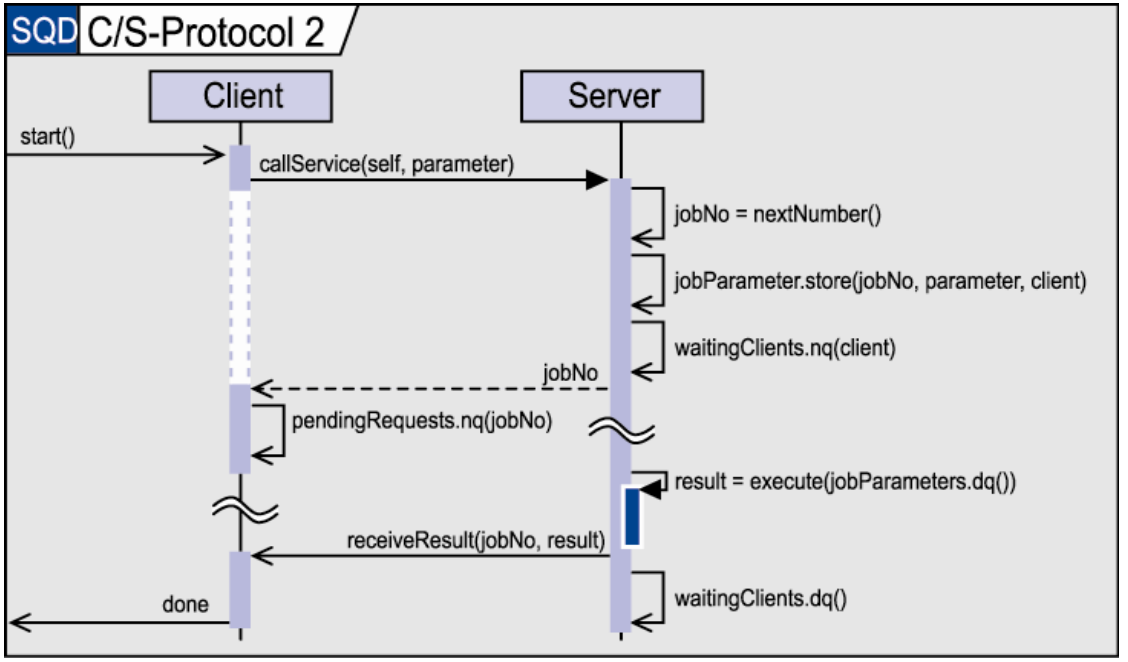




Usage: Class interactions

- Interaction **participants** are classes and objects rather than actors and systems.
- Again, successive **refinement** may be applied in different styles:
  - break down processing of messages
  - break down structure of interaction participants.
- All kinds of interaction diagrams may be applied, depending on the circumstances.





Usage: Test cases

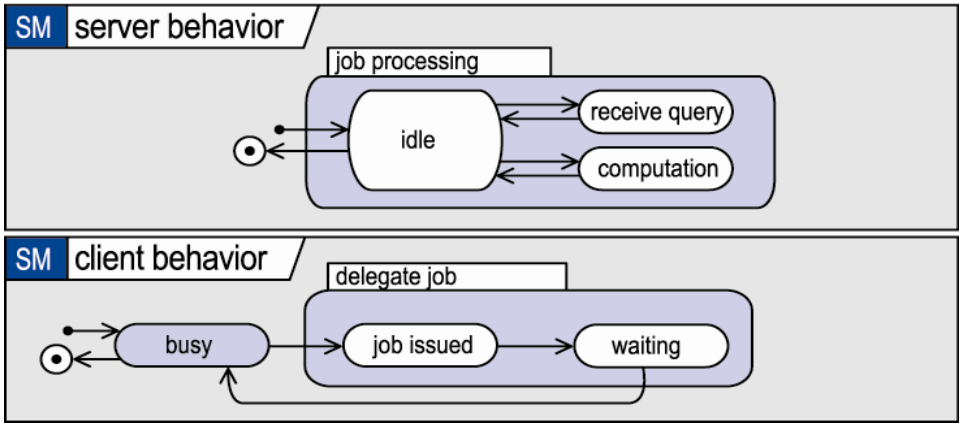
- Like any other interaction, but with a different intention.

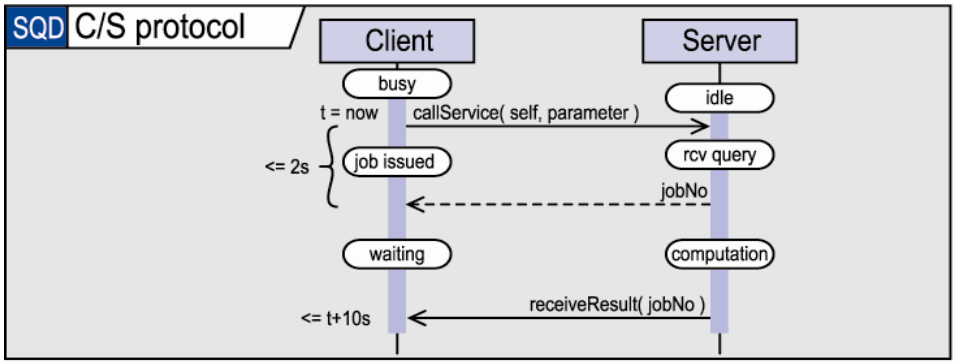
- Typically accompanied by a **tabular description** of purpose, expected parameters and result (similar to use case description).

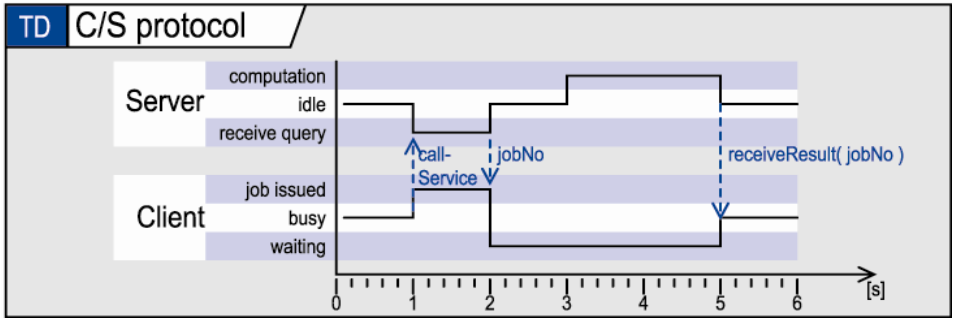
identifier	name
TF-AAA.CIA-4	Check In (automatic) too much luggage
test goal: If a passenger has too many pieces of luggage and tries to check in using the check in machine, he should be referred to the check in counter.	
precondition: passenger is booked on respective flight	
arguments: luggage, bonus mile card, booking data	
result: passenger is referred to counter	
postcondition: luggage is not checked in, passenger is checked in	
remarks, open questions: none	

Usage: Timing specification

- For **embedded** and **real-time** systems, it may be important to specify absolute timings and state evolution over time.
- This is not readily expressed in sequence diagrams, much less communication diagrams.
- UML 2.0 introduces **timing diagrams** for this purpose.

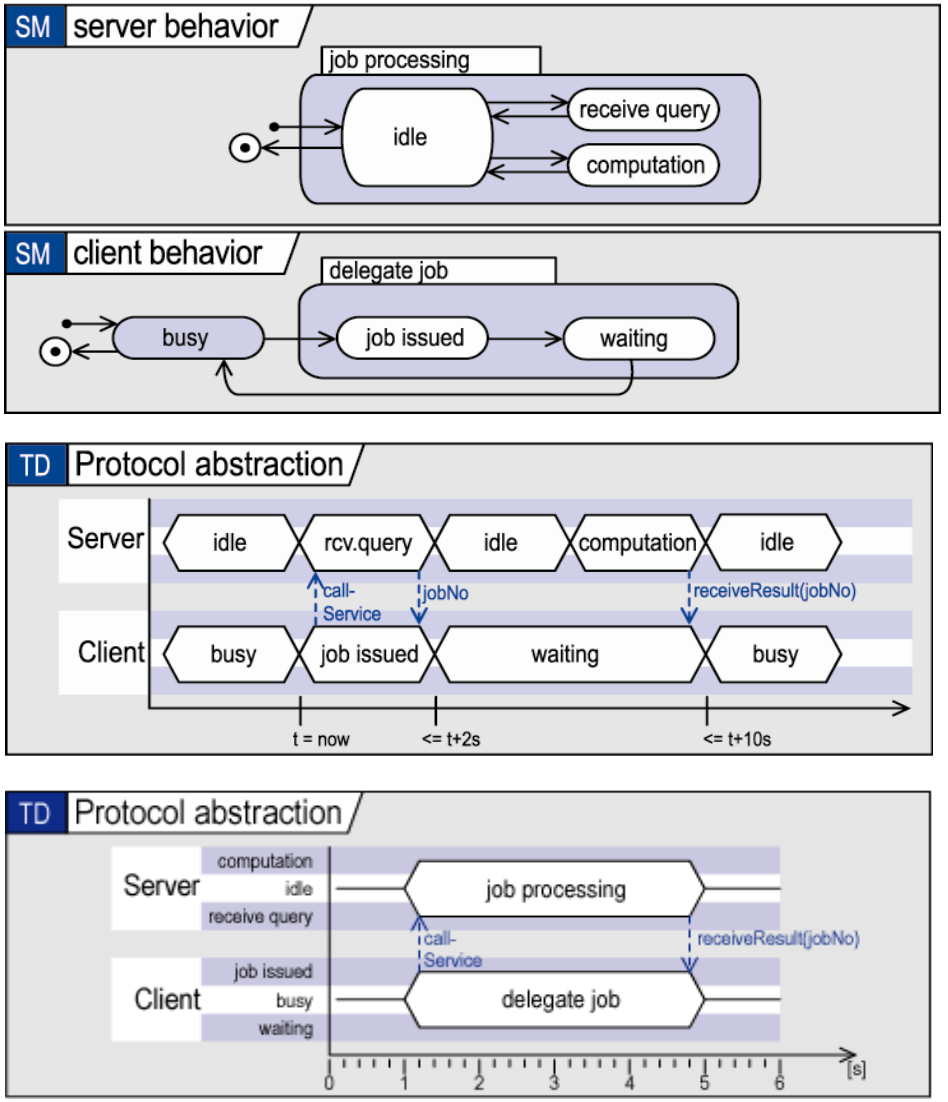






Abstraction in timing diagram

- An alternative syntax presents states not on the vertical axis but as hexagons on the lifeline.
- Timing diagrams present the coordination of (the states of) several objects over (real) time.

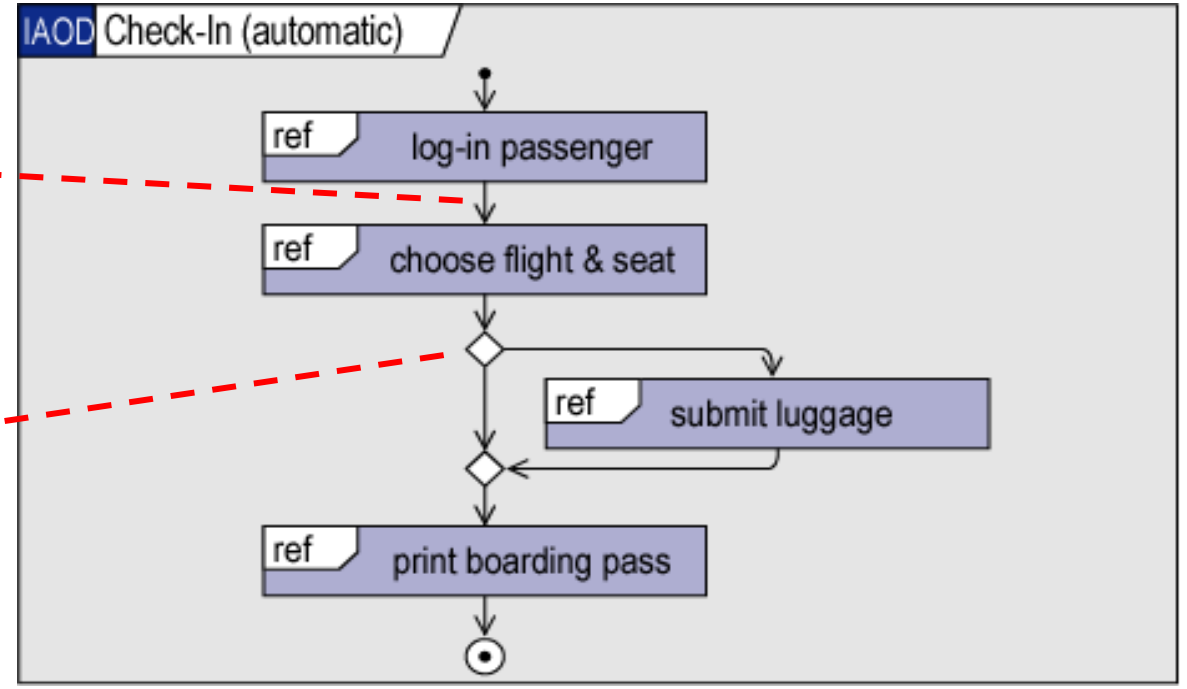


Usage: Interaction overview

- Organize large number of interactions in a more visual style
- Defined as equivalent to using interaction operators

sequence equivalent to seq

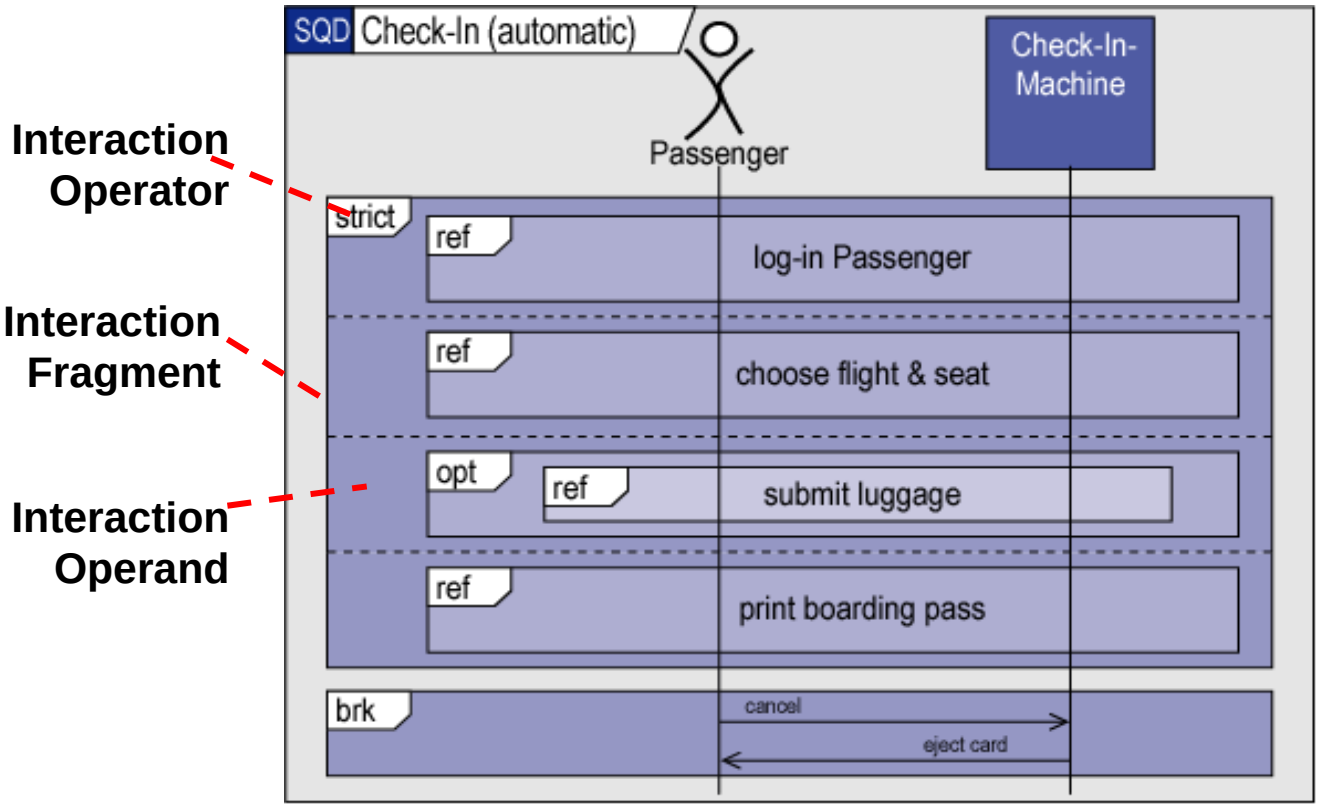
choice/merge
equivalent to alt/opt



also allowed: fork/join
(said to be equivalent to par, but …)

Complex interactions

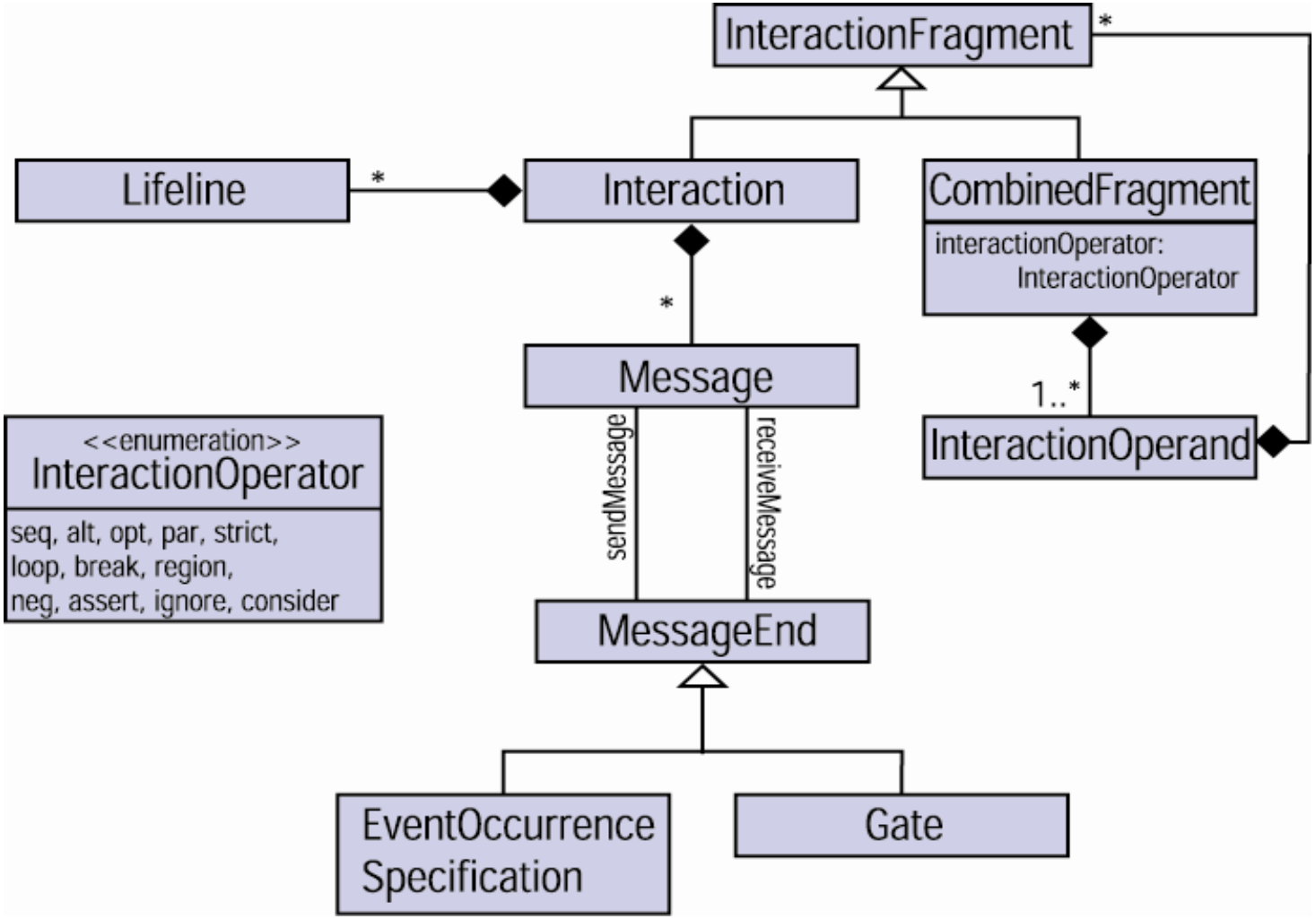
- A complex interaction is like a functional expression:
  - an InteractionOperator,
  - one or several InteractionOperands (separated by dashed lines),
  - (and sometimes also numbers or sets of signals).



Interaction operators (overview)

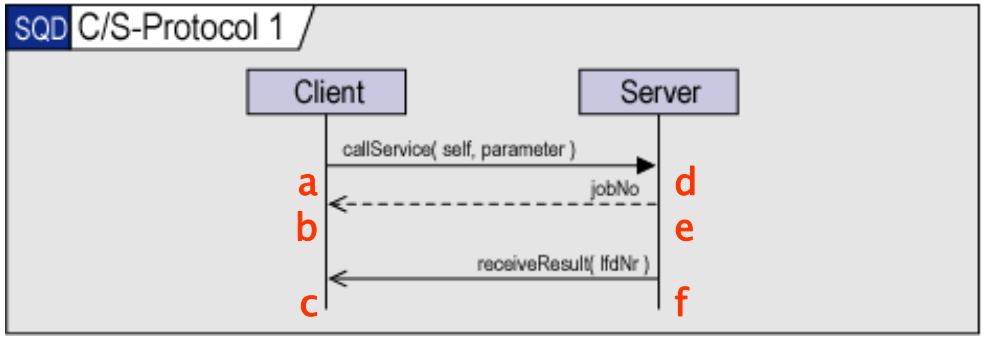
- strict
  - operand-wise sequencing
- seq
  - lifeline-wise sequencing
- loop
  - repeated seq
- par
  - interleaving of events
- region (aka. “critical”)
  - suspending interleaving
- consider
  - restrict model to specific messages
  - i.e. allow anything else anywhere
- ignore
  - dual to consider
- ref
  - macro-expansion of fragment
- alt
  - alternative execution
- opt
  - optional execution
  - syntactic sugar for alt
- break
  - abort execution
  - sometimes written as “brk”
- assert
  - remove uncertainty in specification
  - i.e. declare all traces as valid
- neg
  - declare all traces as invalid
    (→ three-valued semantics)

Main concepts (metamodel)



Semantics

- The meaning of an interaction is
  - a set of valid traces, plus
  - a set of invalid traces.
- Traces are made up of occurrences of events such as
  - sending/receiving a message,
  - instantiating/terminating an object, or
  - time/state change events.
- Two types of constraints determine the valid traces:
  1) send occurs before receive,
  2) order on lifelines is definite.



This diagram contains the following seven constraints:

1) a→d, e→b, f→c
2) a→b, b→c, d→e, e→f

The set of resulting traces is:
{ a.d.e.b.f.c, a.d.e.f.b.c }.