WHITESTEIN Technologies

Unified Modeling Language

Interactions

Radovan Cervenka

Interaction Model

→ **Defines the mutual interactions and collaboration of objects in certain situations.**

Diagram types:

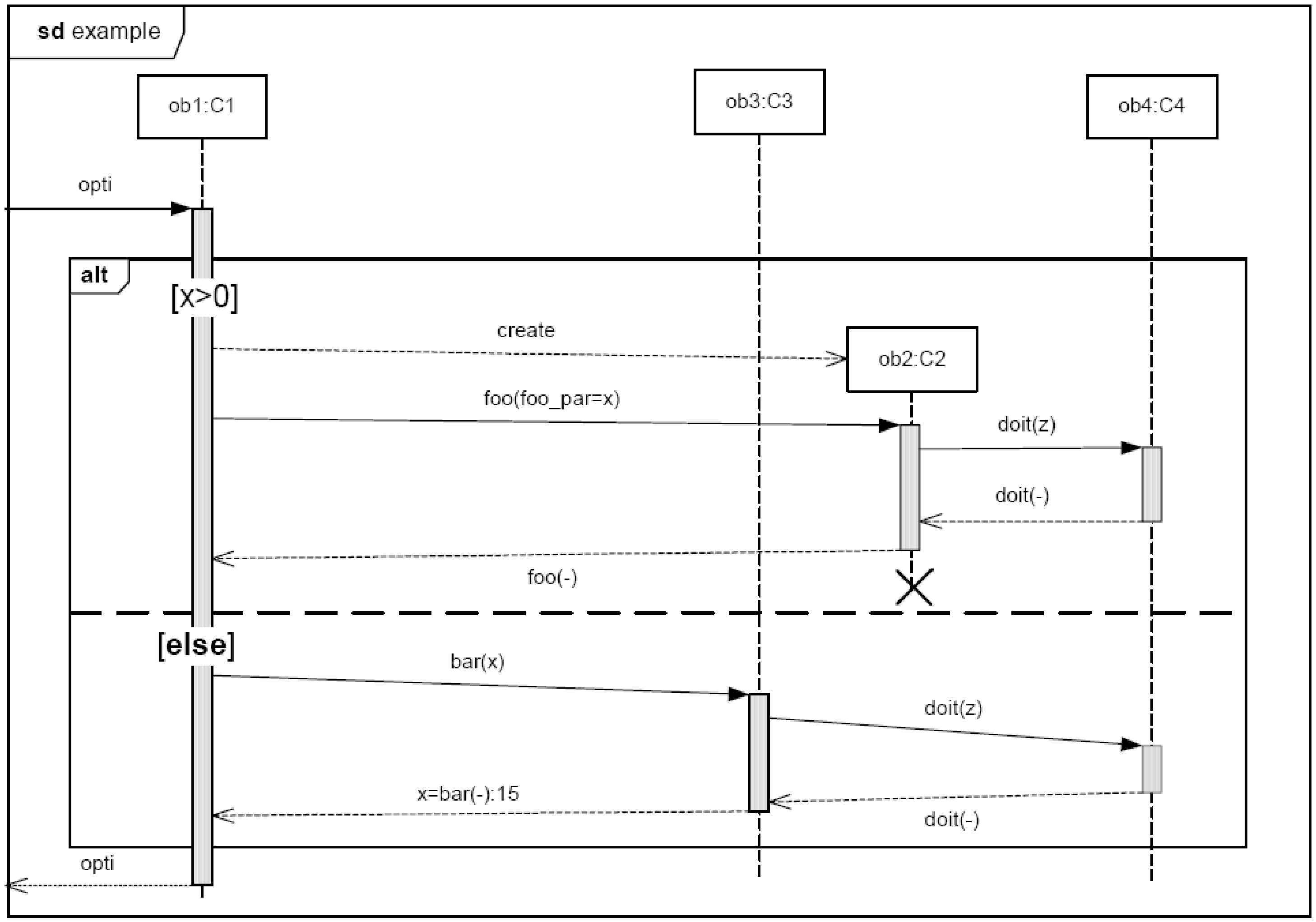
- Sequence diagrams.
- Interaction overview diagrams.
- Communication diagrams.
- Timing diagrams.
- Interaction tables.

Used (mainly) in:

- Analysis and Design $\Rightarrow$ use case realization and interaction of objects.

Sequence Diagram

→ An interaction diagram which focuses on the message interchange between a number of lifelines.

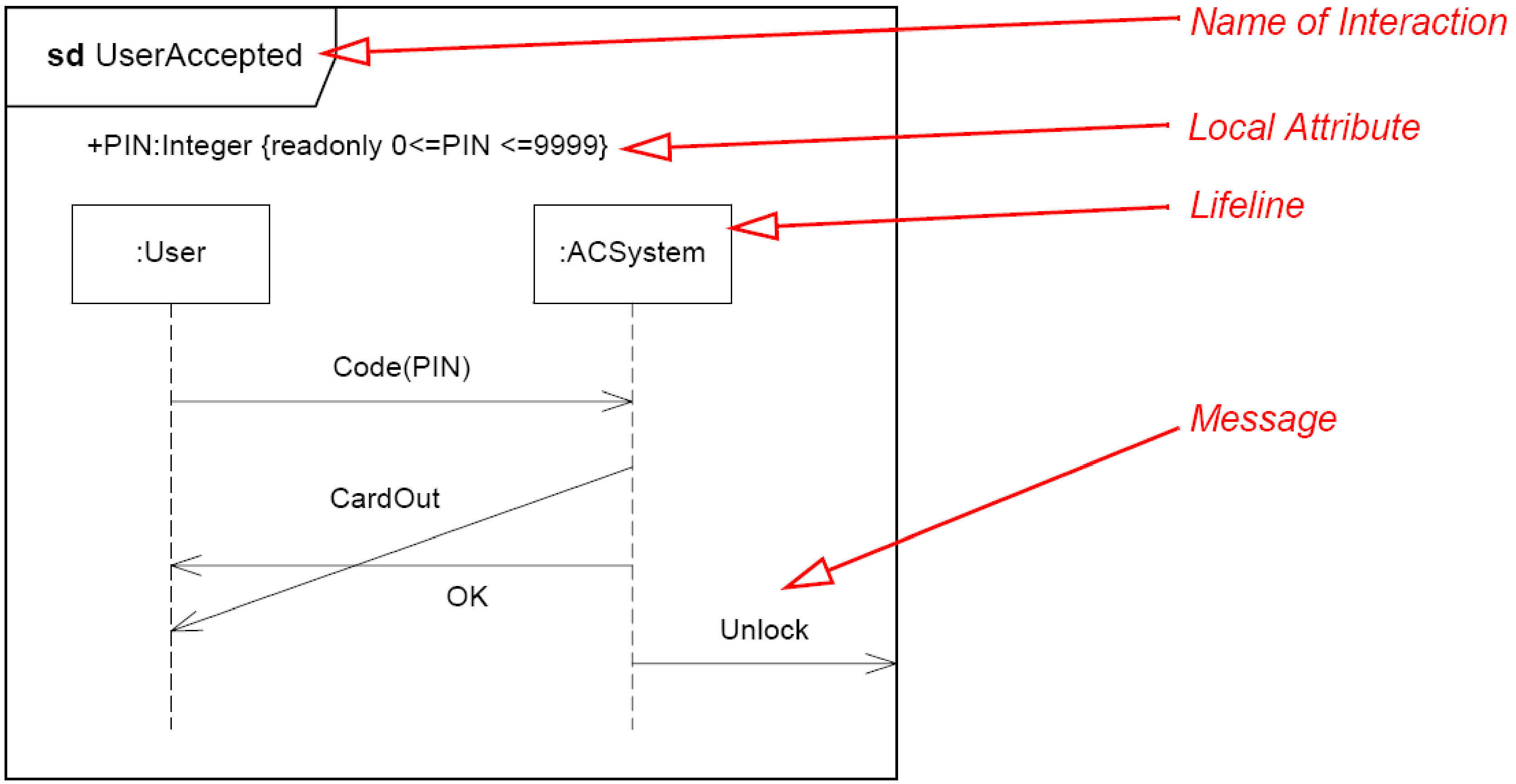


Interaction

→ A unit of behavior that focuses on the observable exchange of information between connectable elements.

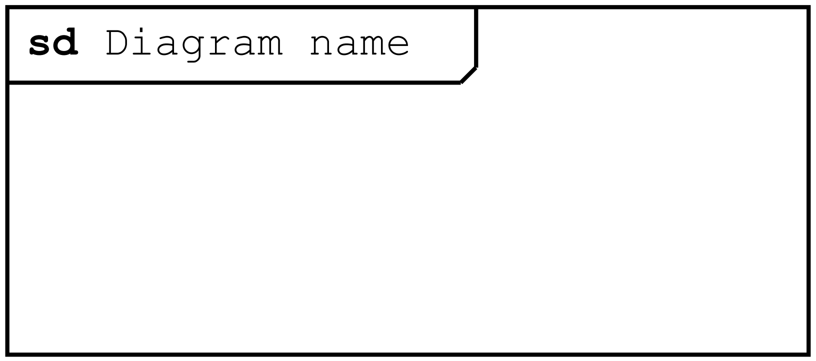
- Focus on exchanging messages between the connectable elements (of the classifier owning the interaction).
- An interaction comprises:
  - lifelines,
  - messages,
  - interaction fragments,
  - formal gates, and
  - actions.
- The semantics expressed by ***valid*** and ***invalid traces***.
  - A ***trace*** is a sequence of event occurrences (such as send operation/signal event, receive operation/signal event or destruction event) in a model.
- A specialization of interaction fragment and of behavior.

Example of Interaction



Frame

- → A rectangular frame around the diagram with a name in a compartment in the upper left corner.
- ■ “sd” is used to determine the interaction diagram.
- ■ Is used in all kinds of interaction diagrams.



Lifeline

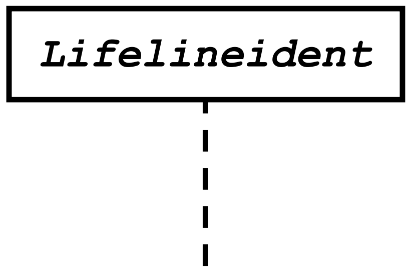
→ Represents an individual participant in the interaction.

- A reference to a connectable element.
- Lifelines represent only one interacting entity; have multiplicity 1.
    - If the referenced connectable element is multivalued, then the lifeline may have the ***selector*** that specifies which particular part is represented by this lifeline. If the selector is omitted, an arbitrary representative of the multivalued connectable element is chosen.
- Can refer to the interaction that represents the decomposition.
- Name format:

lifelineident ::= ([*connectable-element-name* [‘[‘ *selector* ‘]’]]
[: *class_name*] [*decomposition*]) | ‘self’

selector ::= *expression*

decomposition ::= ‘ref’ *interactionident* [‘strict’]

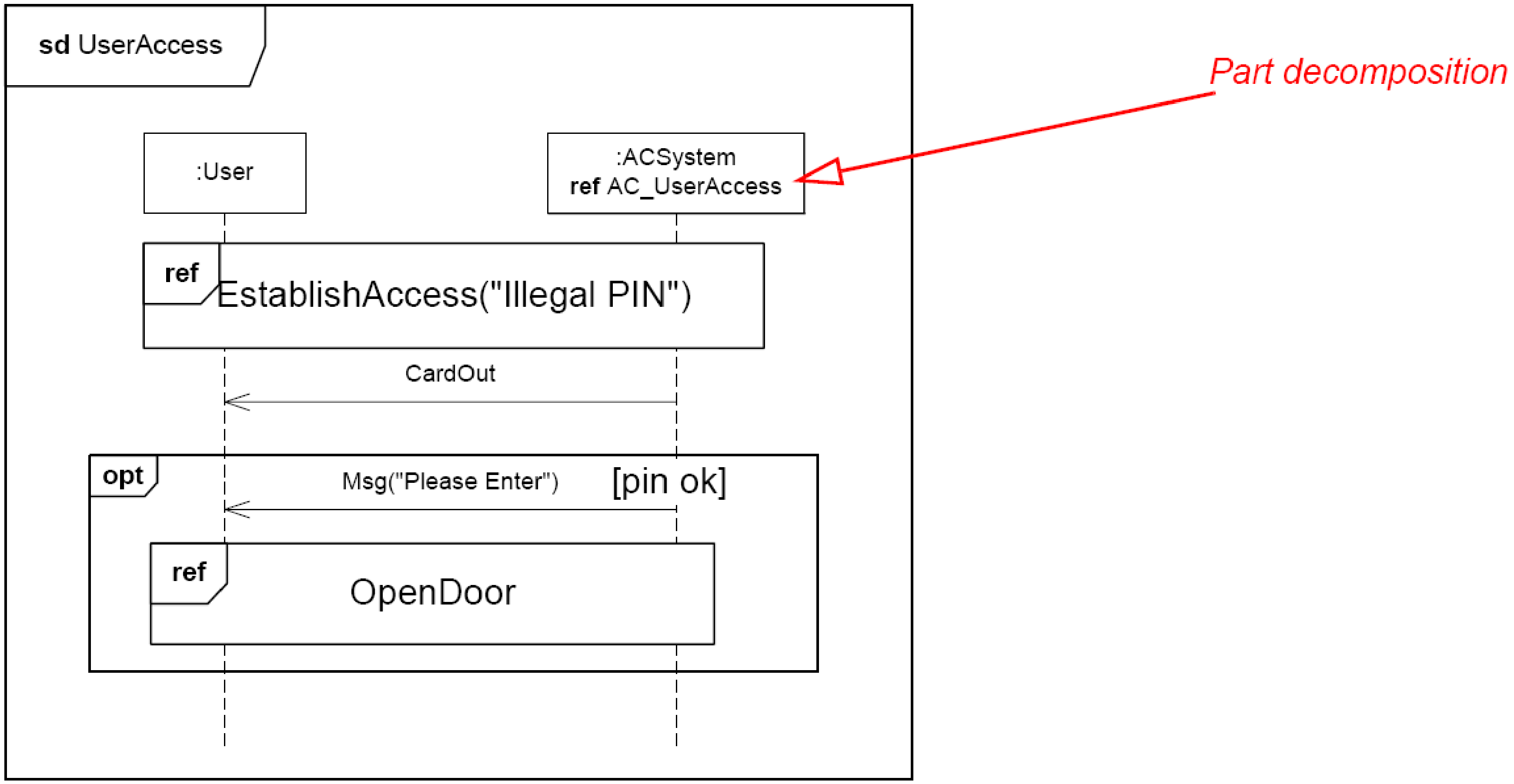


Part Decomposition

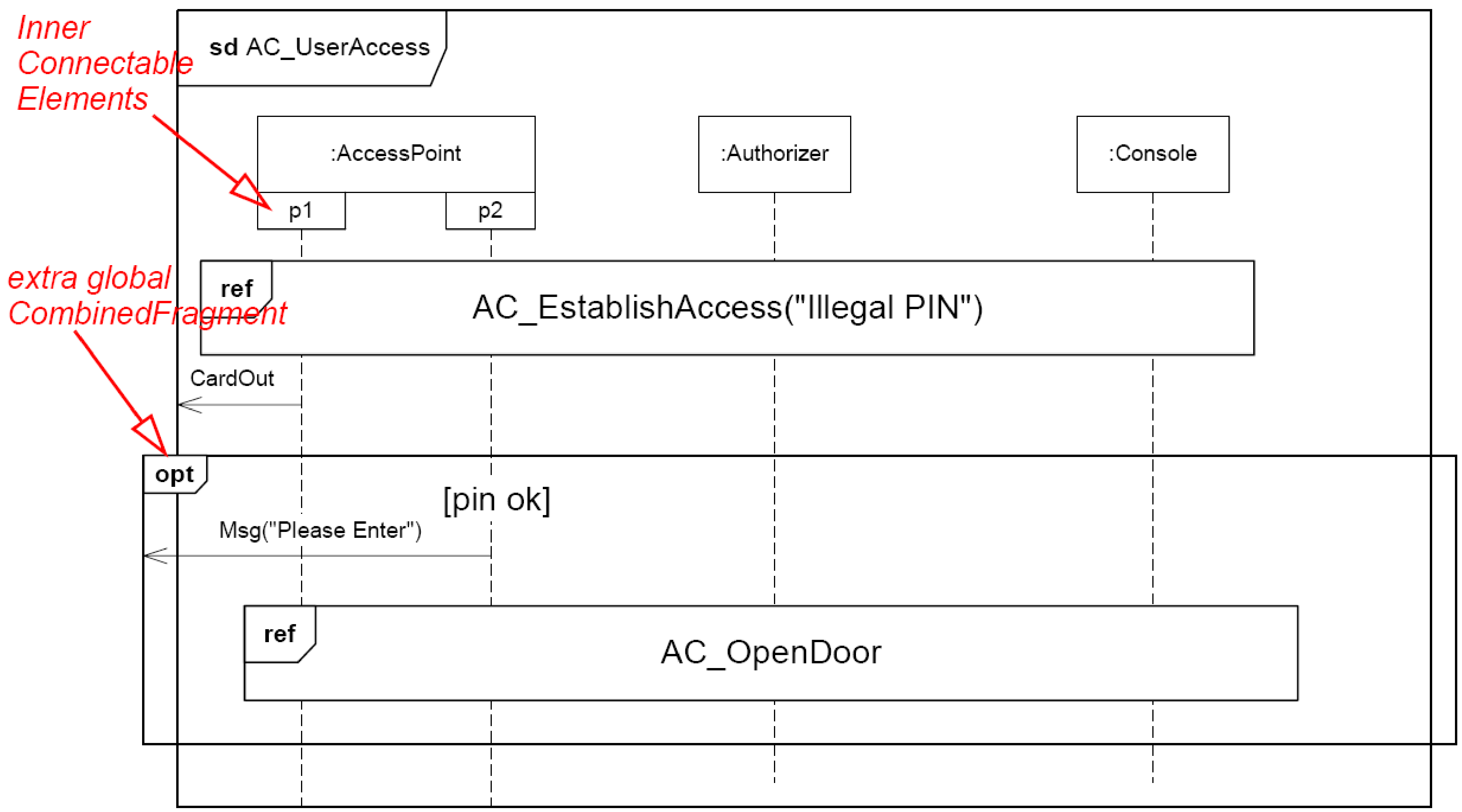
→ A description of the internal interaction of one lifeline relative to an interaction.

- A Lifeline has a class associated as the type of the connectable element that the lifeline represents. That class may have an internal structure and the part decomposition is an interaction that describes the behavior of that internal structure relative to the Interaction where the decomposition is referenced.
- The messages that go into (or go out from) the decomposed lifeline are interpreted as actual gates that are matched by corresponding formal gates on the decomposition.

Examples of Part Decompositions (1)

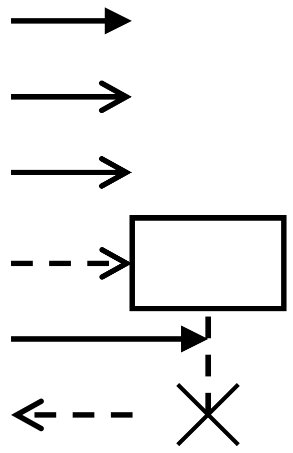


Examples of Part Decompositions (2)



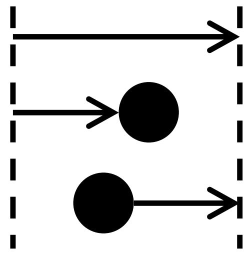
Message

- Defines a particular communication between lifelines of an interaction.
- Specifies the kind of communication, sender and the receiver.
- Sorts of message:
  - ***Synchronous call*** of an operation.
  - ***Asynchronous call*** of an operation.
  - ***Asynchronous signal*** – an asynchronous send action.
  - ***Create message*** – the creation of another lifeline object.
  - ***Delete message*** – the termination of another lifeline.
  - ***Reply*** – a reply message to an operation call.



Message (cont.)

- Message connects *send event* and *receive event*.
- Kinds of a message:
  - ***Complete*** – send event and receive event are present.
  - ***Lost*** – send event present and receive event absent.
  - ***Found*** – send event absent and receive event present.
  - ***Unknown*** – send event and receive event absent (should not appear).



- Format of message name:

  messageident ::= ([*attribute* '=’] *signal-or-operation-name*
  ['(' [*argument* [',’*argument*]* ')’] [':’ *return-value*]) | '*’

  argument ::= ([*parameter-name* '='] *argument-value*) |
  (*attribute* '=' *out-parameter-name* [':' *argument-value*]) | ' -'

  - '*’ is a shorthand for more complex alternative combined fragment to represent a message of any type.
  - '-’ is used for undefined arguments.

Examples of Message Names

mymessage(14, - , 3.14, “hello”) // second argument is undefined

v=mymsg(16, variab):96 // a reply message assigning the return value 96 to v

mymsg(myint=16) // the input parameter ‘myint’ is given the value 16

Combined Fragment

→ Defines an “expression” of interaction fragments.

- A combined fragment is defined by an ***interaction operator*** and corresponding ***interaction operands***.
- A compact and concise manner to define a number of traces.
- An *interaction operand* is an interaction fragment with an optional guard expression.
  - Only the interaction operands with a guard evaluated to true at this point in the interaction will be considered for the production of the traces for the enclosing combined fragment.
  - The order of the interaction operands is given by their vertical positions.
- ***Alternatives (alt)***
  - A choice of behavior. At most one of the operands will be chosen.
  - The chosen operand must have an explicit or implicit guard expression that evaluates to true. (An implicit true guard is implied if the operand has no guard.)
  - An operand guarded by ***else*** is applied if no other operand is chosen.

Combined Fragment (cont.)

- ***Option (opt)***
  - A choice of behavior where either the (sole) operand happens or not.
  - Semantically equivalent to an alternative combined fragment where there is one operand.
- ***Break (break)***
  - The operand is a scenario that is performed instead of the remainder of the enclosing interaction fragment.
  - When the guard of the break operand is false, the break operand is ignored and the rest of the enclosing interaction fragment is chosen.
  - Should cover all lifelines of the enclosing interaction fragment.
- ***Parallel (par)***
  - A parallel merge between the behaviors of the operands.
  - The occurrence specifications of the different operands can be interleaved in any way as long as the ordering imposed by each operand as such is preserved.
  - *Coregion* is used as a simplified form of the parallel fragment.

Combined Fragment (cont.)

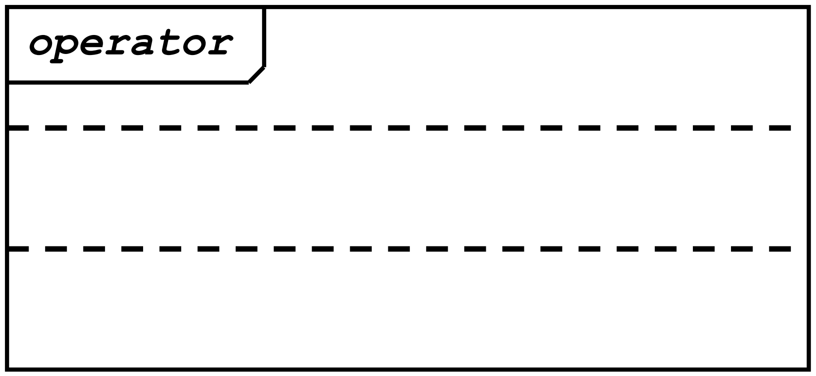
- ***Weak sequencing (seq)***
  - The ordering of occurrence specifications within each of the operands are maintained in the result.
  - Occurrence specifications on different lifelines from different operands may come in any order.
  - Occurrence specifications on the same lifeline from different operands are ordered such that an occurrence specification of the first operand comes before that of the second operand.
- ***Strict sequencing (strict)***
  - A strict ordering of the operands on the first level.
  - Therefore occurrence specifications within contained combined fragments will not directly be compared with other occurrence specifications of the enclosing combined fragment.
- ***Negative (neg)***
  - The combined fragment represents traces that are defined to be invalid.

Combined Fragment (cont.)

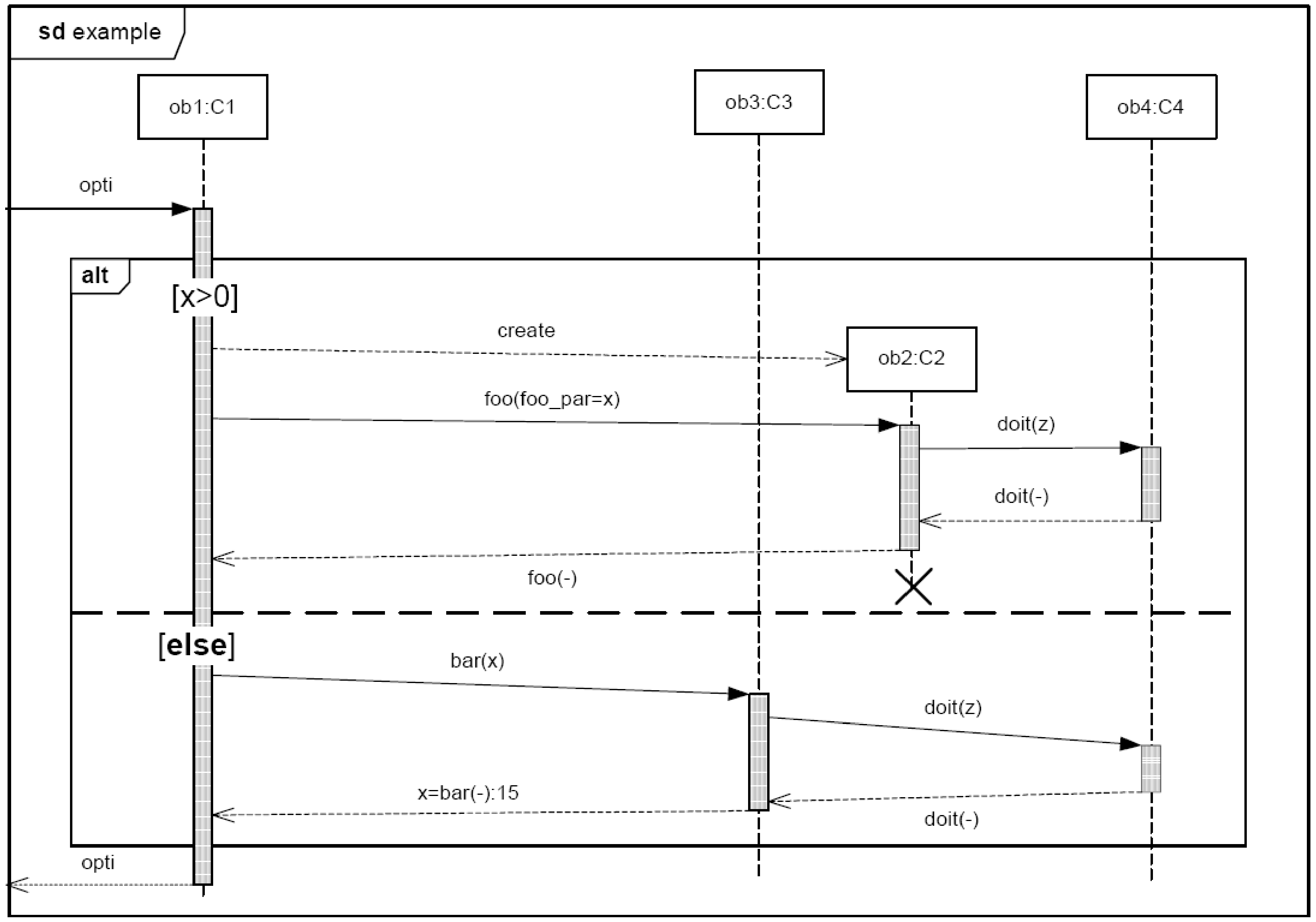
- ***Critical region (critical)***
  - The traces of the region cannot be interleaved by other occurrence specifications (on those lifelines covered by the region).
  - Therefore, the enclosed occurrence specifications must be continuous.
  - Used mainly within parallel combined fragments.
- ***Ignore / Consider (ignore / consider)***
  - Combined with other interaction operators.
  - Ignore = there are some message types that are *not shown* within this combined fragment. These message types can be considered *insignificant* and are *implicitly ignored* if they appear in a corresponding execution.
  - Consider = designates which messages should be considered within this combined fragment. This is equivalent to defining every other message to be ignored.
  - Format: ('ignore' | 'consider') '{' *message-name* [',' *message-name*]* '}'

Combined Fragment (cont.)

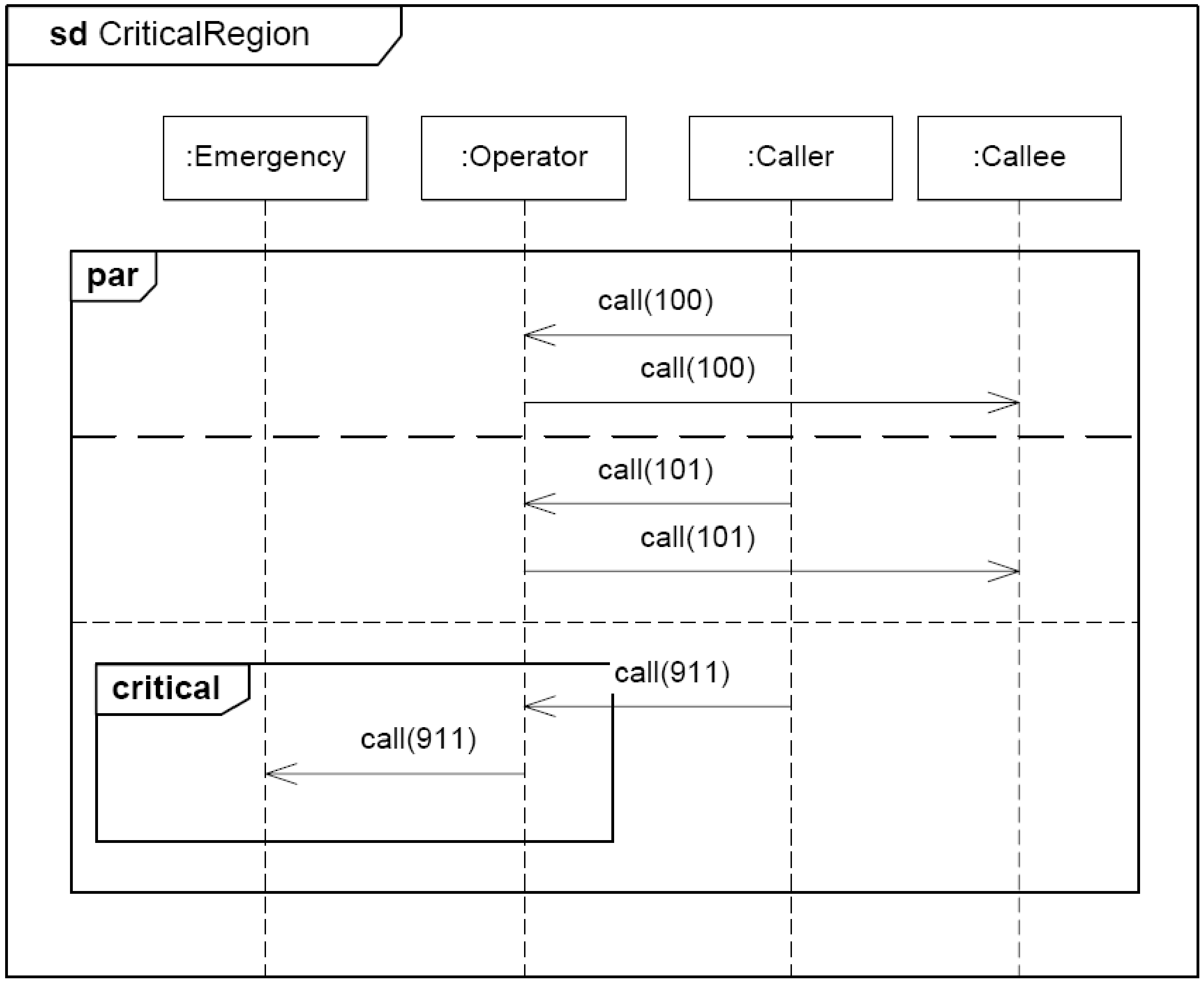
- *Assertion (assert)*
  - The sequences of the operand of the assertion are the only valid continuations.
  - All other continuations result in an invalid trace.
  - Often combined with Ignore or Consider.
- *Loop (loop)*
  - The loop operand will be repeated a number of times.
  - The guard may include a lower and an upper number of iterations of the loop as well as a boolean expression.
  - Format: ‘loop’ [‘(‘ *min* [‘,’ *max*] ‘)’]

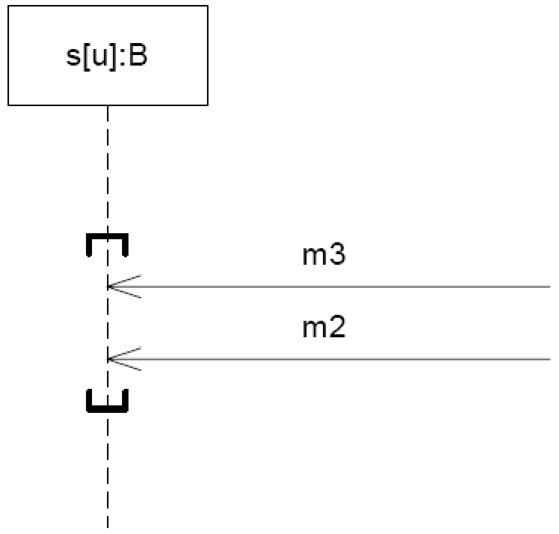


Examples of Combined Fragments (1)



Examples of Combined Fragments (2)





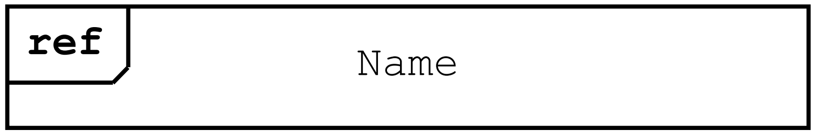
Interaction Use

→ A reference to an interaction.

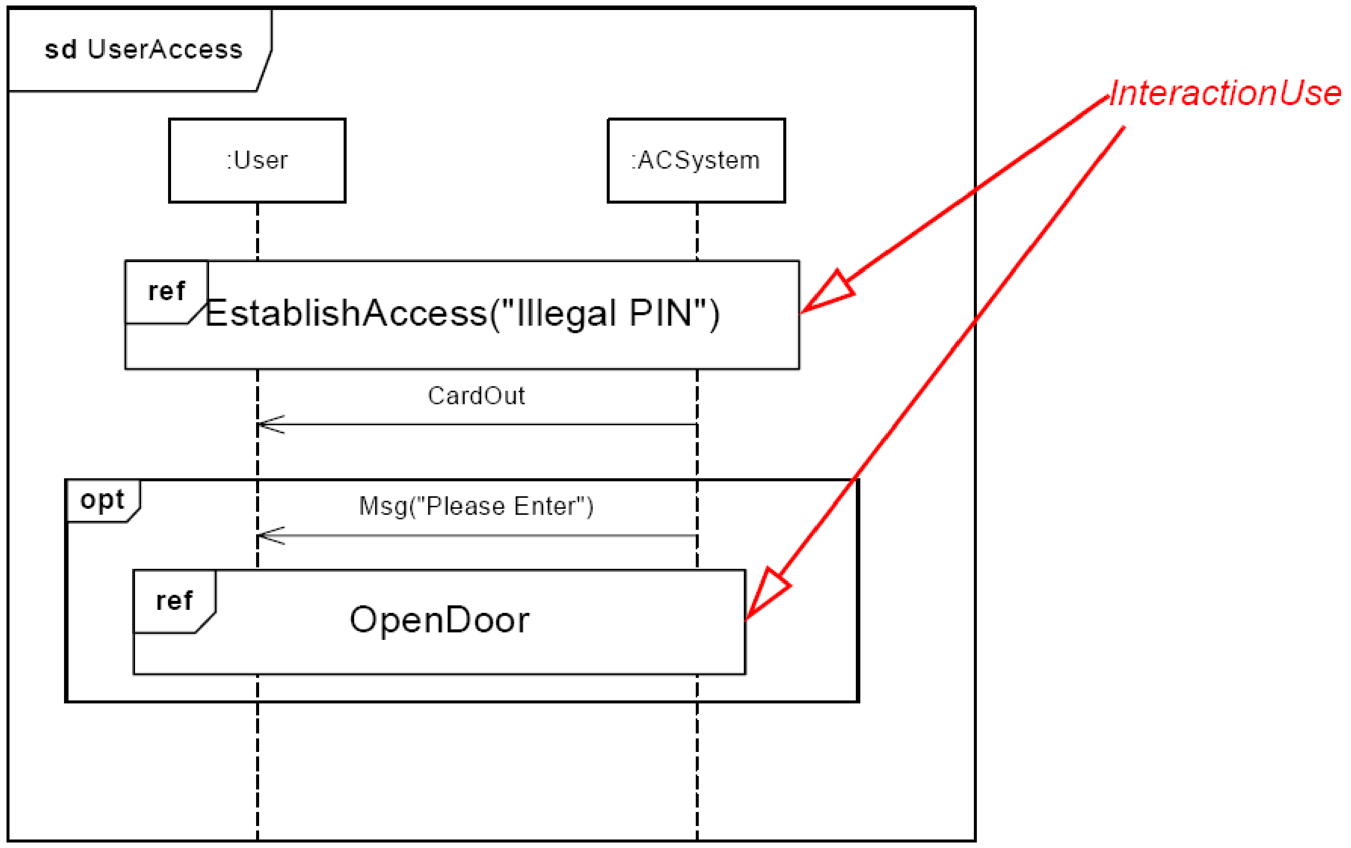
- The interaction use is a shorthand for copying the contents of the referred interaction at the place where the interaction use occurs.
- The copying must take into account substituting parameters with arguments and connect the formal gates with the actual ones.
- Sharing an interaction as a portion of several other interactions.
- Name format:

  name ::=[*attribute-name* ‘=’] [*collaboration-use* ‘.’] *interaction-name*
  [‘(‘ *io-argument* [‘,’ *io-oargument*]* ‘)’] [‘:’ *return-value*

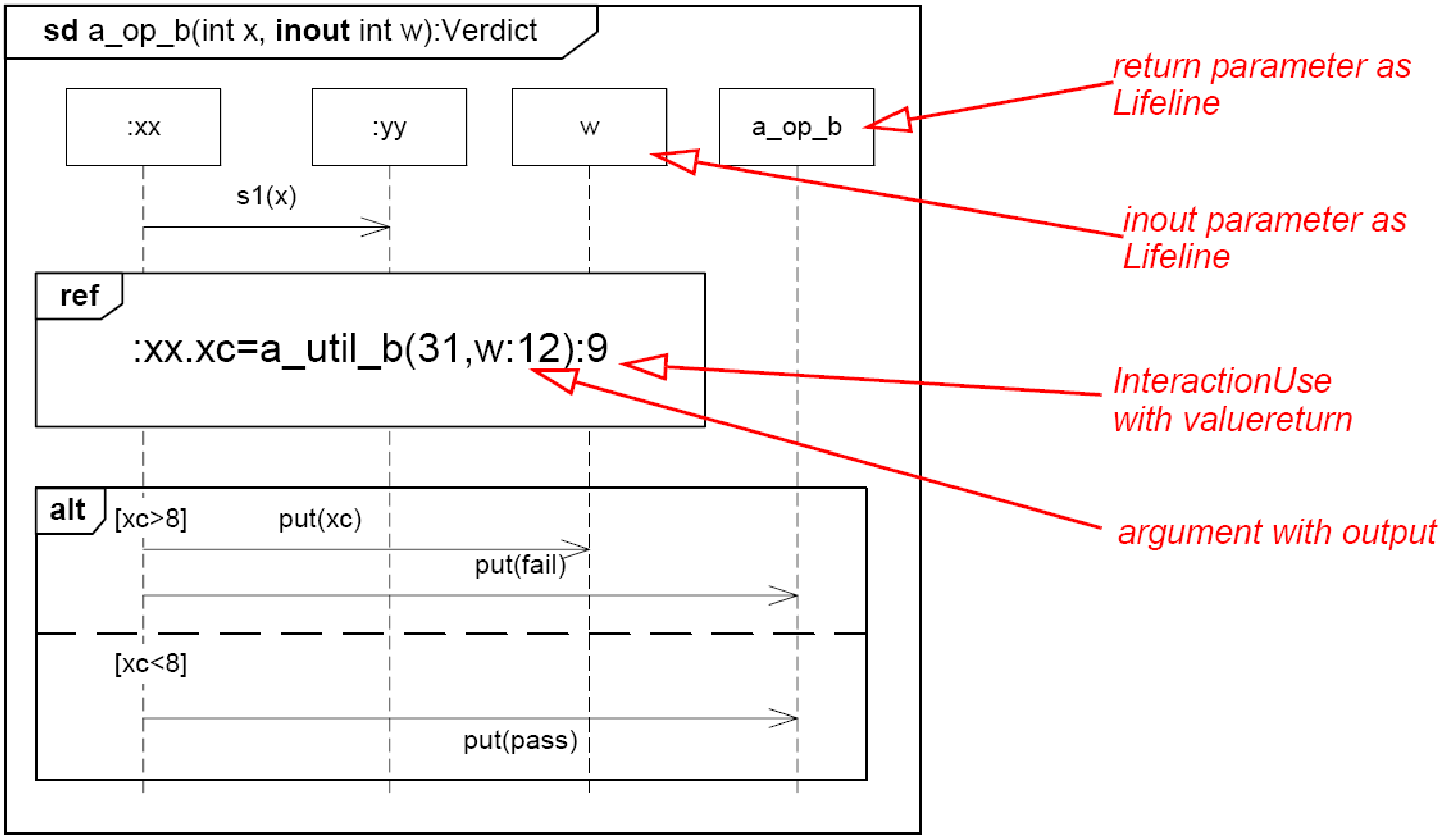
  io-argument ::= *in-argument* | ‘out’ *out-argument*]



Examples of Interaction Uses (1)



Examples of Interaction Uses (2)

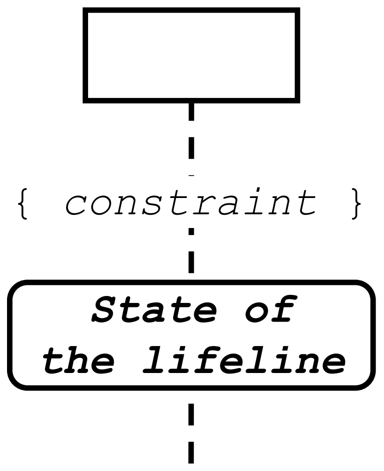


sd a_op_b(int x, inout int w):Verdict
:xx
:yy
w
a_op_b
s1(x)
ref
:xx.xc=a_util_b(31,w:12):9
alt
[xc>8]
put(xc)
put(fail)
[xc<8]
put(pass)
return parameter as Lifeline
inout parameter as Lifeline
InteractionUse with valuereturn
argument with output

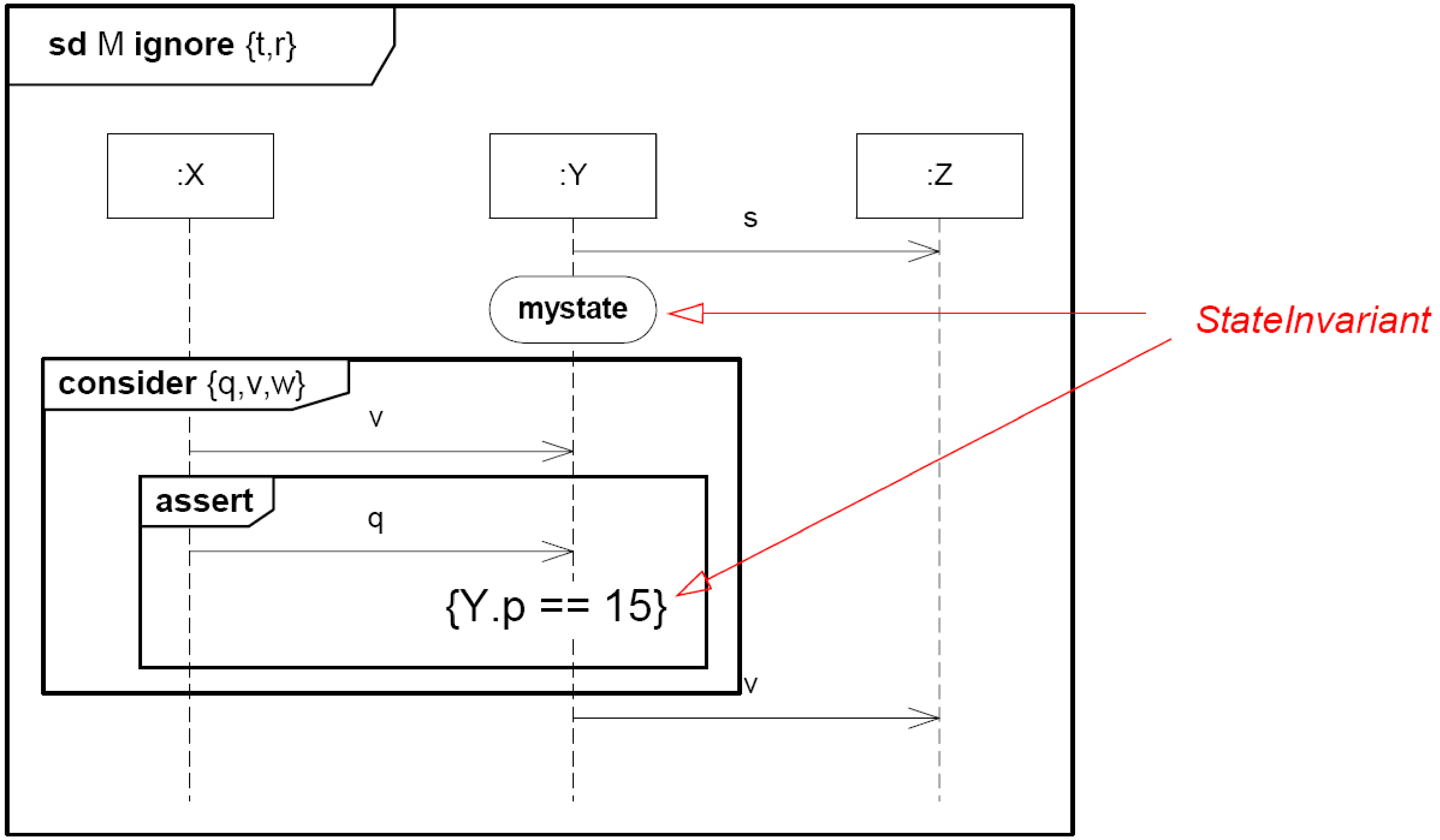
State Invariant

→ A runtime constraint on the participants of the interaction.

- It may be used to specify a variety of different kinds of constraints, such as values of attributes or variables, internal or external states, and so on.
- If the constraint is true, the trace is a valid trace; if the constraint is false, the trace is an invalid trace.



Example of State Invariant

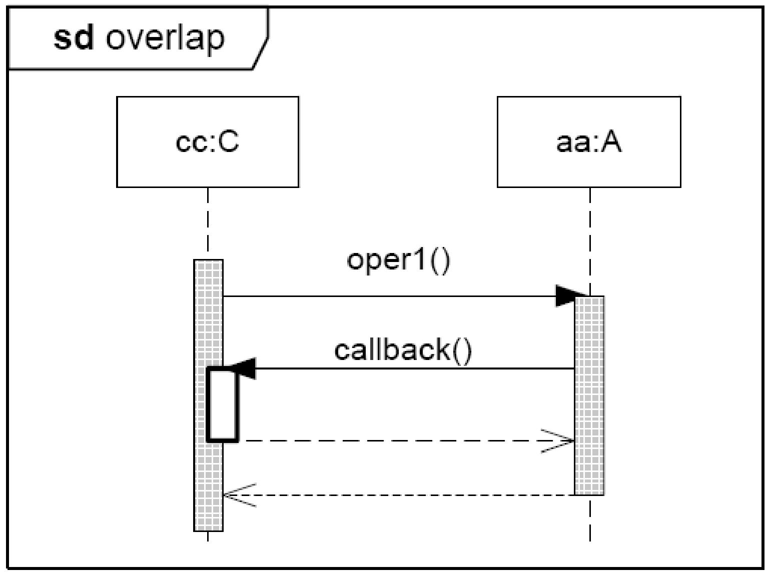


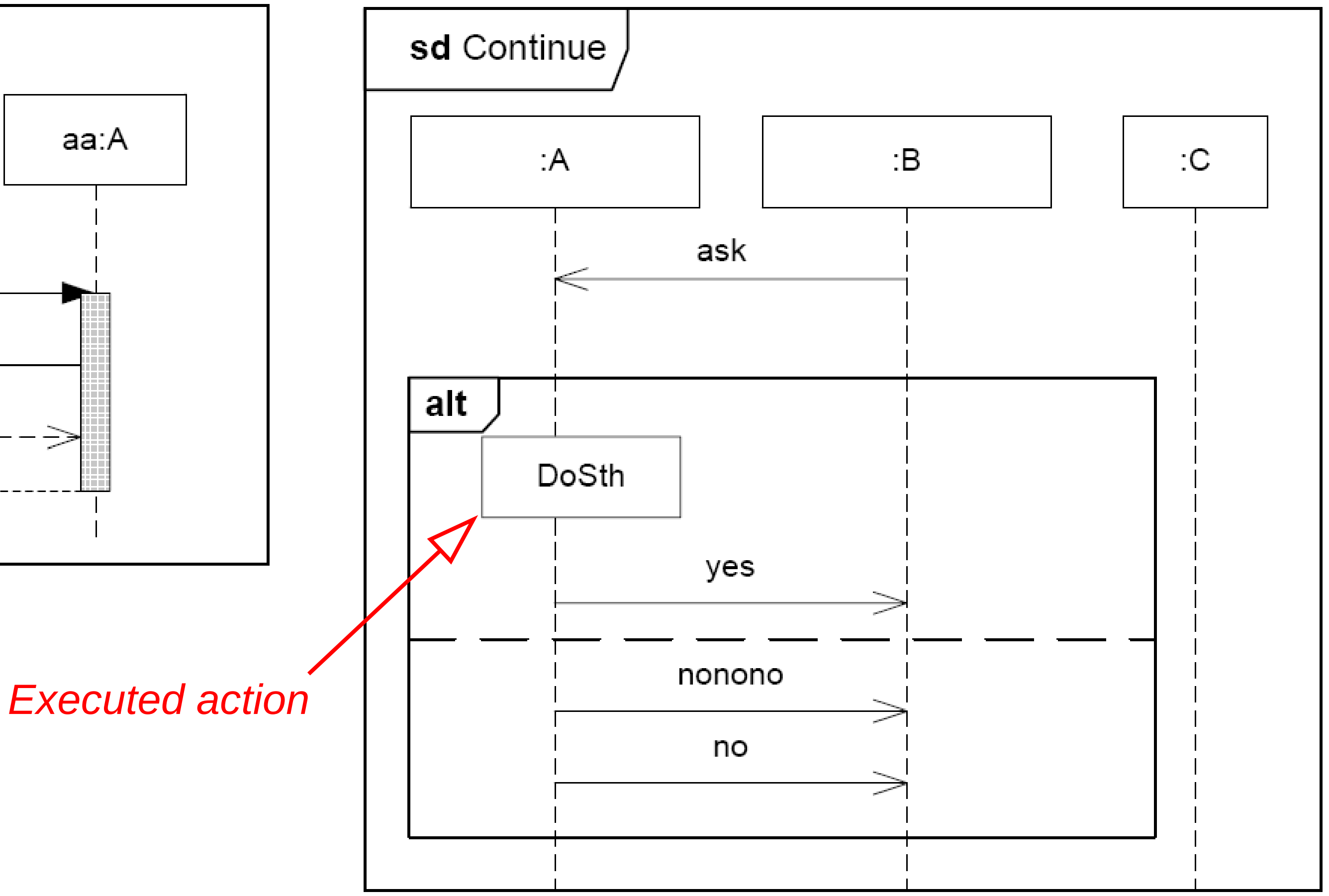
Execution Specification

→ A specification of the execution of a unit of behavior or action within the lifeline.

- The duration of an execution specification is represented by the start execution occurrence specification and the finish execution occurrence specification.
  - An ***execution occurrence specification*** represents a moment in time at which actions or behaviors start or finish.

Examples of Execution Specification



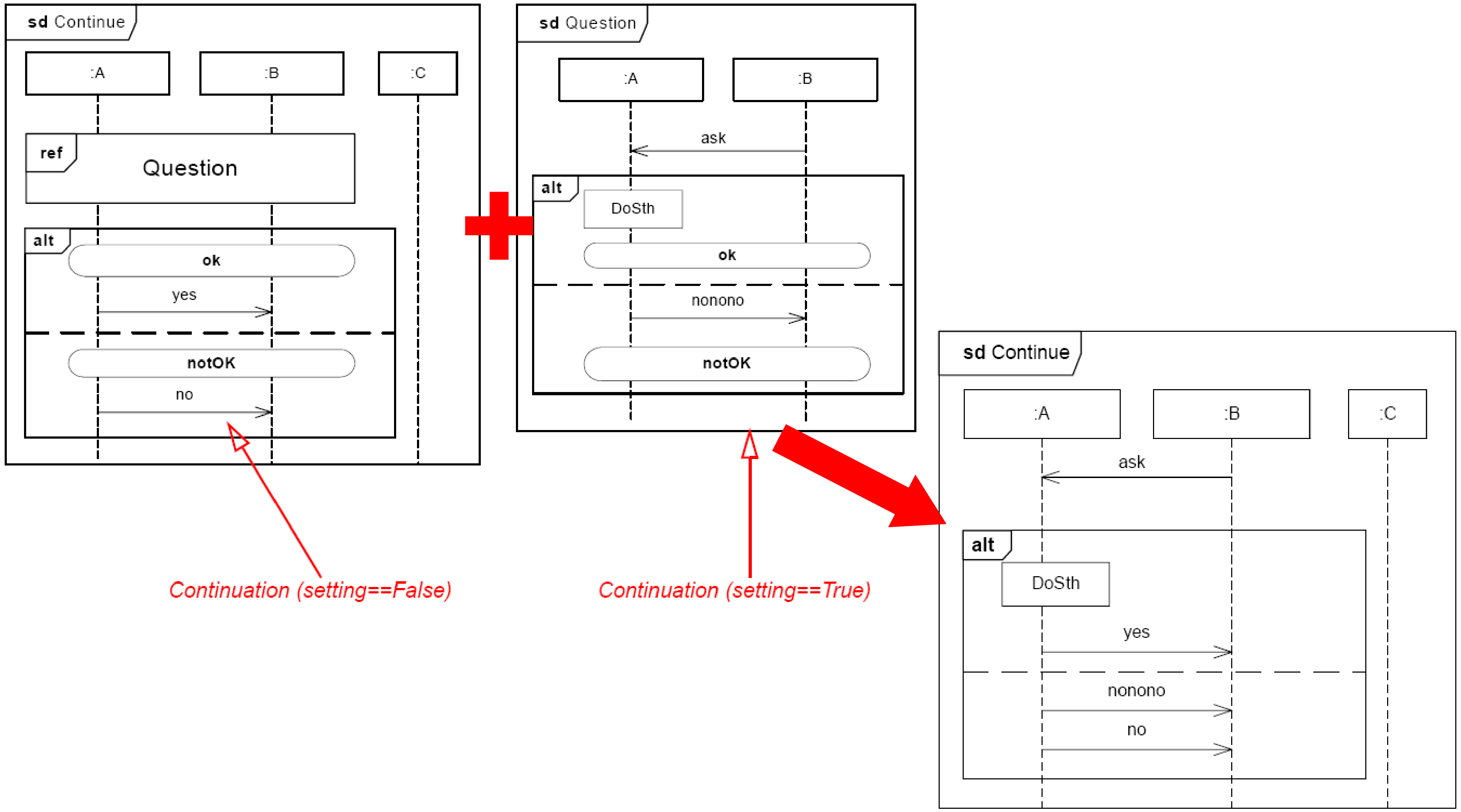


Continuations

→ A syntactic way to define continuations of different branches of an alternative or weak sequencing combined fragment.

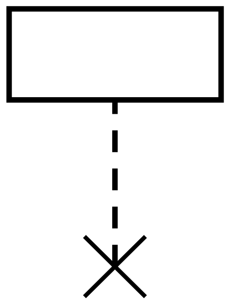
- Continuation is intuitively similar to labels representing intermediate points in a flow of control.
- If an interaction operand of an alternative combined fragment ends in a continuation with name (say) ***X***, only interaction fragments starting with the continuation ***X*** (or no continuation at all) can be appended.

Examples of Continuations



Destruction Event

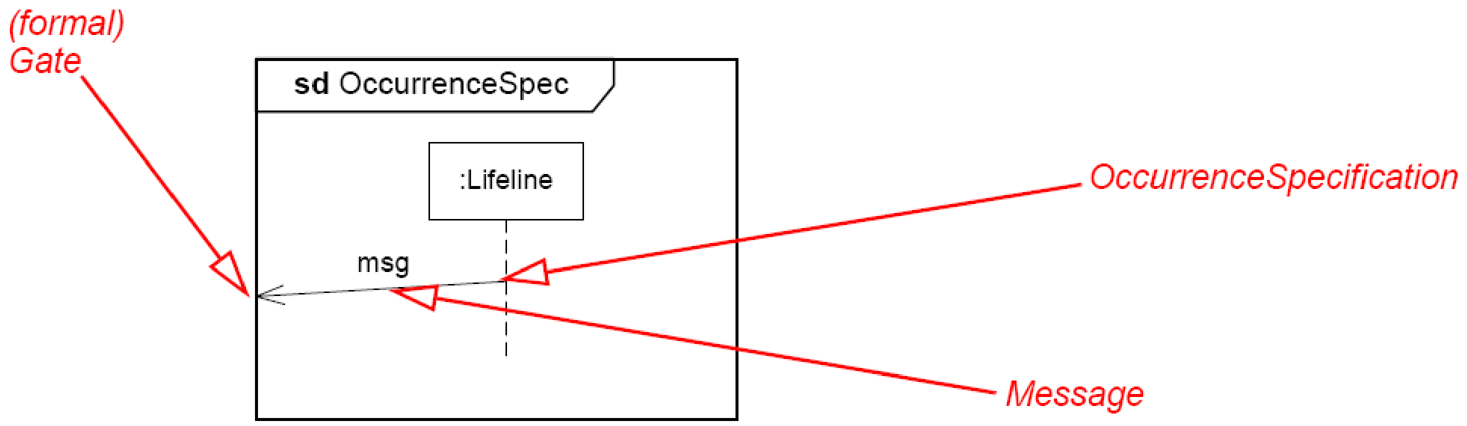
→ Represents the destruction of the instance described by the lifeline.



Gate

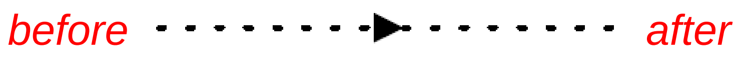
→ A connection point for relating a message outside an interaction fragment with a message inside the interaction fragment.

- Gates are connected through messages.
- Gates may be identified either by name (if specified), or by a constructed identifier formed by concatenating the direction of the message and the message name (e.g., “out_CardOut”).
- Different roles:
  - Formal gates on interactions.
  - Actual gates on interaction uses.
  - Expression gates on combined fragments.



General Ordering

- → Represents a binary relation between two occurrence specifications, to describe that one occurrence specification must occur before the other in a valid trace.
- ■ This mechanism provides the ability to define partial orders of occurrence specifications that may otherwise not have a specified order.



Time Observation and Time Constraint

Time Observation

→ A reference to a time instant during an execution.

- It points out the element in the model to observe and whether the observation is when this model element is entered or when it is exited.
- They are usually named.

Time Constraint

→ Defines a constraint that refers to a time interval.

- A ***time interval*** defines the range between two time expressions; in interactions they usually refer to the time observations.

Duration, Observation and Duration Constraint

Duration

→ Defines a value specification that specifies the duration in time, i.e., temporal distance between two time instants–starting point in time and ending point in time.

Duration Observation

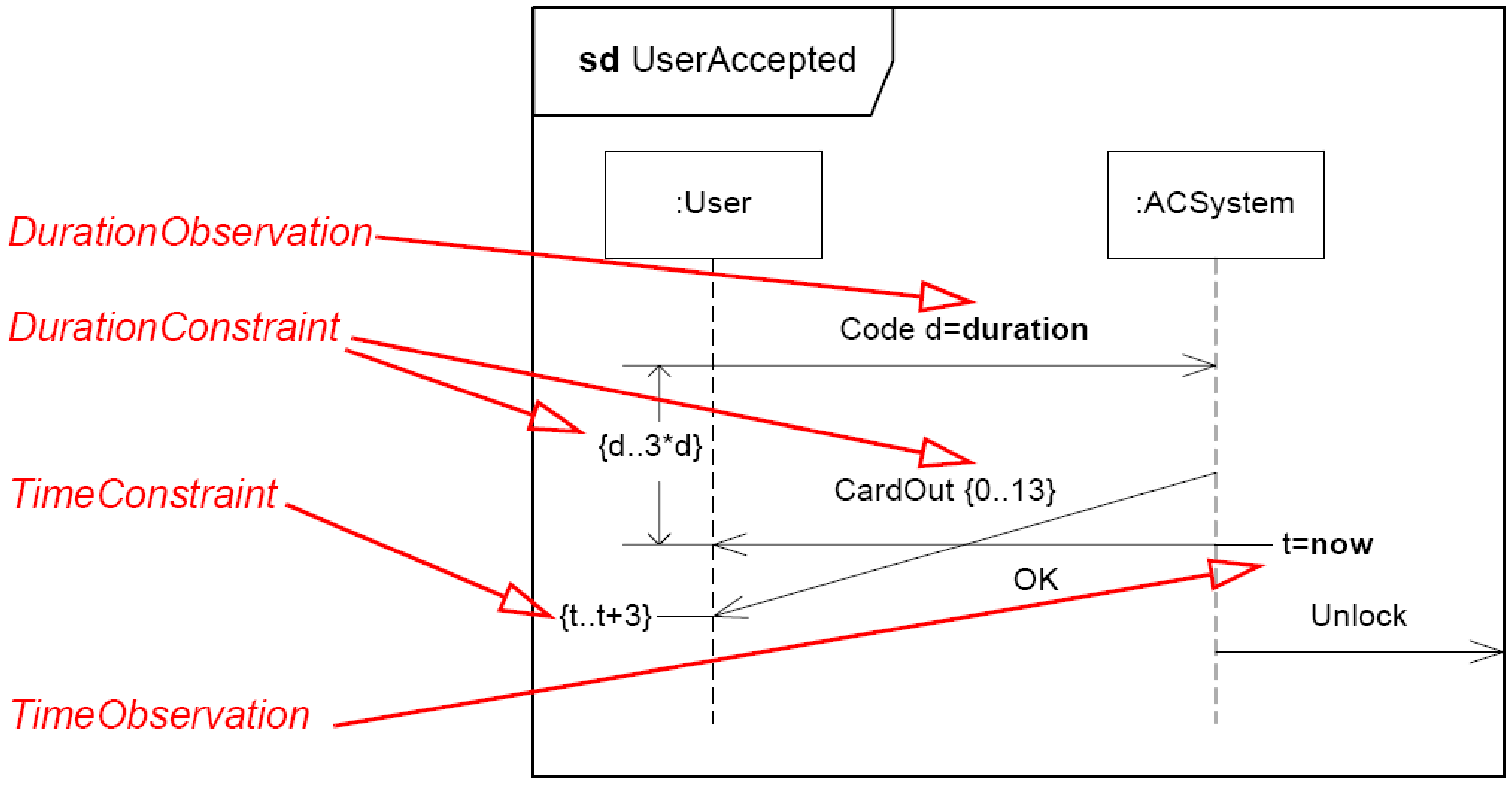
→ A reference to a duration during an execution.

Duration Constraint

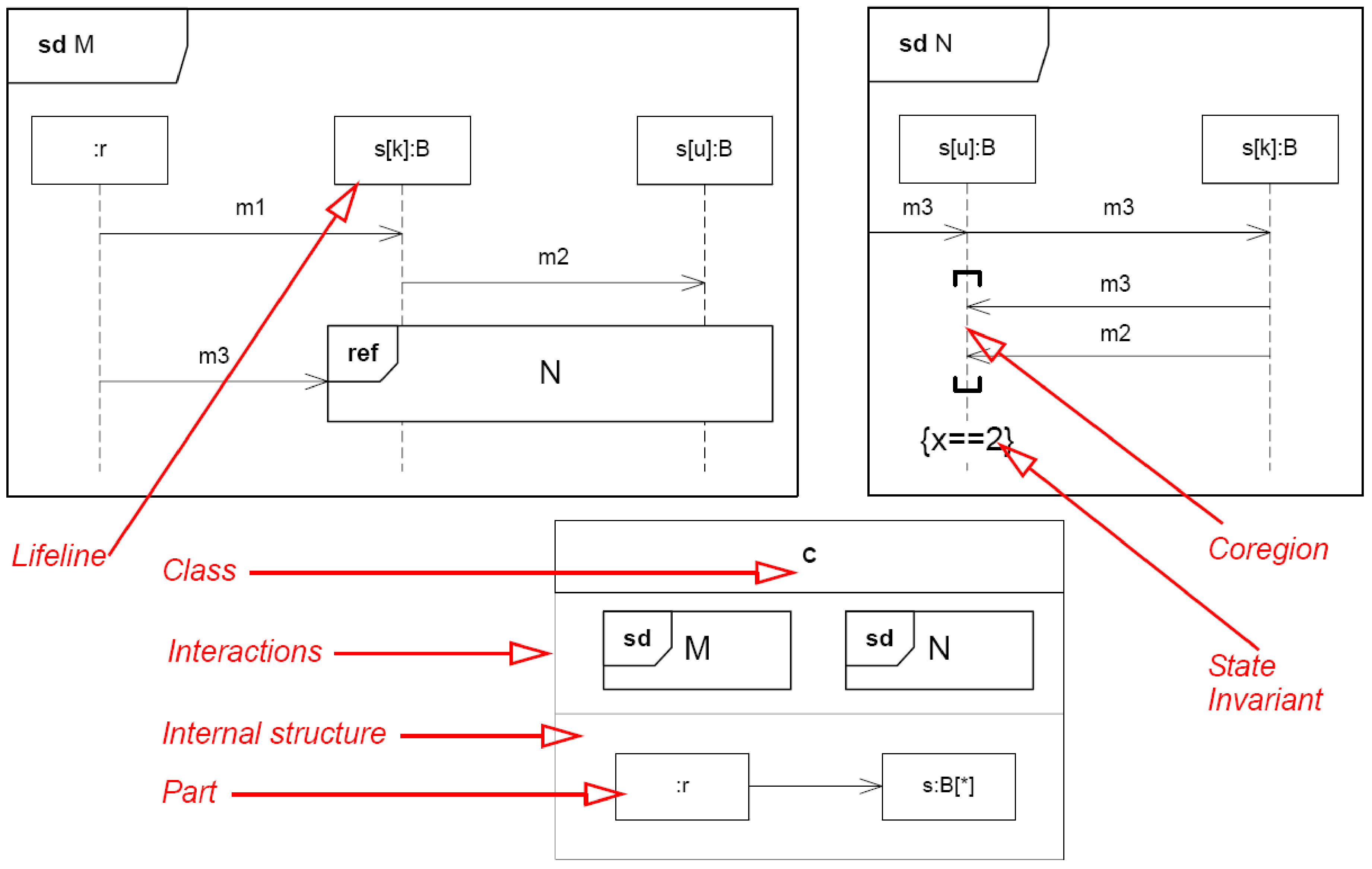
→ Defines a constraint that refers to a duration interval.

- A ***duration interval*** defines the range between two durations.

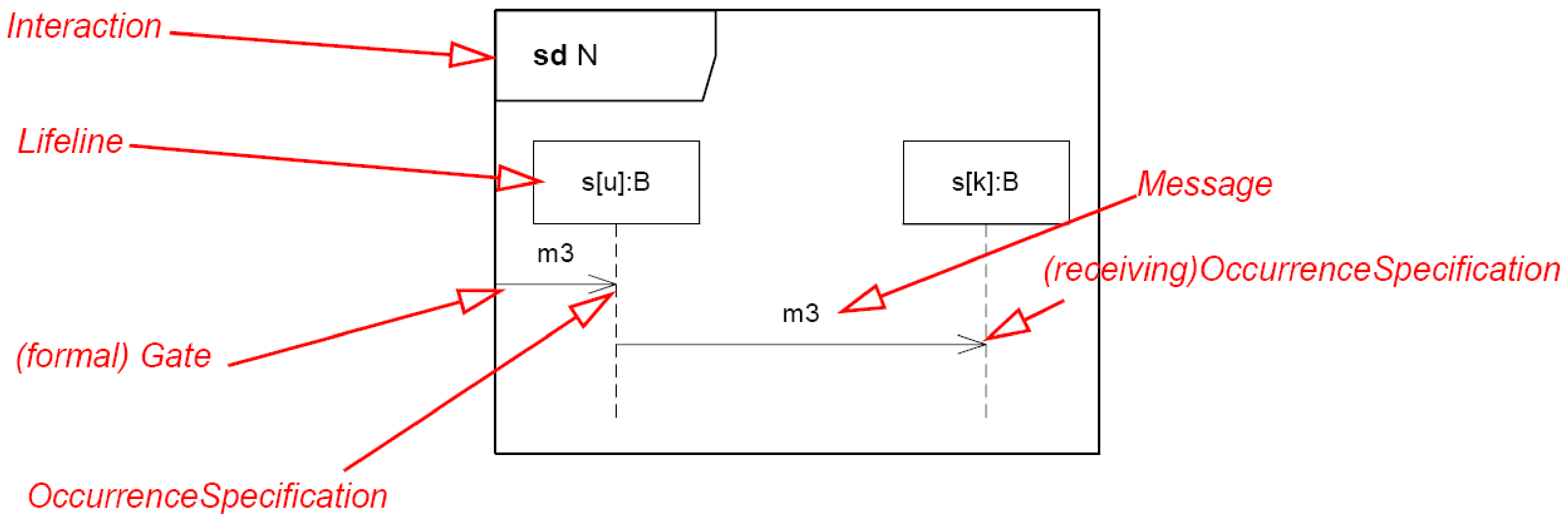
Example of Timing Concepts



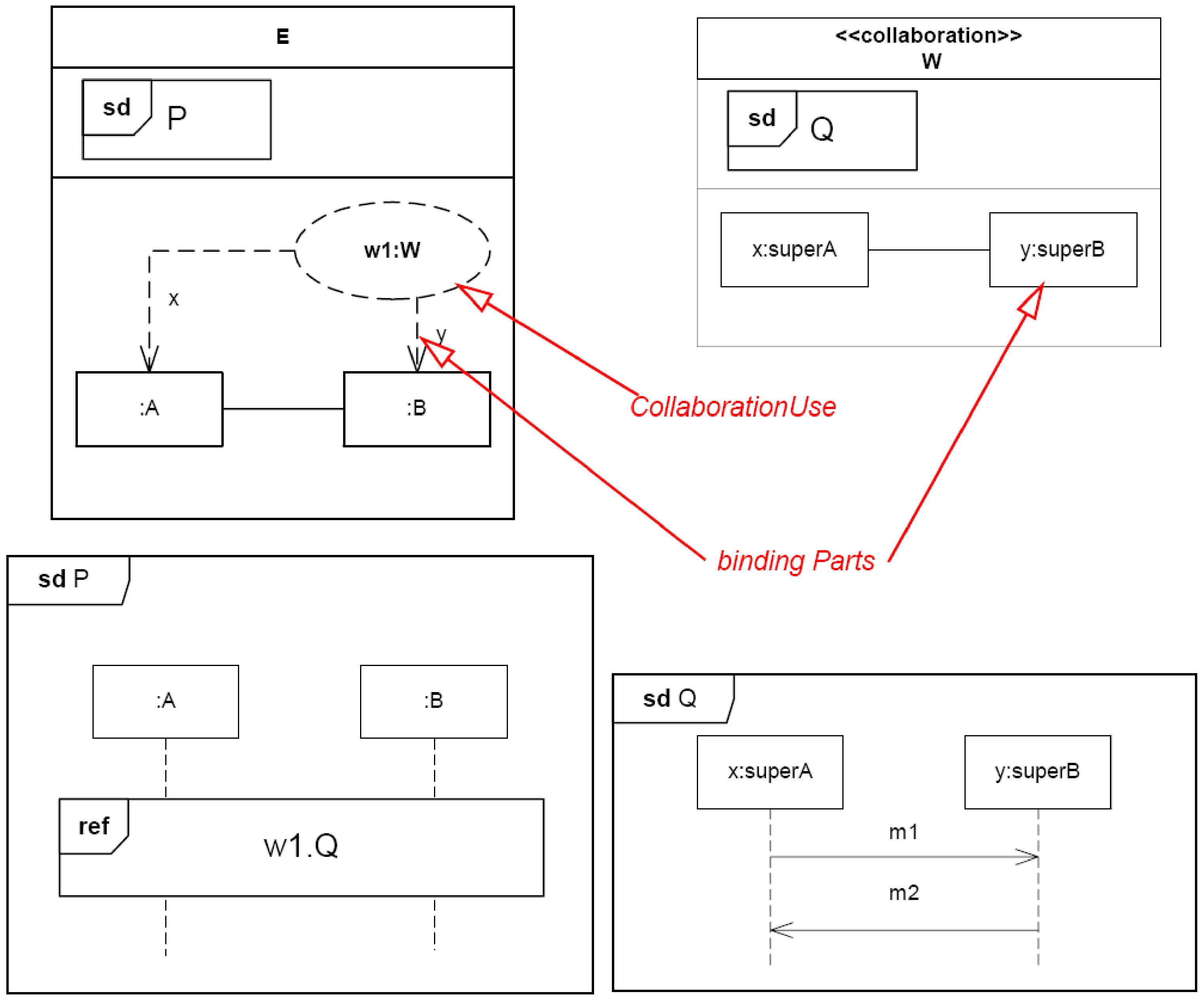
Additional Examples of Sequence Diagrams (1)



Additional Examples of Sequence Diagrams (2)



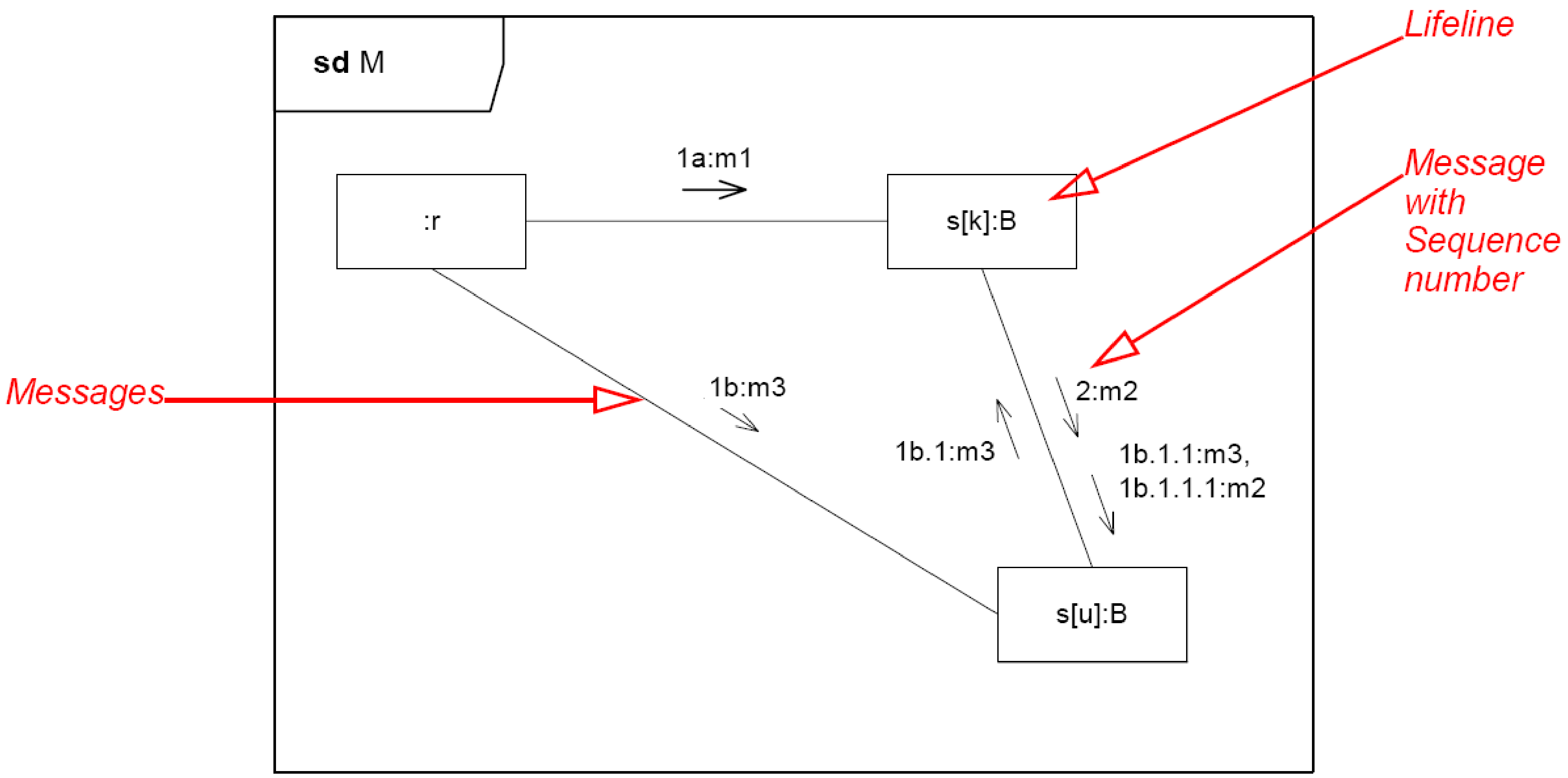
Additional Examples of Sequence Diagrams (3)



Communication Diagram

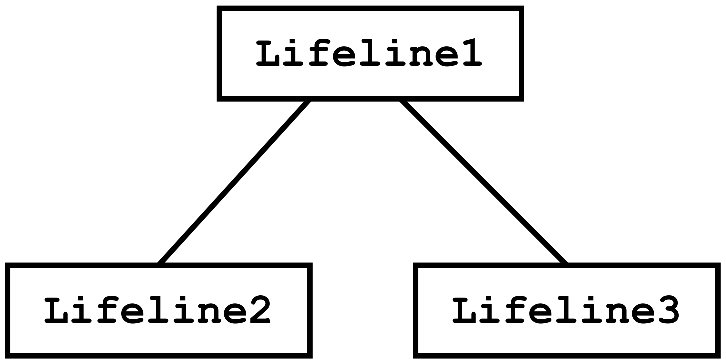
→ An interaction diagram which focuses on the interaction between lifelines where the architecture of the internal structure and how this corresponds with the message passing is central.

❑ Semantically corresponds to a simple sequence diagram.



Lifeline

- Semantically identical to the lifeline from sequence diagrams.
- The format of the lifeline name (“lifelineident”) identical to the lifeline from sequence diagrams.
- Communicating lifelines are linked by connectors.



Message

- Semantically identical to the message from sequence diagrams.
- Arrow determines the communication direction.
- The message name is given by the following format:

  message-name ::= *sequence-expression* ‘:’ *messageident*

  sequence-expression ::= *sequence-term* [*sequence-term*]*

  sequence-term ::= [‘.’ *integer* | *name*] [*recurrence*]

  recurrence ::= ‘*’ [‘| | ’] ‘[’ *iteration-clause* ‘]’ | ‘[’ *guard* ‘]’

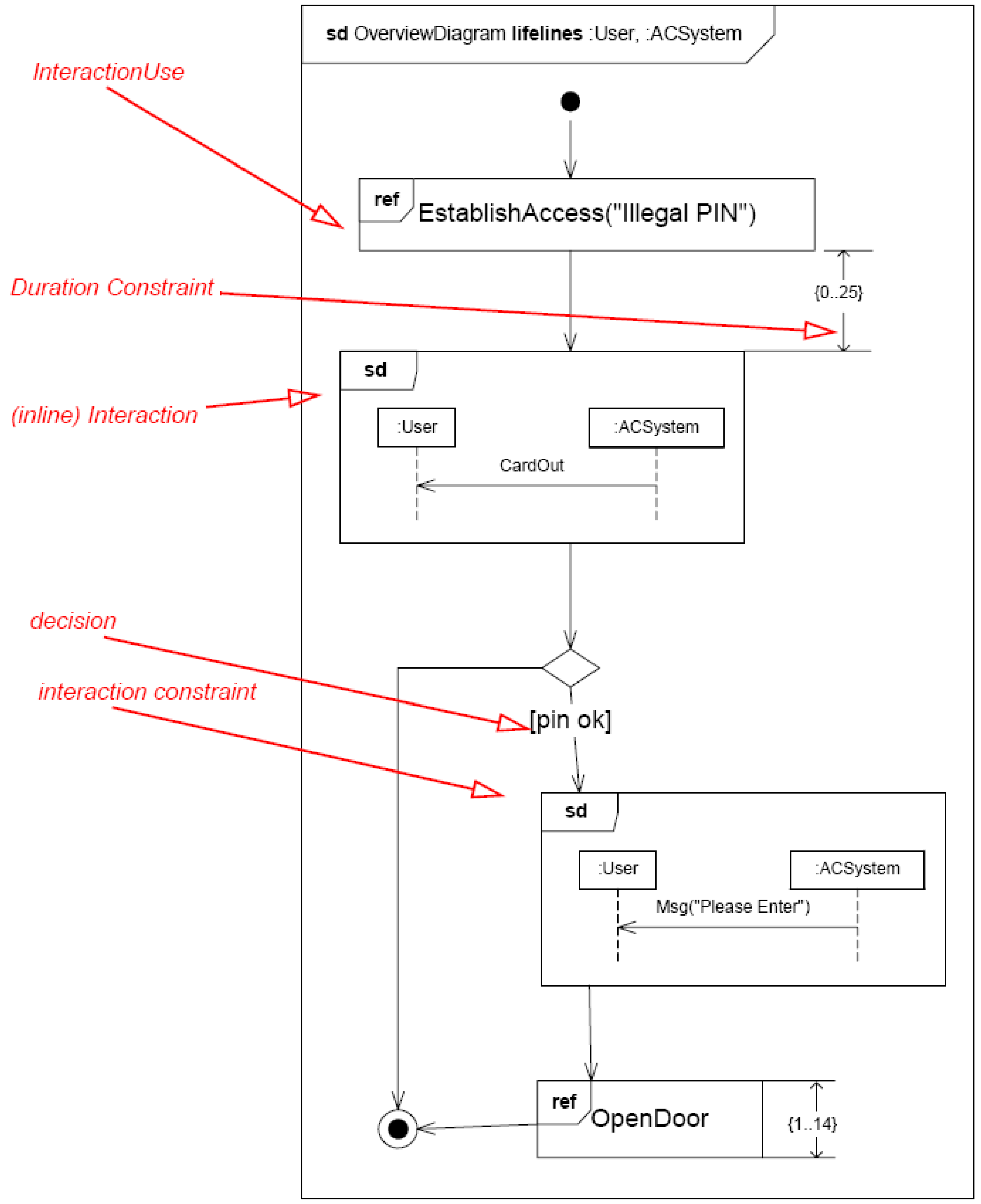
  - The *integer* represents the sequential order of the message within the next higher level of procedural calling.

    3.1.3 before 3.1.4 in 3.1
  - The *name* represents a concurrent thread of control.

    3.1a is concurrent with 3.1b within activation 3.1
  - The *recurrence* represents conditional or iterative execution.

    1.2*[i := 1..n]   3.5a[x > y]   4.3*| | [1..3]

Interaction Overview Diagram



→ An interaction diagram which defines interactions through a variant of activity diagrams in a way that promotes overview of the control flow.

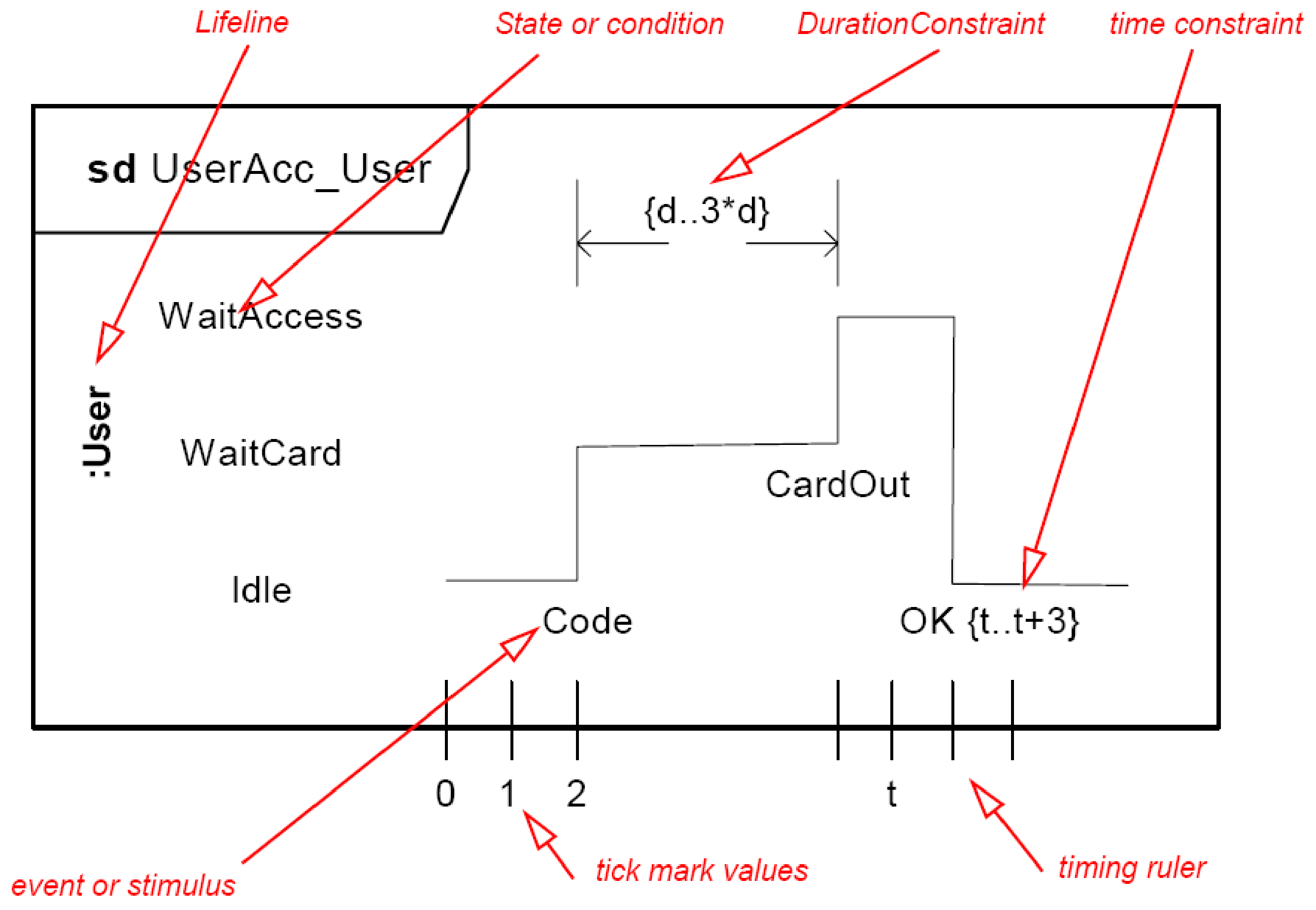
- Interaction overview diagrams focus on the overview of the flow of control where the nodes are interactions or interaction uses.
- The lifelines and the messages do not appear at this overview level.

Differences from Activity Diagrams

1. In place of object nodes of activity diagrams, interaction overview diagrams can only have either (inline) interactions or interaction uses. Inline interaction diagrams and interaction uses are considered special forms of call behavior action.
2. Alternative combined fragments are represented by a decision node and a corresponding merge node.
3. Parallel combined fragments are represented by a fork node and a corresponding join node.
4. Loop combined fragments are represented by simple cycles.
5. Branching and joining of branches must in interaction overview diagrams be properly nested. This is more restrictive than in activity diagrams.
6. Interaction overview diagrams are framed by the same kind of frame that encloses other forms of interaction diagrams. The heading text may also include a list of the contained lifelines (that do not appear graphically).

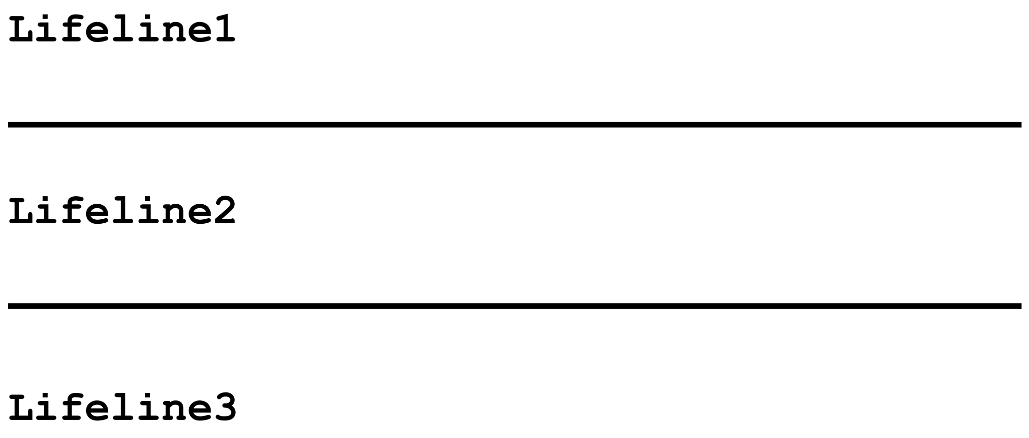
Timing Diagram

→ An interaction diagram which is used to show interactions when a primary purpose of the diagram is to ***reason about time***, focusing attention on time of occurrence of events causing changes in the modeled conditions of the lifelines.



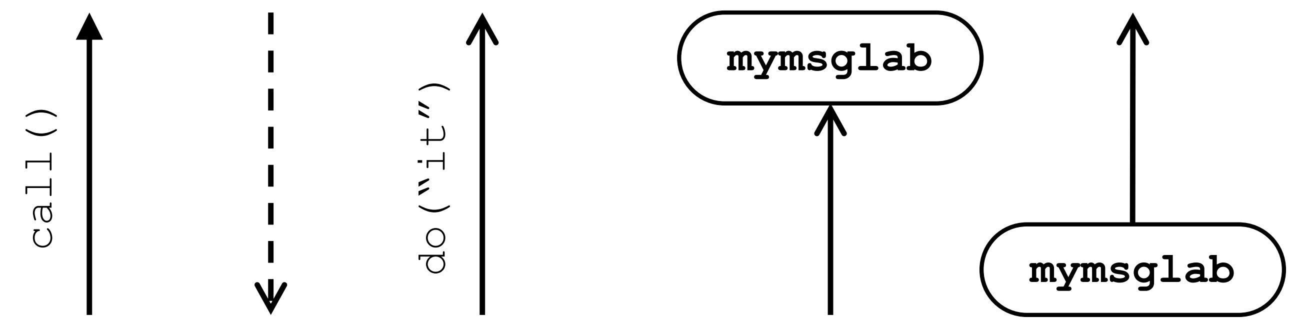
Lifeline

- Semantically identical to the lifeline from sequence diagrams.
- The format of the lifeline name (“lifelineident”) is identical to the lifeline from sequence diagrams.



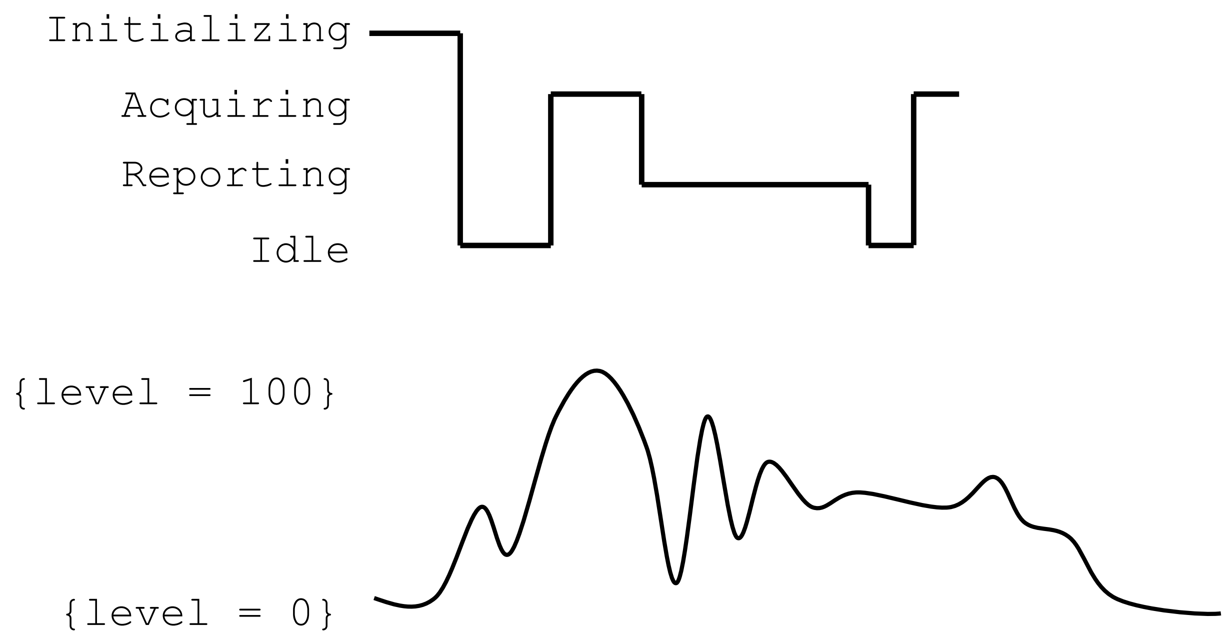
Message

- Semantically identical to the message from sequence diagrams.
- The format of the message name (“messageident”) is identical to the message from sequence diagrams.
- The *message labels* can be used to denote that a message may be disrupted by introducing labels with the same name.
  - Message labels are the notational shorthands used to prevent cluttering of the diagrams with a number of messages crisscrossing the diagram between lifelines that are far apart.



State or Condition Timeline

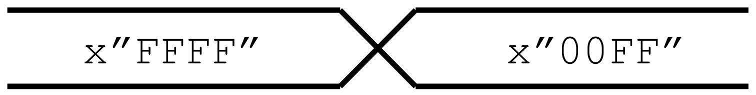
- → Representation of changing the state of the classifier or attribute, or some testable condition in time.
- ■ It is also permissible to let the state-dimension be continuous as well as discrete. This is illustrative for scenarios where certain entities undergo continuous state changes, such as temperature or density.



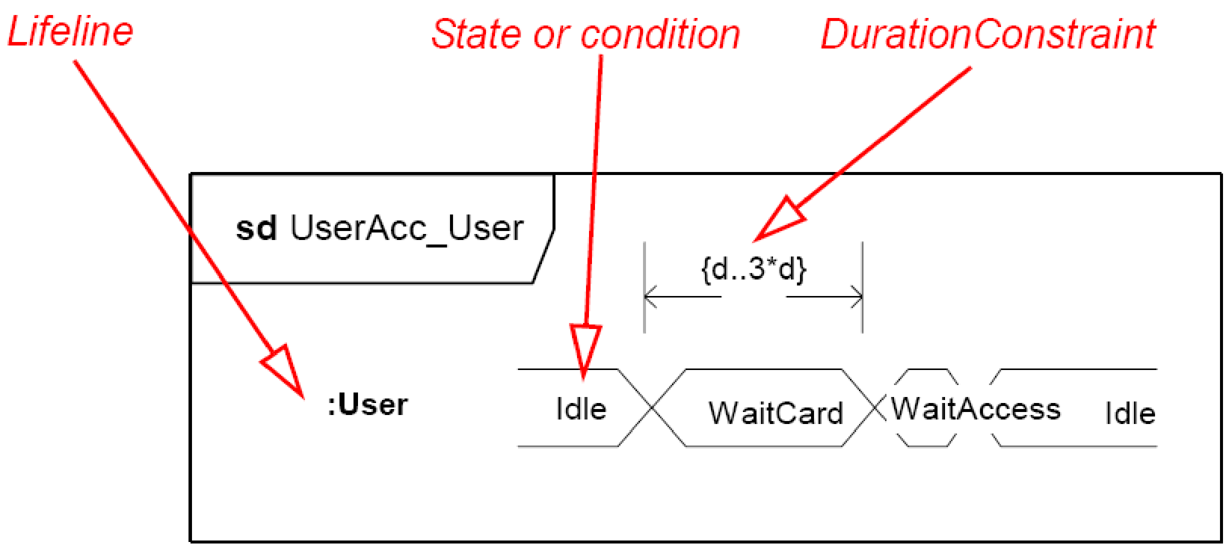
General Value Lifeline

→ Shows the value of the connectable element as a function of time.

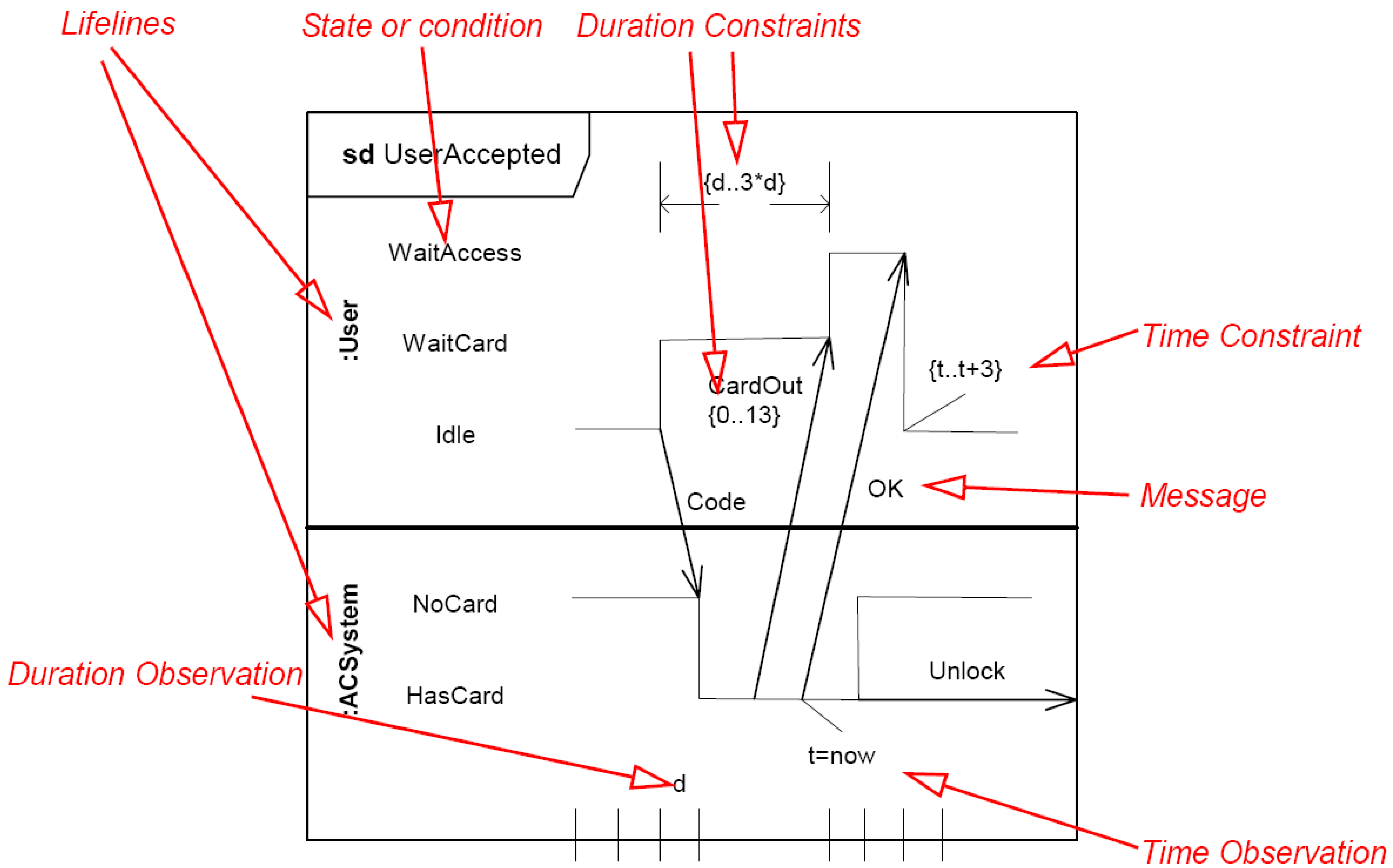
- Value is explicitly denoted as text.
- Crossing reflects the event where the value changed.



Additional Examples of Timing Diagrams (1)



Additional Examples of Timing Diagrams (2)


Lifelines
State or condition
Duration Constraints
sd UserAccepted
{d..3*d}
WaitAccess
:User
WaitCard
Idle
CardOut
{0..13}
{t..t+3}
Time Constraint
Code
OK
Message
:ACSystem
NoCard
HasCard
Unlock
Duration Observation
d
t=now
Time Observation