

Operation & Interaction Modeling

Unified Modeling Language (UML) — Operation & Interaction Modeling

Goals of Chapter

Understand elements of UML Interaction Diagrams & their meaning

Understand object messaging

Understand how Interaction Diagrams are derived from use-cases

Outline

Operation & Interaction Concepts

Models & Diagrams

Notations & Guidelines

Summary

What is an Operation?

An Operation is:

- A process of a practical or mechanical nature in some form of work or production.

UML Definition: (*Glossary*, p. B-13)

- A service that can be requested from object to effect behavior
- Has signature, which may restrict possible actual parameters

In programming languages:

- Message Pattern (Smalltalk)
- Member–Function Declaration (C++)
- Method (JAVA)
- Procedure, Function Declaration (most programming languages)
- Entry Specification (Ada)

What is a Method?

An Method is:

- Manner or mode of procedure
- Way of doing something.

UML Definition: (*Glossary*, p. B-13)

- Implementation of an operation
- Specifies the algorithm or procedure that effects the result of the operation

In programming languages:

- Member–Function Definition (C++)
- Method (JAVA, Smalltalk)
- Procedure, Function Definition (most programming languages)
- Accept Statement / Entry Body (Ada)

What is a Message?

Is:

- Communication containing some information, advice, request, or the like.

UML Definition:

- Particular communication between instances that is specified in an interaction
 - Can be:
 - ◆ raising a signal (or exception)
 - ◆ invoking an operation
 - ◆ creating/destroying on instance

In programming languages:

- Message (Most Object–Oriented Programming Languages)
- Member–Function Call (C++)
- Procedure, Function Call (most programming languages)
- Entry Call (Ada)

Interaction

Is

- Behavior specification consisting of
 - Set of objects/roles in collaboration
 - Stimuli/messages exchanged to achieve specific purpose
 - e.g. implementation of an operation
- Used to describe
 - Operation or use-case
 - ❖ Said to realize operation/use-case

Outline

Operation & Interaction Concepts

Models & Diagrams

Notations & Guidelines

Summary

Interaction Diagram

Is

- Static picture of potential communications
- Pattern of messages exchanged among objects to accomplish specific purpose
 - A *collaboration* with messages
 - ◆ Each set of messages applied to a collaboration results another interaction

2 forms

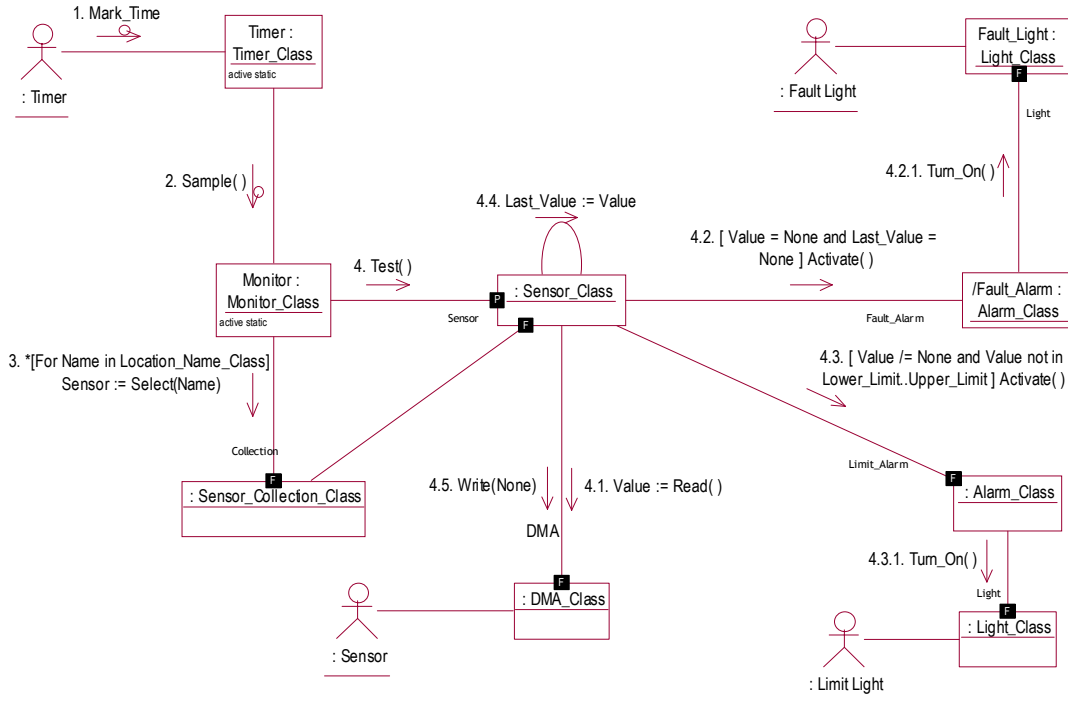
- *Collaboration Diagram*
- *Sequence Diagram*

Collaboration Diagrams

Shows

- Interaction organized around
 - Participating Objects/Roles
 - Links among objects/roles
 - ❖ Not shown in Sequence Diagram
- Sequence & concurrent threads of messages
 - Must be determined from sequence numbers
 - More obvious in Sequence Diagram
 - ◆ Time is separate dimension in Sequence Diagrams

Collaboration Diagram Example



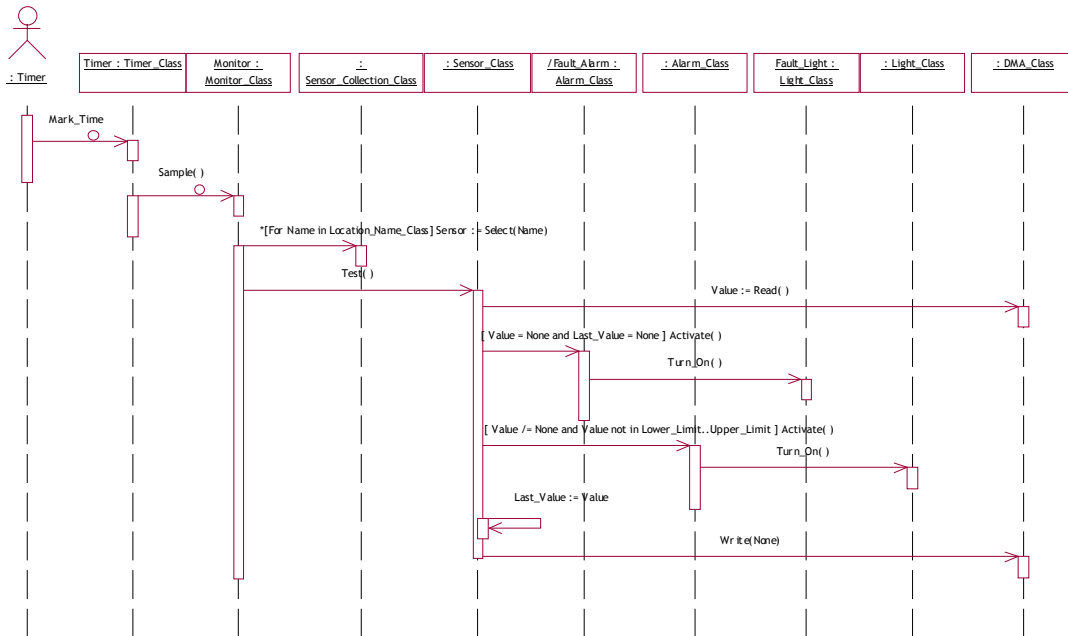
Sequence Diagrams

Shows

- Instances participating in interaction by their "*lifelines*"
- Stimuli they exchange arranged in time sequence

Does NOT show association among objects

Interaction Diagram
Sequence Diagrams Example



Comparison of Sequence & Collaboration Diagram

Similar information / different views

- Sequence Diagrams
 - Show explicit sequence of messages
 - Better for
 - ◆ Real-time specifications
 - ◆ Complex scenarios

- Collaboration diagrams
 - Show relationships among objects
 - Better for
 - ◆ Understanding all effects on given instance
 - ◆ Procedural design

Forms of Interaction Diagrams

Generic form

- Shows all possible sequences
- Useful for describing execution control & concurrency

Instance form

- Shows 1 actual sequence
 - Must be consistent with generic form
- Useful for highlighting message flow through set of objects under specific conditions

If *Generic* form doesn't have any loops or branches, forms are isomorphic

Outline

Operation & Interaction Concepts

Models & Diagrams

Notations & Guidelines

Summary

Concepts & Notation

Collaboration Diagrams

Sequence Diagrams

Messages

Objects interact through their links with other objects

- An object is
 - *Client*
 - ◆ Object that invokes the operation
 - *Supplier/Server*
 - ◆ Object that provided the operation

- Client & Server can be same object
 - i.e.** self-link

Messages (cont.)

Messages are sent across links

- May represent
 - An operation call
 - ◆ Most common message
 - Signal between active threads
 - Raising of events
- Messages can't flow backward over 1-way link

Representation

- An arrow placed next to link
 - Indicates
 - ◆ Direction of message
 - ◆ Control Mechanism
- Label

Message (cont.)

Control Mechanisms

Describes how 2 objects communicate

- Does client wait for acceptance of request?
- Does client wait for end of communication?

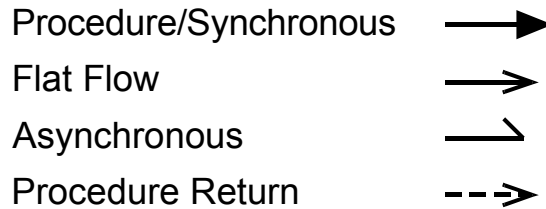
UML mentions 5 request mechanisms

- Procedure Call or other nested flow of control
 - e.g. Synchronous**
- Flat flow of control
- Asynchronous
- Timed
- Balking

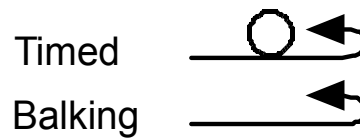
Representation

Arrows on links

- UML defines 3-styles of arrows for



- Others are considered to be extensions of UML
 - Common Extensions



Label

Contains:

- Name of Message
- List of arguments (optional)
 - Actual parameters supplied during call
- Return Value Container (optional)
 - For Functional operations
- Relative position in Interaction sequence
- Conditions for message to be sent (optional)
- Iteration (optional)

Syntax

*Predecessor Guard-condition Sequence-expression
Signature*

Signature

Syntax

return-value := message-name argument-list

- Where:

- *Message-Name* is name of

- ◆ Operation
- ◆ Signal
- ◆ Event raised

- *Argument-list*

- ◆ Comma-separated list of actual parameters
- ◆ Enclosed in parentheses

Signature (cont.)

- Where (cont.):

- *Return-Value*¹

- ◆ List of names
 - ❖ Specifies objects where values returned by functional operation should be placed
 - ❖ Can be used as arguments to subsequent messages

Notes:

- If operation is not functional don't include
 - ◆ *Return-value*
 - ◆ " := "

¹ Should be called "receiving-objects"

Sequence Expression

Dot-separated list of sequence-terms followed by a colon

i.e. *Term.Term.Term :*

Each term has form:

`[integer / name] [recurrence]`

- Where:
 - *Integer* indicates sequential order of message within next higher level of nesting
 - e.g.** 3.1.4 follows 3.1.3 within 3.1
 - *Name* indicates concurrent thread of control
 - ◆ Messages that differ in final name are concurrent at that level of nesting
 - e.g.** 3.1a and 3.1b are concurrent within activation 3.1

Sequence Expression (cont.)

Recurrence Expression has 3 forms

- Branch
 - `'[condition-clause]'`
- Sequential Iteration
 - `'*' '[' iteration-clause]'`
- Concurrent Iteration
 - `'*|/' '[' iteration-clause]'`

Clauses are described either in

- Pseudo-code
- Natural Language
- Programming language

Examples

`*[J := 1..n] *|| [For each agent in collection]`
`[x>y]`

Predecessor

Comma-separated list of sequence-numbers followed by a slash

sequence-number ' , ' ... '/'

- Where:
 - Sequence-numbers are sequence-expressions without recurrence

Examples

1.2 /

1.4A, B.1 /

Note:

- Mainly used for synchronizing concurrent threads
- By default message
 - 1.3 is preceded by either
 - ◆ 1.2.*n* if exists
 - ◆ 1.2 otherwise

Guard-Condition

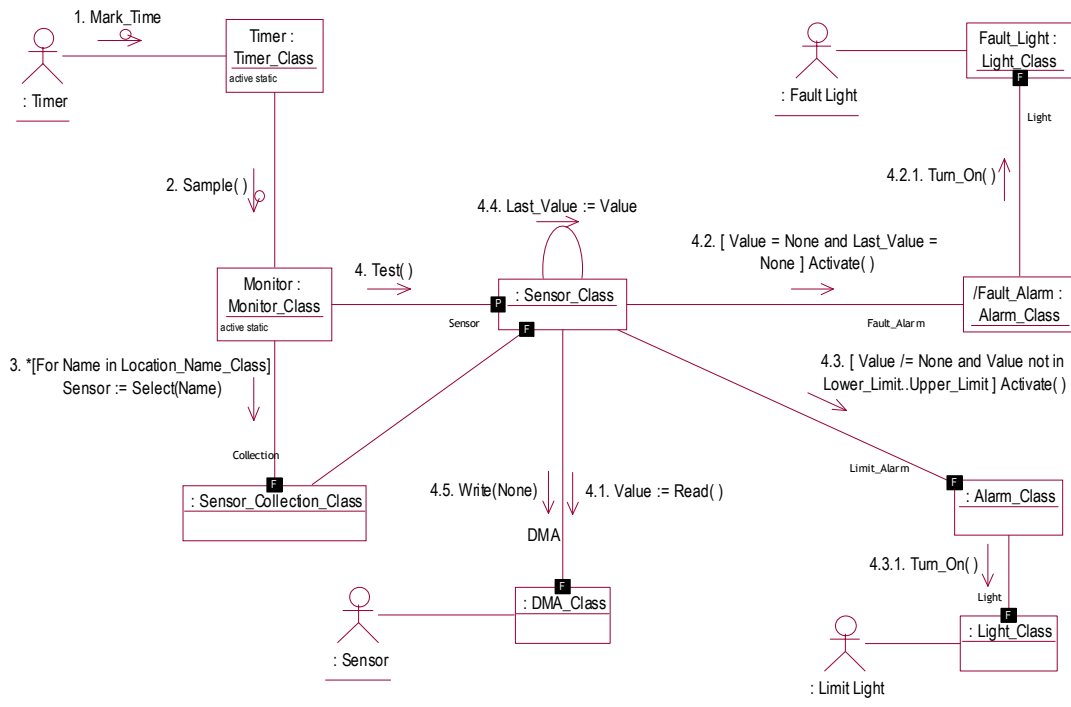
Specifies additional conditions that must be satisfied when synchronizing concurrent threads

Same syntax as in Statecharts

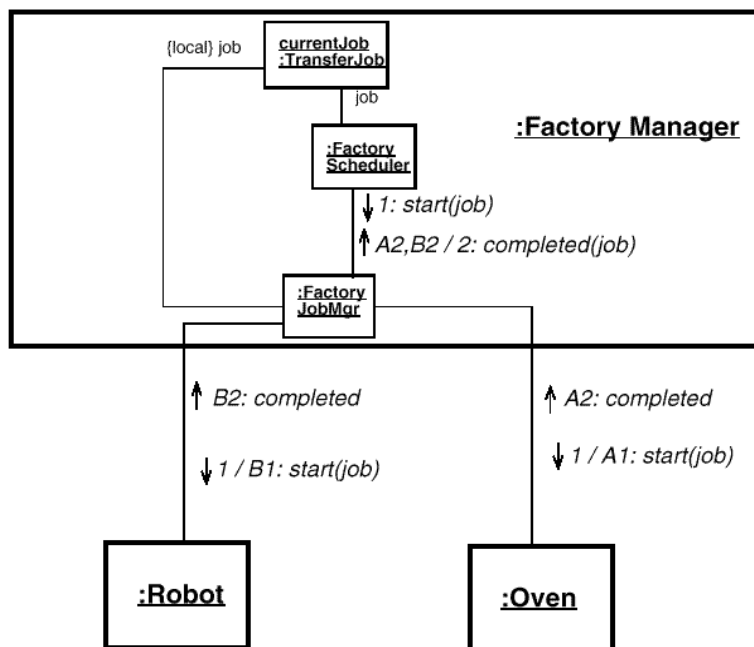
[Boolean expression]

- where
 - In state of
 - ◆ parameters of triggering event
 - ◆ state of attributes
 - ◆ links
 - Involve test of optionally fully qualified concurrent states
 - ◆ in
 - ◆ not in

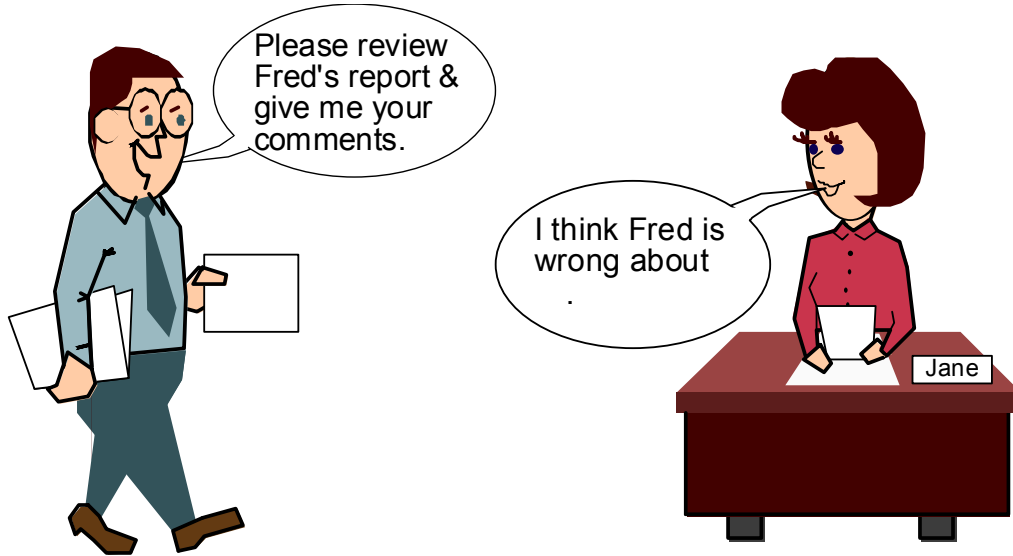
Simple Messages Example



Concurrent Messages Example



Selecting Control Mechanism



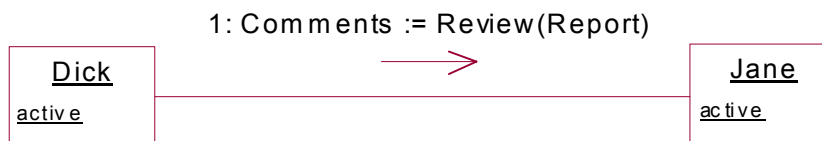
Dick requests Jane perform review of report & return comments

Selecting Control Mechanism (cont.)

Elements of interaction:

<i>Client Object:</i>	Dick
<i>Server/Supplier Object:</i>	Jane
<i>Operation:</i>	Review
<i>Object needed to perform the operation:</i>	Report
<i>Data returned as a result of the operation:</i>	Comments

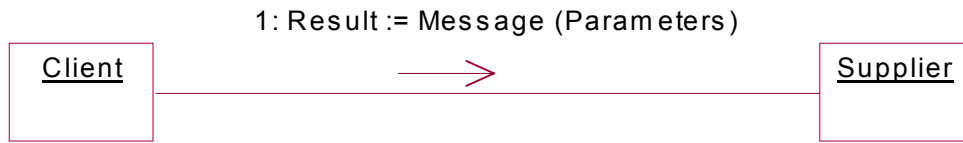
Representation:



Issue:

- What control mechanism to use?

Key Events



Start of interaction (issuing of request)

Start of communication (acceptance of request)

Processing of requested operation

End of communication

End of interaction

Start of Interaction

An object asks another object to perform some operation

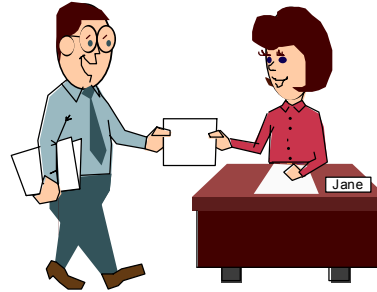


Active Objects may issue a request on their own power

Passive Objects can issue a request **only** in performing an operation

Start of Communication

Object accepts a request



Active object **chooses** when to accept a request

Passive object accepts a request immediately unless already processing another request

- Some passive objects can process > 1 request at a time
 - **e.g.** Multiple requests to read a file.
- For a shared passive object
 - Must define appropriate mutual exclusion mechanism in class operation

Processing of Requested Operation

Performance of the requested operation



Active object may perform:

- While client waits
- after releasing client, or
- Both (i.e., some work while client waits, some after)

Passive object can perform only while client waits

End of Communication & Interaction

Generally both end at same time (see “Interaction Mechanisms” below)

Performing object returns any pertinent objects or data, & releases client



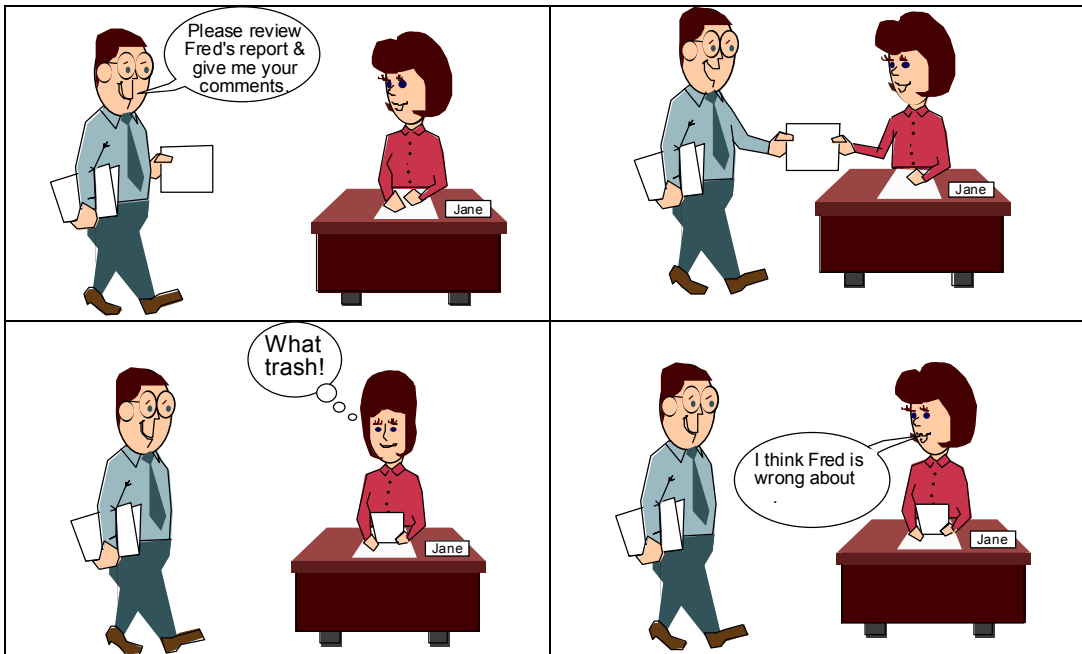
Active object decides what to do next

- May include completing processing of operation

Passive object waits for next request

Procedure/Synchronous Mechanisms

Client waits for both acceptance of request & end of communication



Timeout Mechanisms

Client waits a predetermined time for acceptance of request

- Gives up if request isn't accepted within time limit (ending the interaction)
- Or, if request is accepted, waits for end of communication



Balking Mechanisms

Client does not wait if acceptor is not ready

- Gives up if request is not accepted immediately (ending the interaction)
- Or, if request is accepted, waits for end of communication



Asynchronous Mechanisms

Client sends request for supplier to accept later

- Interaction ends once message is sent
- Communication occurs when request is accepted



Can't issue asynchronous request to passive object

- Passive object left alone has no power to perform

Concepts & Notation

Collaboration Diagrams

Sequence Diagrams

Meaning of Axes

Vertical axis represents time

- Normally increasing down the page
- Usually *relative* metric
 - i.e. Event_2 follows Event_1
- May be an *absolute* metric
 - For real-time applications
 - i.e. Event_2 occurs 5ms after Event_1

Horizontal axis represent different objects

- No significance in order

Option: Switch purpose of axes

One of few times in UML when position on diagram is important!

Object Lifeline

Object Lifeline

- Represents object playing a role

Notation

- Dashed line with classifier box at top
 - Labeled as in collaboration diagram

Object Name / Role : Class Name



Message & Stimulus

Stimulus

- Is communication between 2 objects
- Causes
 - Operation to be invoked
 - Signal to be raised
 - Object to be created
 - Object to be destroyed

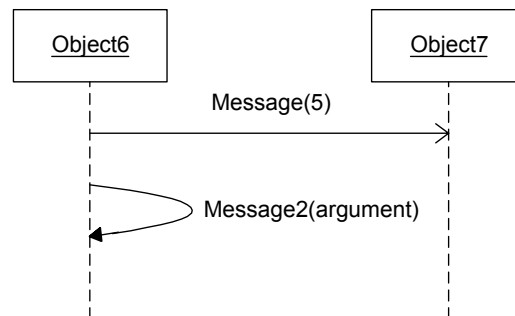
Message

- Specifies stimulus
 - Roles that sender & receiver objects must conform to
 - Action that dispatches conforming Stimulus when executed

Message & Stimulus Representation

Shown as

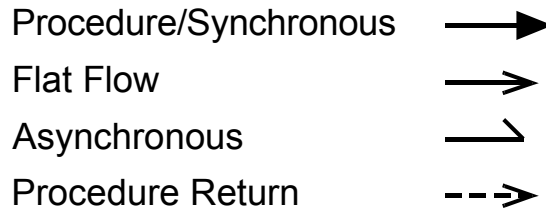
- Solid arrow from lifeline of 1 object to either
 - Lifeline of another object
 - Itself
- Label
 - Name of stimulus
 - ♦ Operation or Signal
 - Argument
 - ♦ Values or Expression
 - Sequence number (optional)
 - Recurrence² (optional)
 - Guard condition (optional)



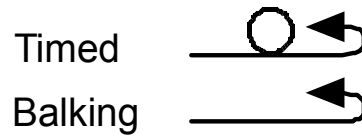
² Only described in the first paragraph of page 3-102.

Kinds of Communication

UML defines 3-styles of arrows for



– Common Extensions

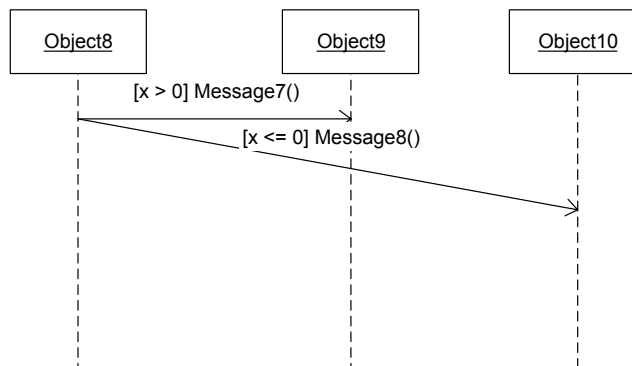


Procedural return is optional in “Procedural Sequence Diagram” (see below)

Branching

Multiple arrows can leave single point on object lifeline

- Each labeled by guard condition
 - If mutually exclusive → Conditional behavior
 - Otherwise → Concurrent³

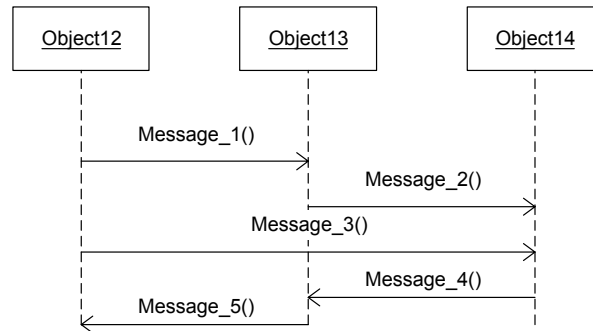


³ Not clear why need guard or why not using notation for concurrent iteration (“||”) used in Collaboration Diagram.

Predecessor Association

Messages are related by *predecessor* association

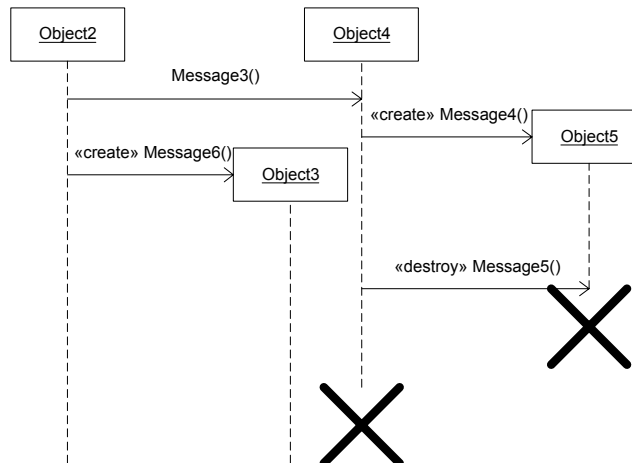
- Typically defined by
 - Successive arrows in vertical sequence
 - *Activation* (see below)
- May use sequence numbers
 - Needed if diagram has concurrent objects & concurrent arrows



Object Lifetime

Object

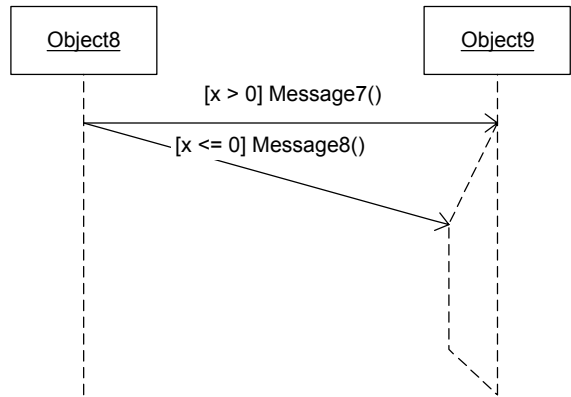
- Exist at start sequence
 - If box drawn at top of diagram
 - ◆ Above 1st arrow
- Created
 - If box drawn below top
 - ◆ Shows when object created
 - ◆ Box is target of arrow ==> Message is *CreateAction*
- Existence ends
 - If marked with "X"
 - ◆ If "X" is target of arrow ==> Message is *DestroyAction*
 - ◆ Otherwise, "X" ==> *TerminateAction*



Object Lifetime
Conditional Behavior

An object lifeline may split into two or more parallel lifelines

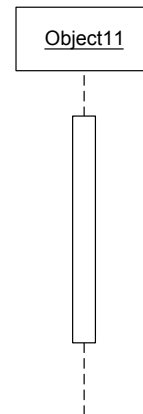
- Each track of split lifeline corresponds to conditional branch
- Branches may merge together at some later time



Activation

Semantics

- Shows period during which object is performing work either
 - Directly
 - Through subordinate procedure
- Represents both
 - Duration of action in time
 - Control relation between activation & its callers (stack frame)
- Also called *focus of control*



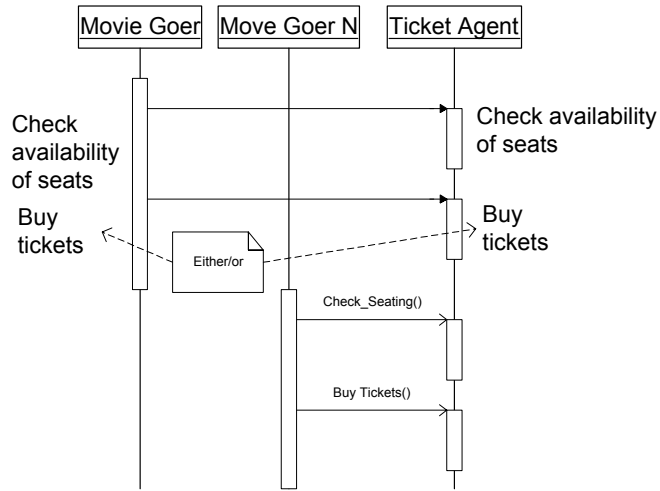
Notation

- Tall thin rectangle centered to object lifeline
 - Top → start of operation/action
 - Bottom → end of operation/action

Activation
Describing Action

3 ways to describe action

- Action being performed may be labeled
 - In text next to activation symbol
 - In the left margin
- Incoming message may indicate action
 - Action may be omitted from activation itself



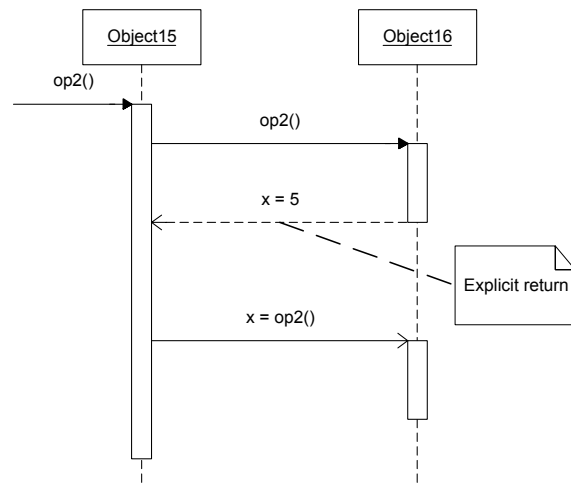
Suggestion:

- Use text in left margin to give natural language description of action
 - e.g. Use-case step
- Use message label to show actual message

Activation
Procedural Code

Activation shows duration during which procedure is active

- Time spent in
 - Procedure
 - Subordinate procedure(s)
 - ◆ Possibly in other object(s)
- Start of procedure
 - Incoming arrow lines up with top of activation
- End of procedure
 - Return message lines up with bottom of activation
 - ◆ Optional
 - ❖ Return is implicit when activation ends
 - ❖ Return results can be shown on call

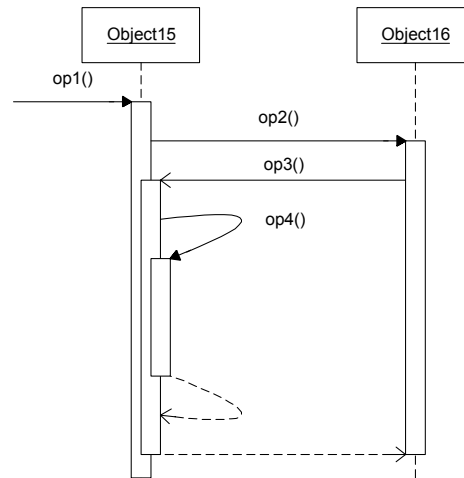


Activation

Recursive Calls

Recursive calls to object with existing activation

- Second activation symbol is drawn slightly to right of first one
- Subsequent activation symbols appear to "stack up" visually
 - May be nested to an arbitrary depth



Activation

Concurrent Objects

Each active object has own thread of control

- Activation shows the duration when each object is performing an operation
- Operation by other objects are not relevant
 - Although, objects may communicate via
 - ◆ Asynchronous messages
 - ◆ Synchronous messages

Entire lifeline may be shown as activation when there does not need for distinction between

- Direct computation
- Indirect computation (e.g. by a nested procedure)

Procedural Sequence Diagrams

Procedural Sequence Diagram is

- Diagram with calls & visible focus of control

Arrow sequence defines *predecessor* associations between Messages

Arrow to head of a focus of control region → a nested *activation*

- Message between the instances playing ClassifierRoles
 - ◆ *CallAction* performed by instance at source of arrow
 - ◆ Defines target Operation & arguments
- Each arrow departing focus →
 - ◆ Message has *activation* association with message at top of activation
- Optional explicit *return* arrow
 - ◆ Returns value (optional)
 - ◆ Must be final message in predecessor chain

Transition Time

For most systems & most messages,

- Time to transmit message from sender to receiver is insignificant
- Nothing else can happen during transmission
 - Delivery is non-atomic
- Message arrow is drawn horizontally

For messages that are not atomic

- Can draw arrow so arrow head is below tail

Time to transmit may be represented by Time Markers

- Time markers can be constrained

Timing Markers

Messages may have different times that are of interest

e.g. send, receive, elapsed, execution, queued time

UML defines functions that can be used in constraints

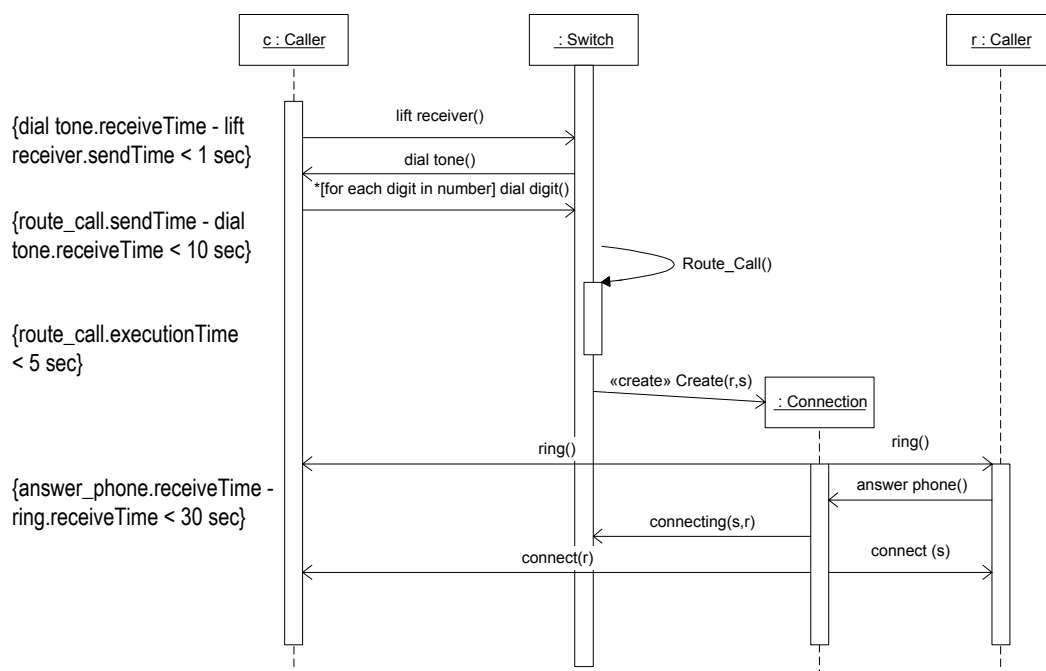
- Predefined
 - sendTime — Time when message is sent
 - receiveTime — Time when message is received
- User can invent new ones

Notation

Message_Name.Time_Function_Name

- sendTime function can be represented by just name of message
- receiveTime function can be represented by just name of message followed by tic mark (')

Time Maker Example



Use-Case Representation Example
Use-Case Description

Use Case Name	Test Sensors
Purpose	Test whether any sensors are out of limits.
Actors	Timer, Sensor, Fault Light, Limit Light
Importance	Primary
Overview	A timer signal occurs. The system reads (the memory location assigned to each) sensor for which monitoring is enabled. If the value is not available, the system waits until the next signal (5 seconds) and tries again, if a value is still not available, the system turns on the Fault light. If the value is available, the system checks whether the value is in the current limits. If the value is out of limits, the system turns on the Limit Light for the specified sensor.
Requirements	R1.4.1, R.1.4.2, R2.3 (see Q4)
Status	Real
Uses	

Typical Course of Action

	Actor Actions	System Response
1.	Timer signal occurs	
2		For each sensor
2.1		Read sensor value from sensor's memory location (16#FF# -- see Q8)
2.2		[Value != "None" value] test the value against current limits
2.3		[Value is in limits] Write "None" value to sensor's memory location (16#FF# -- see Q8)
2.4		Set Last_Value to value read

Use-Case Representation Example
Use-Case Description (cont.)

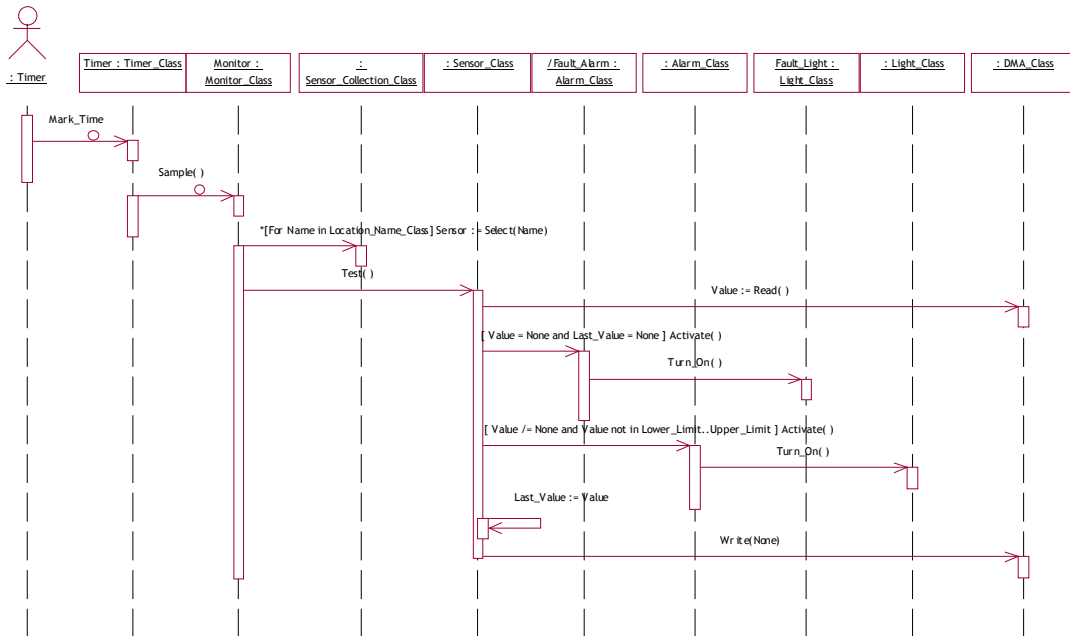
Alternate Course of Action: *Value is Out Of Limits*

	Actor Actions	System Response
2.3		[Value is not in limits] turn on limit light for sensor by writing to light's port
2.4		End Of Course

Exceptional Course of Action: *Sensor May be Broken*

	Actor Actions	System Response
2.2		[Value = "None" value]
2.3		[Last_Value = "None" value] turn on fault light by writing to light's port
2.4		End Of Course

**Use-Case Representation Example
Sequence Diagrams**



Outline

Operation & Interaction Concepts

Models & Diagrams

Notations & Guidelines

Summary

Key Points

Interaction Diagrams represent

- Set of objects cooperating to implement some capability
 - An operation
 - A class
 - A use-case
- Behavior of objects
 - Messages exchanged
 - Conditions for exchange
 - Information that's exchanged
 - Control of communication
 - Message timing

2 diagram styles

- Collaboration
- Sequence