WIA2002: Software Modelling

Semester 1, Session 2016/17

Lecture 8: Modelling Object Interactions - UML Interaction Diagrams

(Part 1: Sequence Diagrams)

Learning Objectives

- Know how to develop object interaction from use cases.
- Understand what are the messages and actions supported by UML interaction diagrams.
- Know how to model object interaction using an interaction sequence diagram.
- Know how to model complex interactions using different techniques.
- Know how to cross-check between interaction diagrams and a class diagram.

Object Interaction

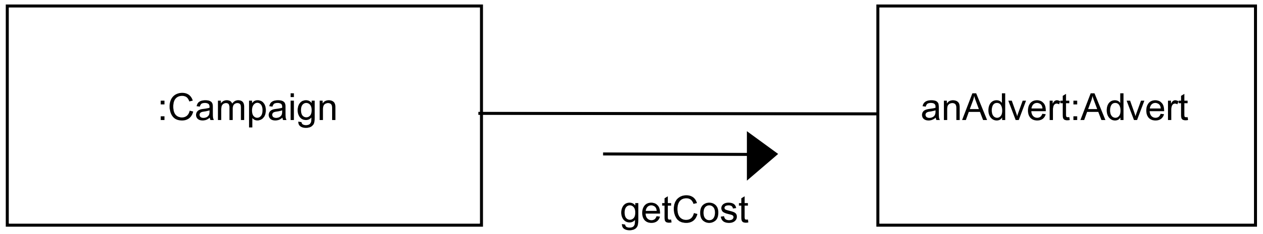
- An object is a black box.
  - Its inner workings and its knowledge are hidden from the outside world.
  - The users of the system and other objects—may interact with the object only through its public interface.

- How do the users interact with the system?
  - The answer is by exchanging messages.

Object Messaging

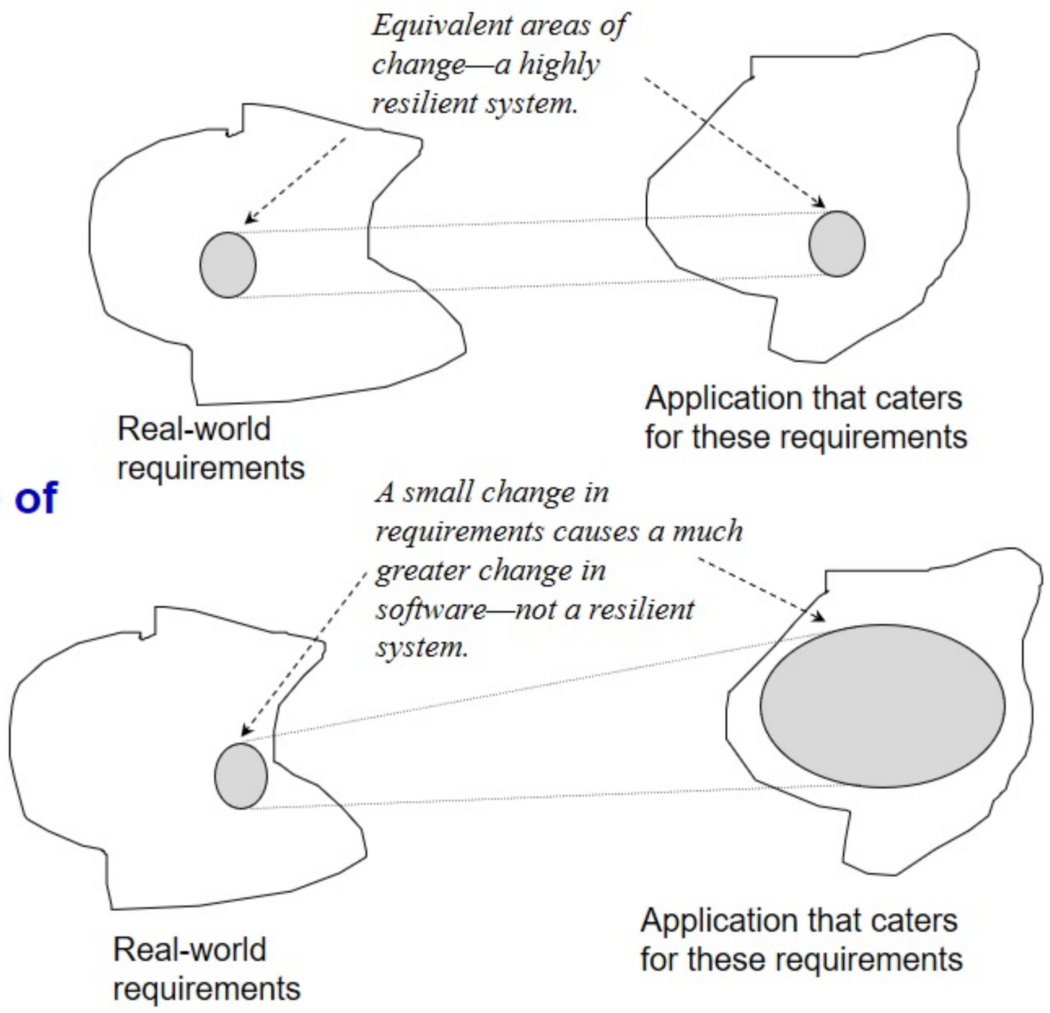
- Objects communicate by sending messages.
- Sending the message getCost() to an Advert object, might use the following syntax.

currentadvertCost = anAdvert.getCost()



Object Messaging

Resilience of Design


Equivalent areas of change—a highly resilient system.
Real-world requirements
Application that caters for these requirements
A small change in requirements causes a much greater change in software—not a resilient system.
Real-world requirements
Application that caters for these requirements

Interaction & Collaboration

- A collaboration is a group of objects or classes that work together to provide an element of functionality or behaviour.

- An interaction defines the message passing between lifelines (e.g. objects) within the context of a collaboration to achieve a particular behaviour.

Modelling Interactions

- Interactions can be modelled using various notations.
  1. Interaction sequence diagrams
  2. Communication diagrams
  3. Interaction overview diagrams
  4. Timing diagrams

Messages & Actions

- A message is a communication between two objects, or within an object, that is designed to result in some activity.

- This activity involves one or more actions, which are executable statements that result in
  - changes in the values of one or more attributes of an object,
  - or the return of some value(s) to the object that sent the message,
  - or both

Messages & Actions (cont.)

- There are 5 kinds of actions that the UML explicitly supports: -
  - Call and Return
  - Create and Destroy
  - Send

Messages & Actions (cont.)

Action	Description
Call	• A call action invokes an operation on an object • It is synchronous - the sender assumes that the receiver is ready to accept the message, and the sender waits for a response from the receiver before proceeding
Return	• A return action is the return of a value to the caller, in response to a call action
Create	• A create action creates an object • It tells a class to create an instance of itself
Destroy	• A destroy action destroys an object • An object can perform a destroy action on another object, or on itself
Send	• A send action sends a signal to an object • A signal is an asynchronous communication between objects • does not expect a response from the receiver (unlike a call action)

1. SEQUENCE DIAGRAMS

Sequence Diagrams

- In the UML, an interaction diagram is used to model the dynamic aspect of a system.
- These dynamic aspect may involve : -
  - the interaction of any kind of instance in any view of a system's architecture, including instances of a class, interface, components and nodes.
  - a system as a whole, a subsystem, an operation or a class.

- Interaction diagrams can also be attached to
  - use cases : to model scenarios.
  - collaborations : to model the dynamic aspects of a society of objects.

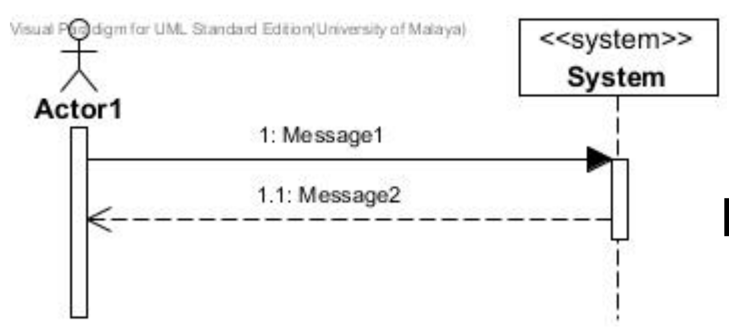
Sequence Diagrams

- Illustrate the objects that participate in a use case and the messages that pass between them over time for one use case.
- Show the explicit sequence of messages that are passed between objects in a defined interaction.
- Emphasize the time-based ordering of the activity that takes place among a set of objects.
  - Very helpful for understanding real-time specifications and complex use cases.

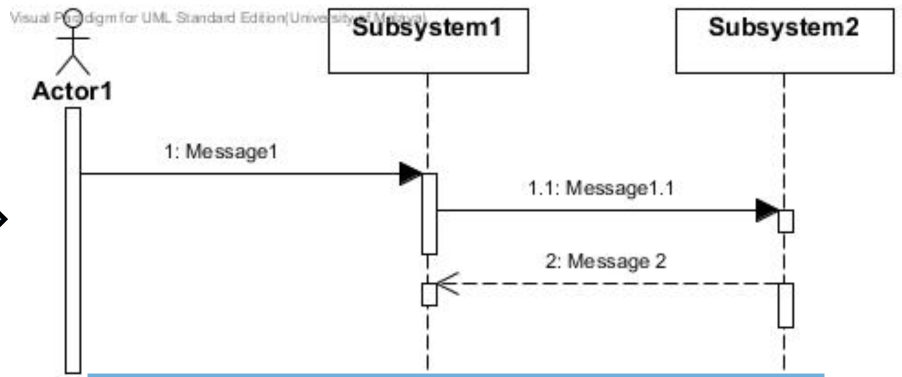
Purpose of Sequence Diagrams

- Shows an interaction between lifelines (e.g. objects) arranged in a time sequence.
- Can be drawn at different levels of detail and to meet different purposes at several stages in the development life cycle.
- Typically used to represent the detailed object interaction that occurs for one use case or for one operation.

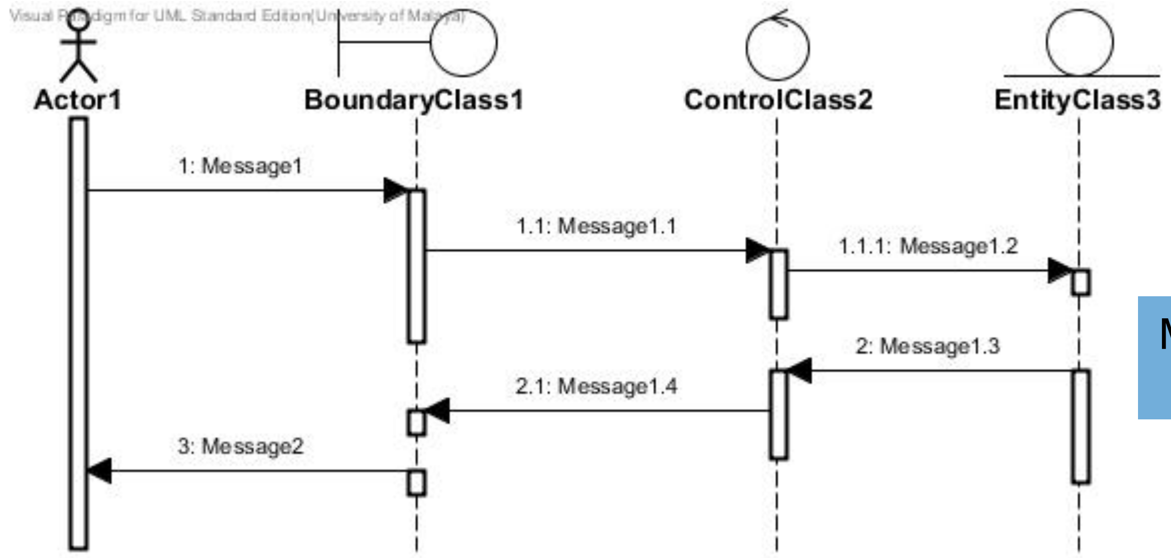
Levels of Sequence Diagrams


Visual Paradigm for UML Standard Edition(University of Malaya)
Actor1
<<system>>
System
1: Message1
1.1: Message2

System-level Sequence Diagram



Subsystem-level Sequence Diagram

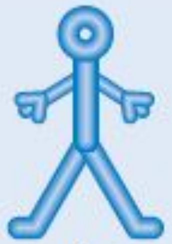
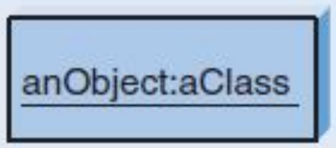
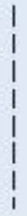


MVC-level / Three-tier Sequence Diagram

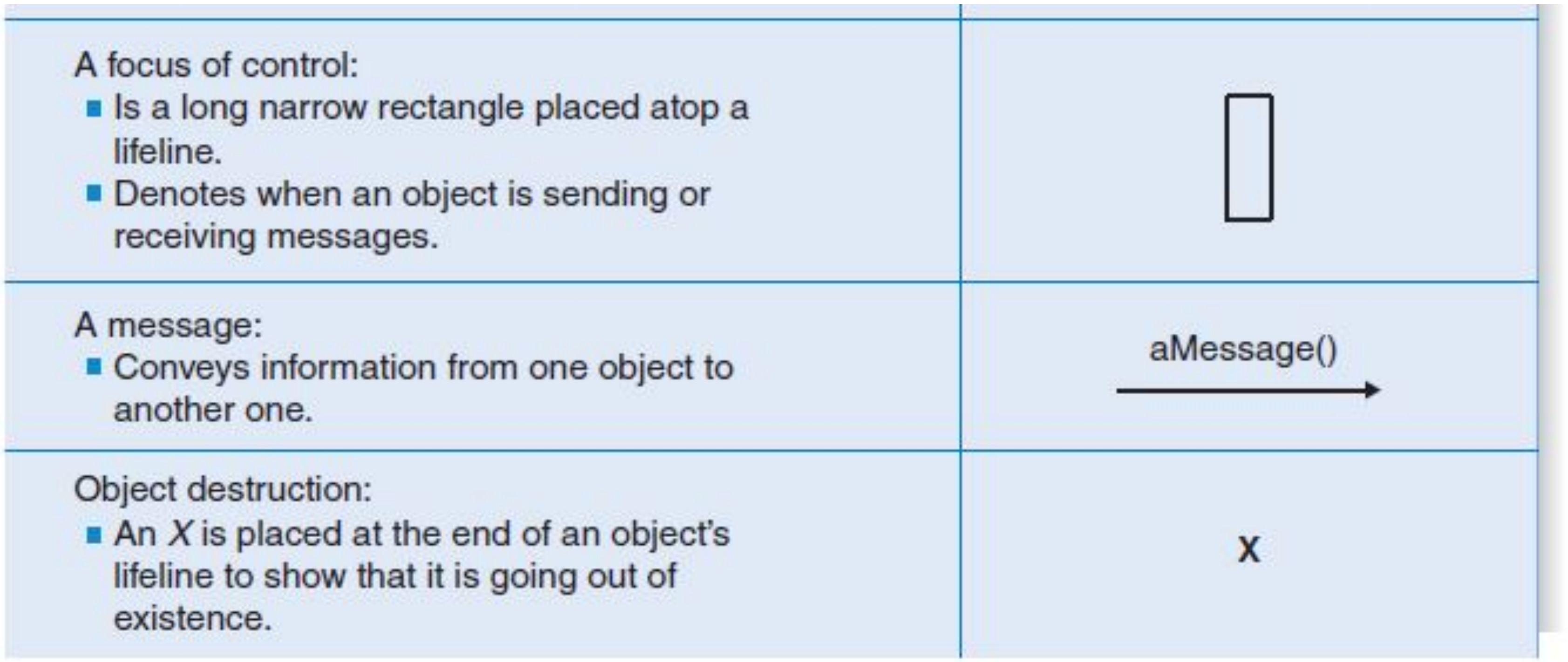
Notations of Sequence Diagrams

- Vertical dimension shows time.
- Objects (or subsystems or other connectable objects) involved in interaction appear horizontally across the page and are represented by lifelines.
- Messages are shown by a solid horizontal arrow.
- The execution or activation of an operation is shown by a rectangle on the relevant lifeline.

Notations of Sequence Diagrams

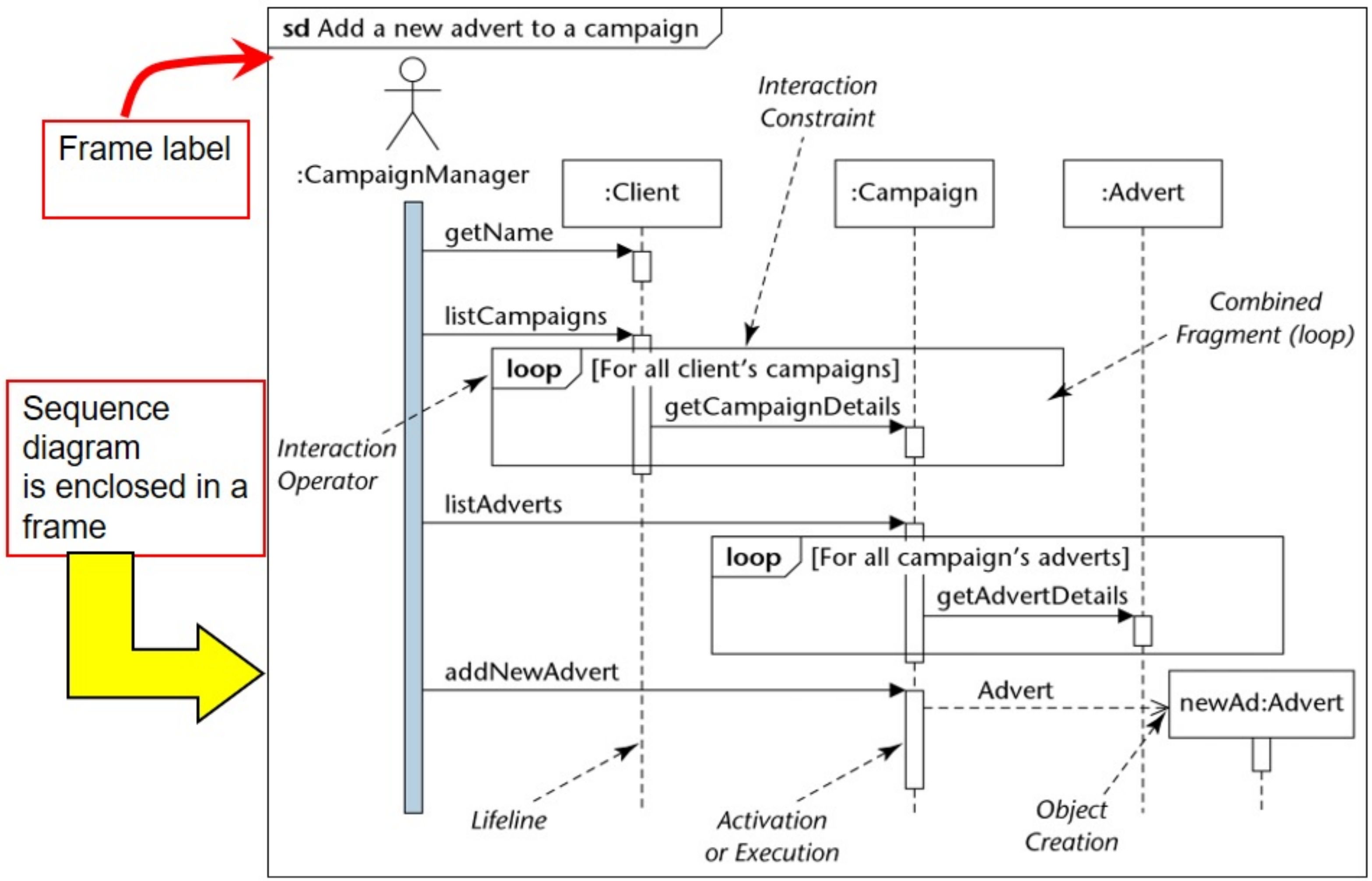
Term and Definition	Symbol
An actor: ▪ Is a person or system that derives benefit from and is external to the system. ▪ Participates in a sequence by sending and/or receiving messages. ▪ Is placed across the top of the diagram.	 anActor
An object: ▪ Participates in a sequence by sending and/or receiving messages. ▪ Is placed across the top of the diagram.	
A lifeline: ▪ Denotes the life of an object during a sequence. ▪ Contains an X at the point at which the class no longer interacts.	

Notations of Sequence Diagrams



A focus of control: ■ Is a long narrow rectangle placed atop a lifeline. ■ Denotes when an object is sending or receiving messages.	▯
A message: ■ Conveys information from one object to another one.	$\xrightarrow{\text{aMessage()}}$
Object destruction: ■ An *X* is placed at the end of an object's lifeline to show that it is going out of existence.	X

Notations of Sequence Diagrams



Notations of Sequence Diagrams

1. The *getName* message is the first message received by the *Client*.
2. The *Client* object then receives a *listCampaigns* message.
3. The *Client* object now sends a message *getCampaignDetails* to each *Campaign* object in turn in order to build up a list of campaigns. This repeated action is called an iteration (loop).
4. The *Campaign Manager* next sends a message to a *Campaign* object asking it to list its advertisements.

Notations of Sequence Diagrams

5. The *Campaign* object delegates responsibility for getting the advertisement title to each *Advert* object.
6. When a new advertisement is added to a campaign, an *Advert* object is created.

Notations of Sequence Diagrams

- Iteration is represented by *combined fragment* rectangle with the *interaction operator* ‘loop’.

- The loop combined fragment only executes if the guard condition in the interaction constraint evaluates as true.

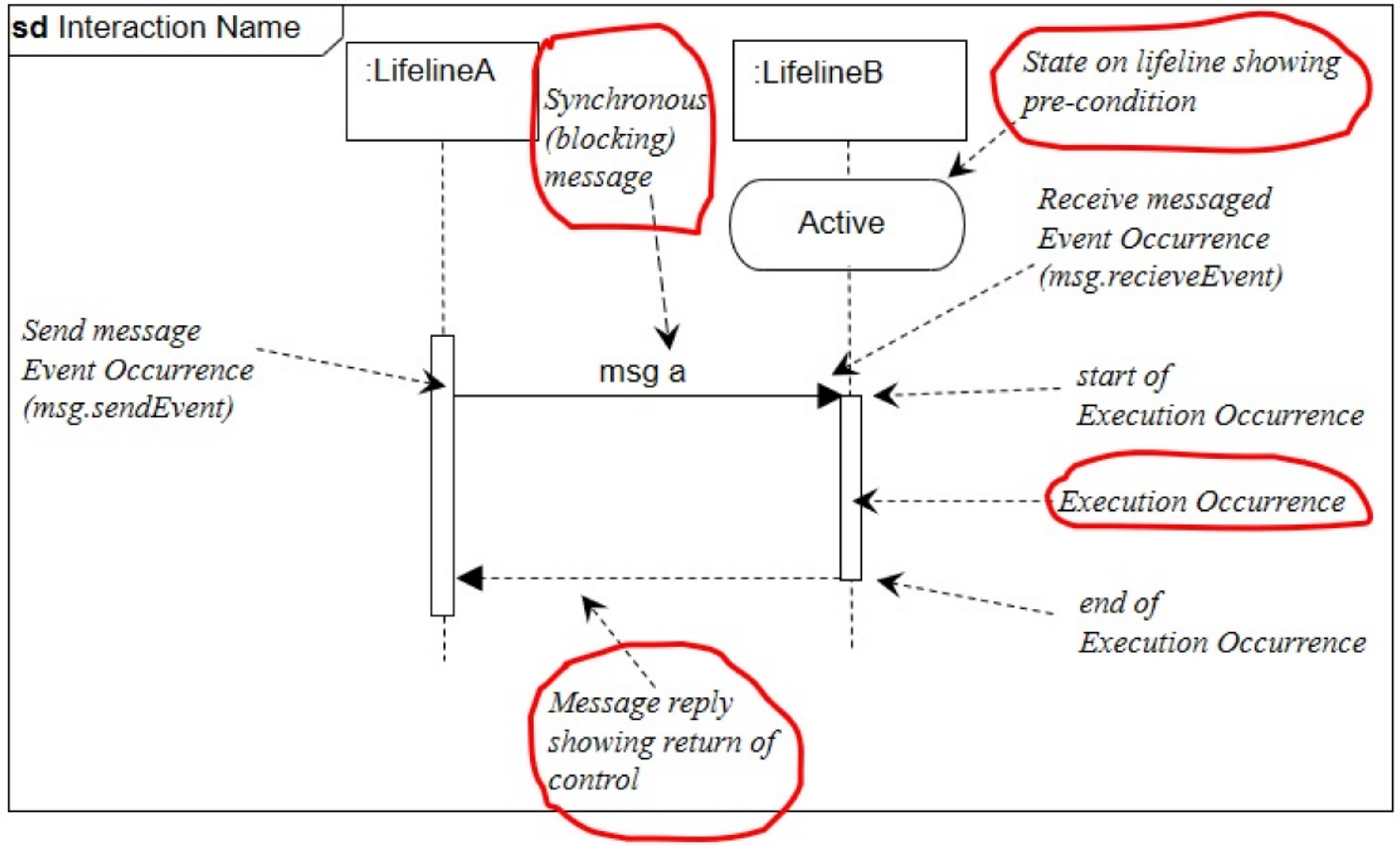
- Object creation is shown with the construction arrow (dashed) going to the object symbol for the Advert lifeline.

Synchronous Message

- A *synchronous message* or *procedural call* is shown with a full arrowhead.

- Cause the invoking operation to suspend execution until the focus of control has been returned to it.

Synchronous Message (Notations)

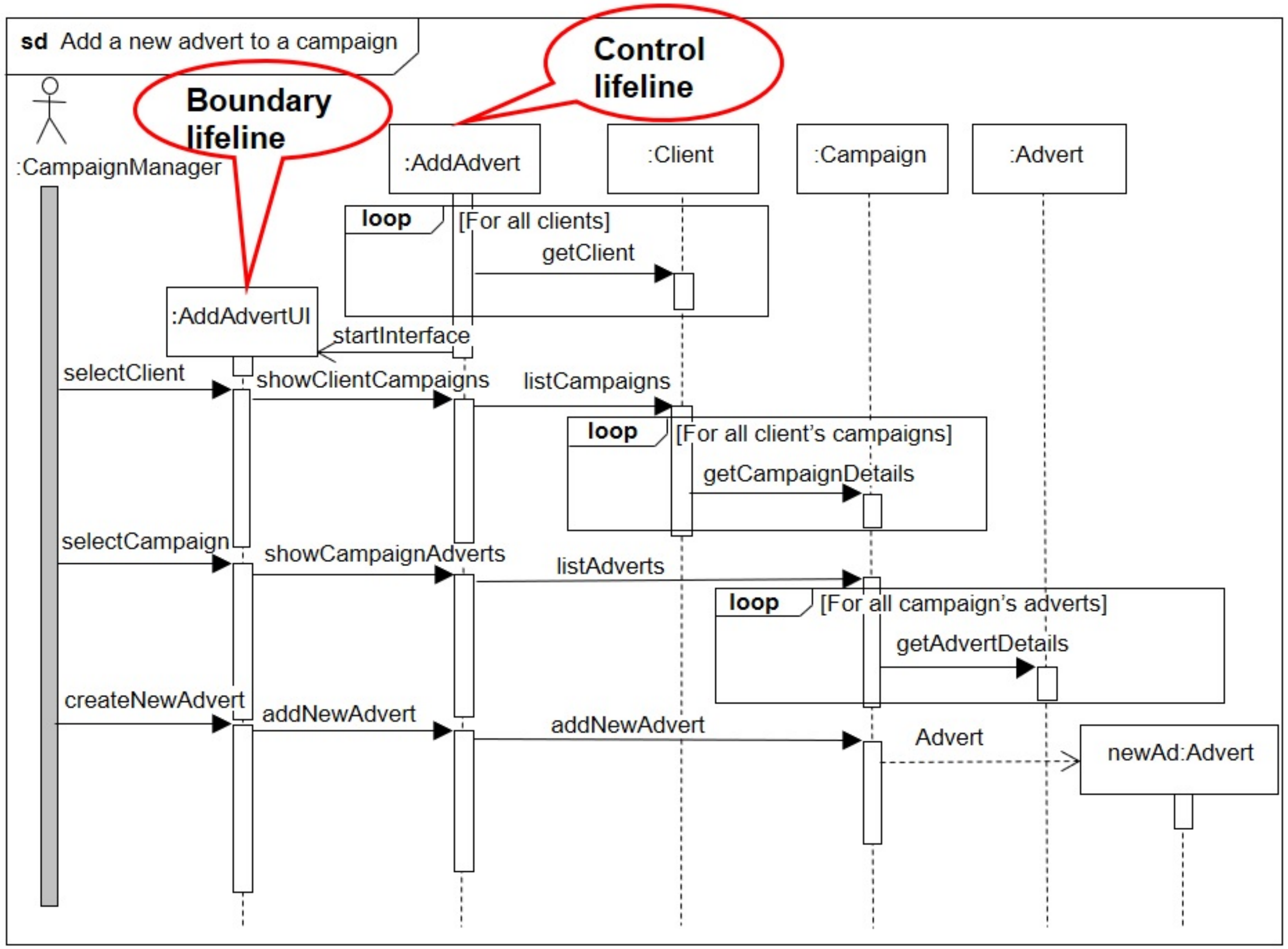

sd Interaction Name
:LifelineA
Synchronous (blocking) message
:LifelineB
State on lifeline showing pre-condition
Active
Receive messaged Event Occurrence (msg.recieveEvent)
Send message Event Occurrence (msg.sendEvent)
msg a
start of Execution Occurrence
Execution Occurrence
end of Execution Occurrence
Message reply showing return of control

Boundary & Control Classes

- Most use cases imply *at least one boundary object* that manages the dialogue between the actor and the system – in the next sequence diagram it is represented by the lifeline `:AddAdvertUI`

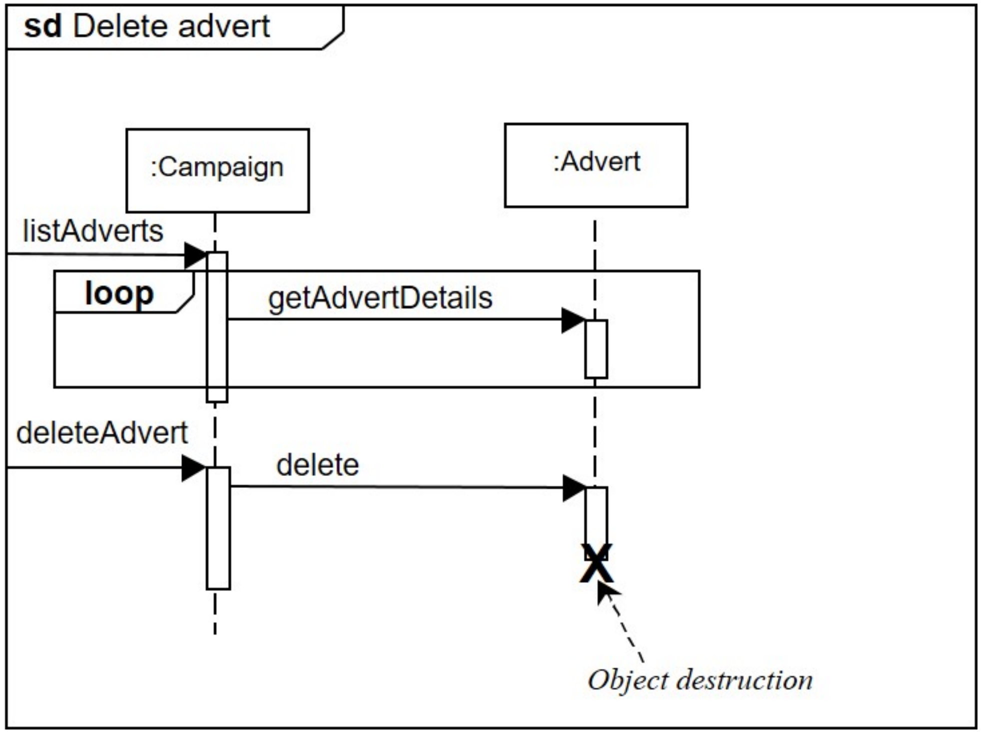
- The control object is represented by the lifeline `:AddAdvert` and this manages the overall object communication.

Boundary & Control Classes

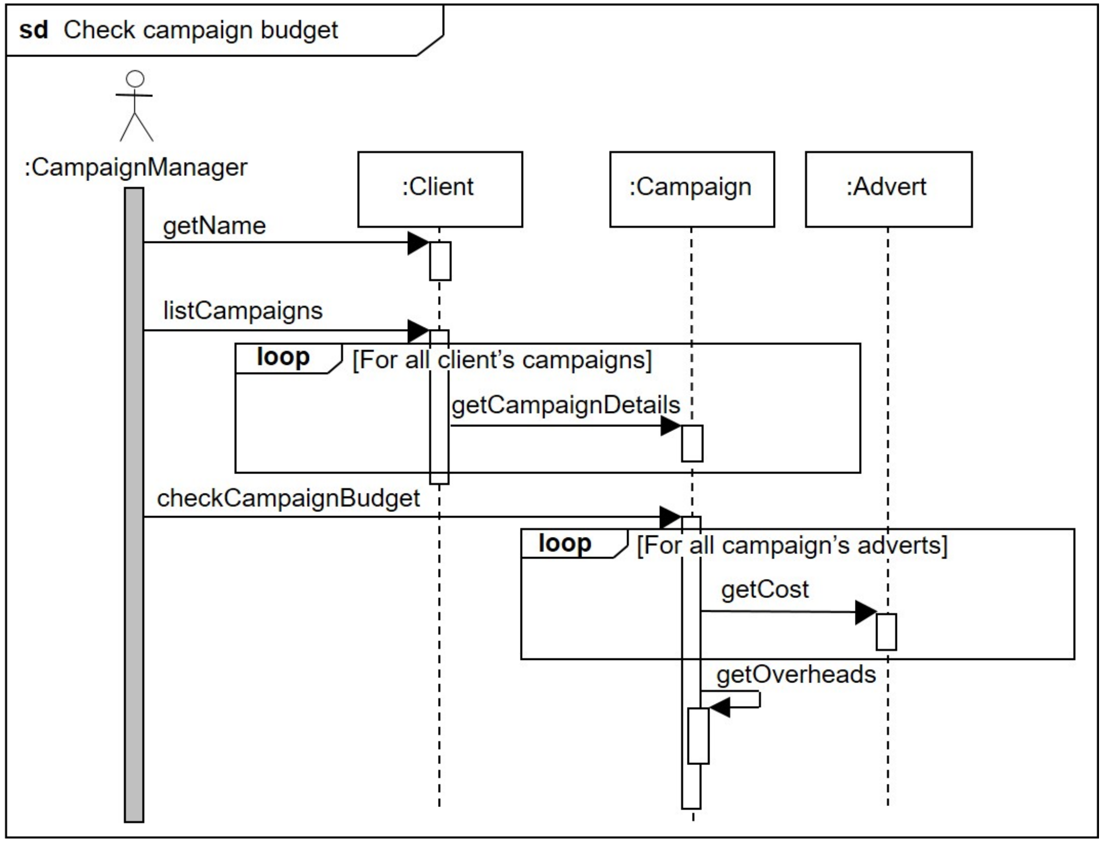


Object Destruction

- On a sequence diagram, the *destruction of an object* is indicated by *a large X* on the lifeline at the point in the interaction when the object is destroyed.



Reflexive Messages



- An object can send a message to itself.
- This is shown by a message arrow that starts and finishes at the same object lifeline.

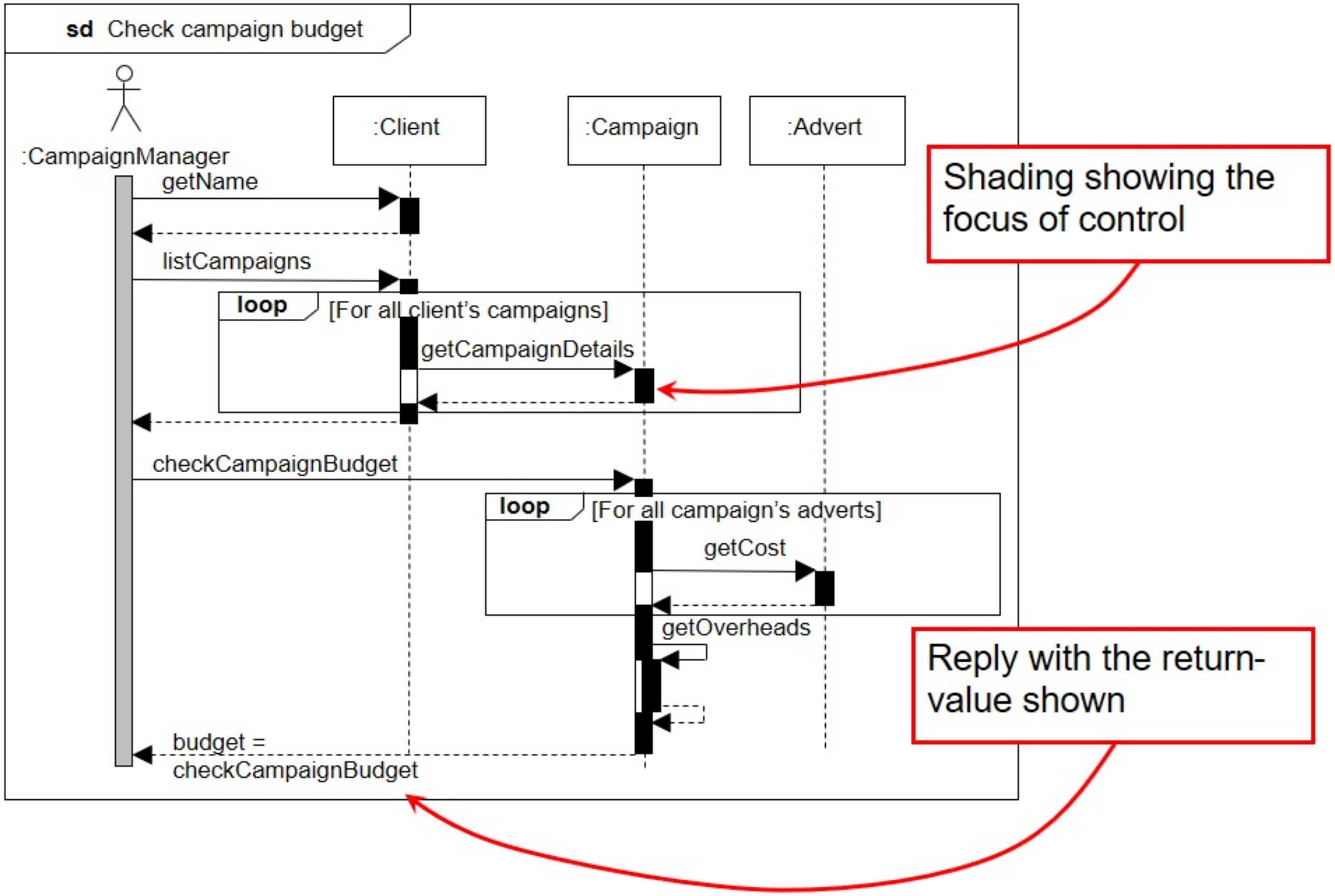
Focus of Control

- Indicates *times during an activation* when processing is taking place within that object.

- Parts of an activation that are not within the focus of control represent periods when
  - for example, an operation is waiting for a return from another object.

- May be shown by shading those parts of the activation rectangle *that correspond to active processing* by an operation.

Focus of Control



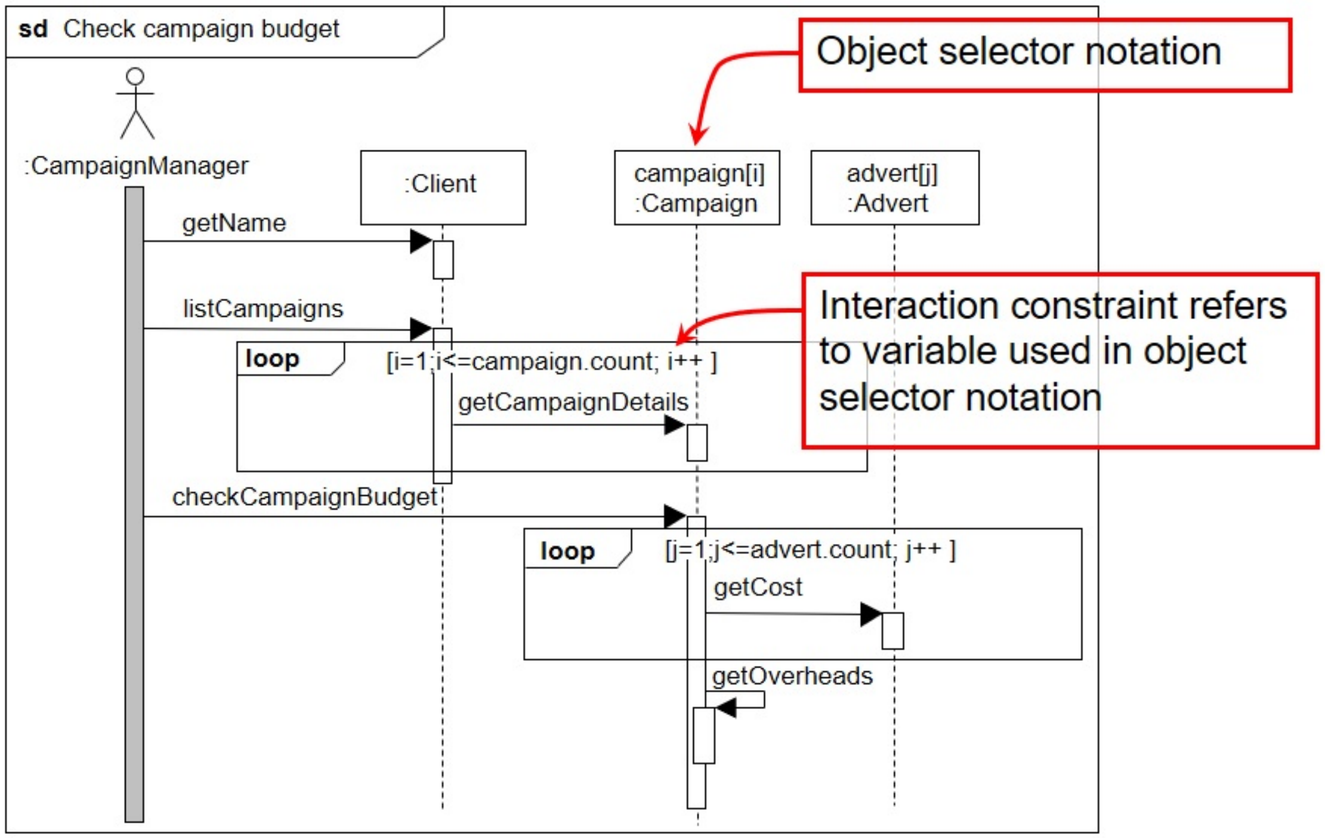
sd Check campaign budget
:CampaignManager
:Client
:Campaign
:Advert
getName
listCampaigns
loop [For all client's campaigns]
getCampaignDetails
checkCampaignBudget
loop [For all campaign's adverts]
getCost
getOverheads
budget = checkCampaignBudget
Shading showing the focus of control
Reply with the return-value shown

Reply Message

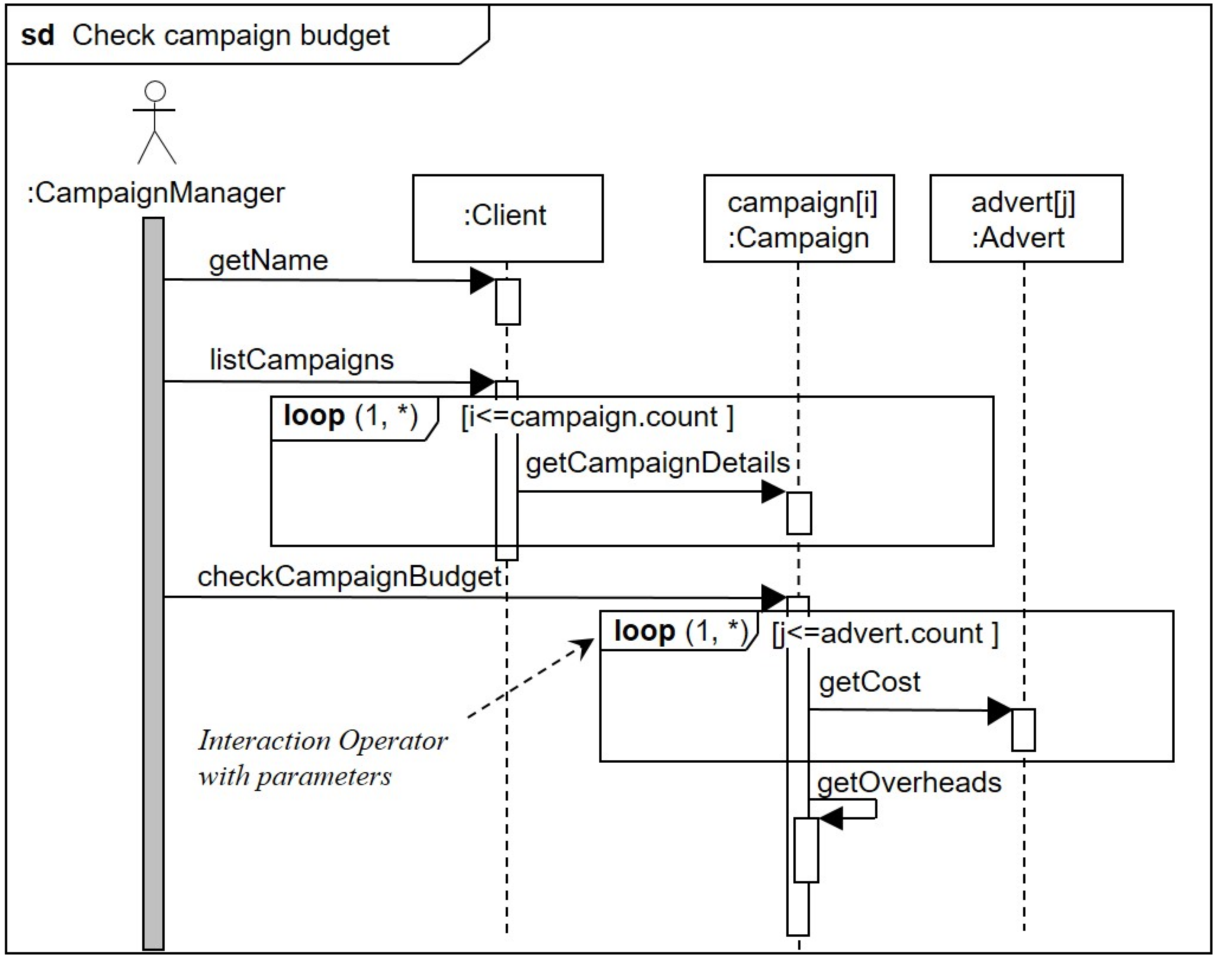
- A *reply message* returns the control to the object that originated the message that began the activation.
- Reply messages are shown with a dashed arrow, but it is optional to show them at all since it can be assumed that control is returned to the originating object.

Object Selector Notation

- To show explicitly that *a set of Campaign objects* **[campaign[i]:Campaign]** will be involved in the interaction one after the other.



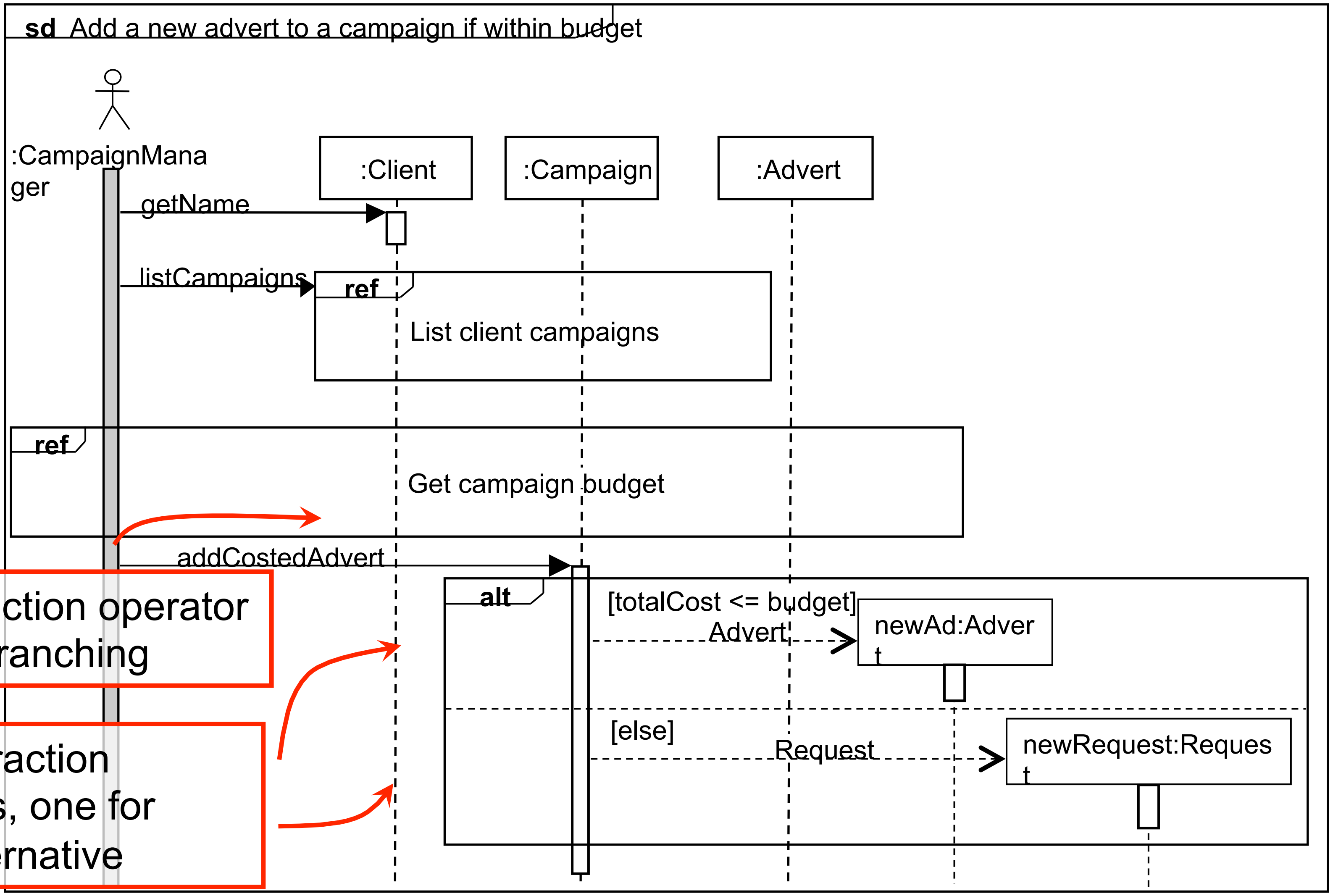
Interaction Operators



The loop interaction operator is shown with the parameter. The first parameter is the minimum number of interactions and the second is the maximum number of interactions.

s

Branching



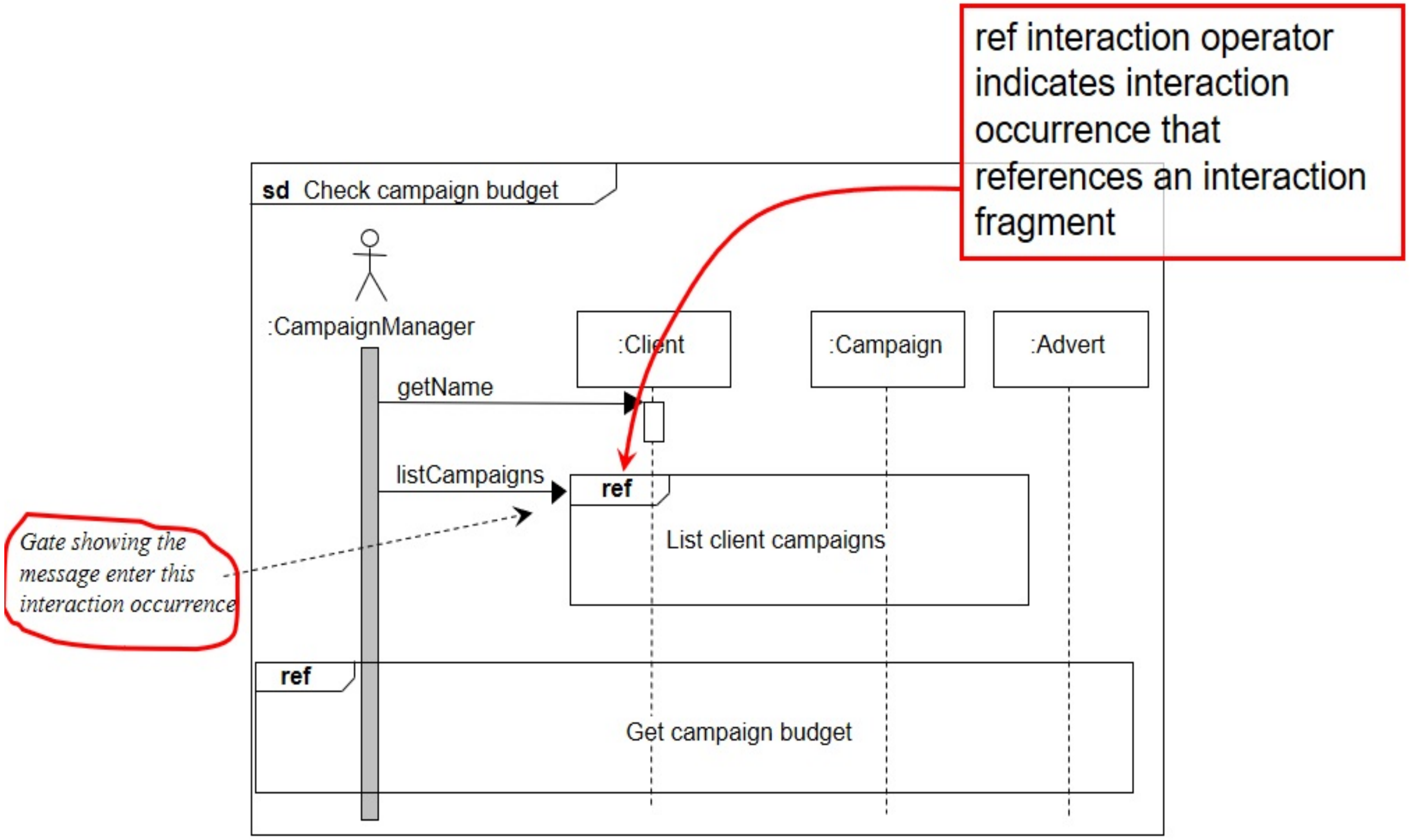
alt interaction operator shows branching

Two interaction operands, one for each alternative

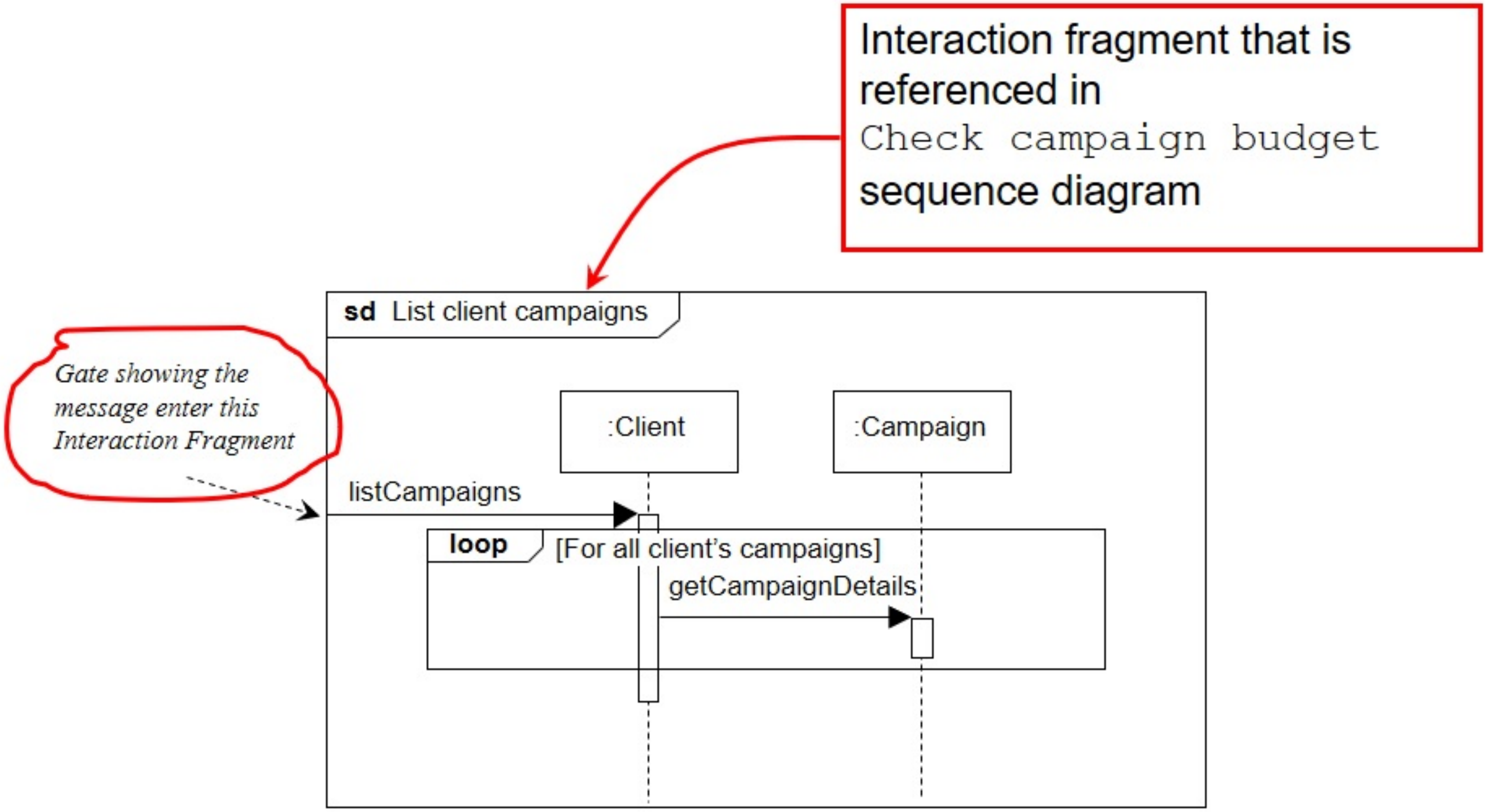
Handling Complexity

- Complex interactions can be modelled using various different techniques
  a. Interaction fragments
  b. Lifelines for subsystems or groups of objects
  c. Continuations
  d. Interaction Overview Diagrams (see later lecture)

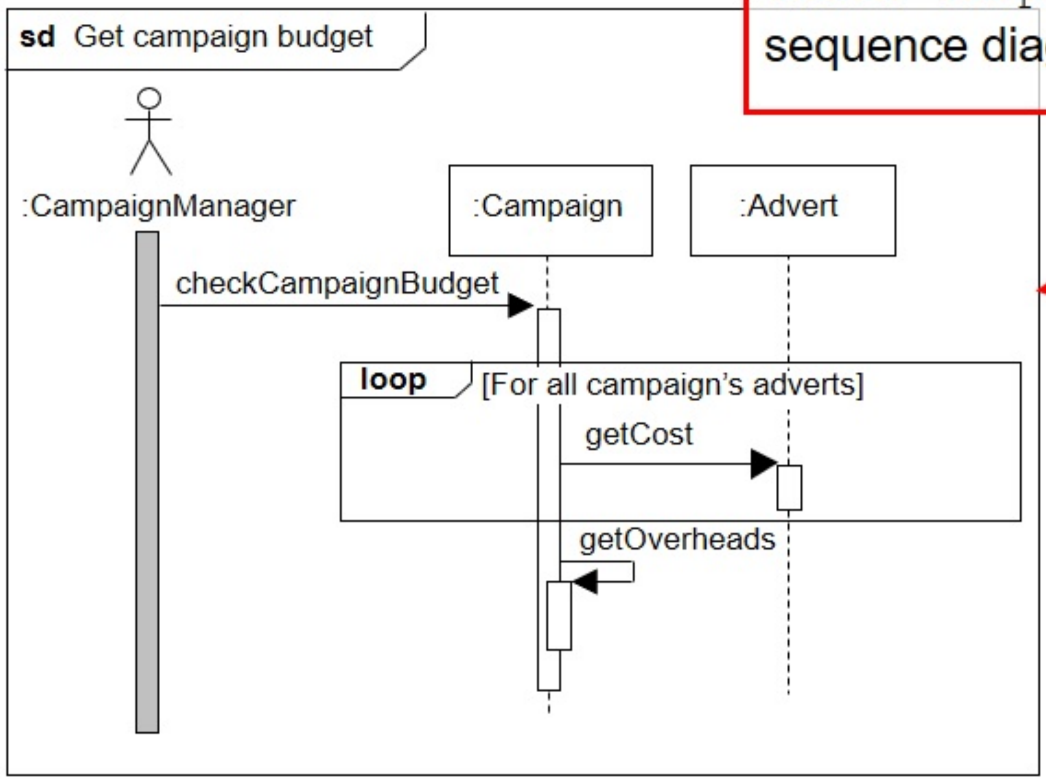
a. Using Interaction Fragments


ref interaction operator indicates interaction occurrence that references an interaction fragment
sd Check campaign budget
:CampaignManager
:Client
:Campaign
:Advert
getName
listCampaigns
ref
List client campaigns
Gate showing the message enter this interaction occurrence
ref
Get campaign budget

Interaction Fragment

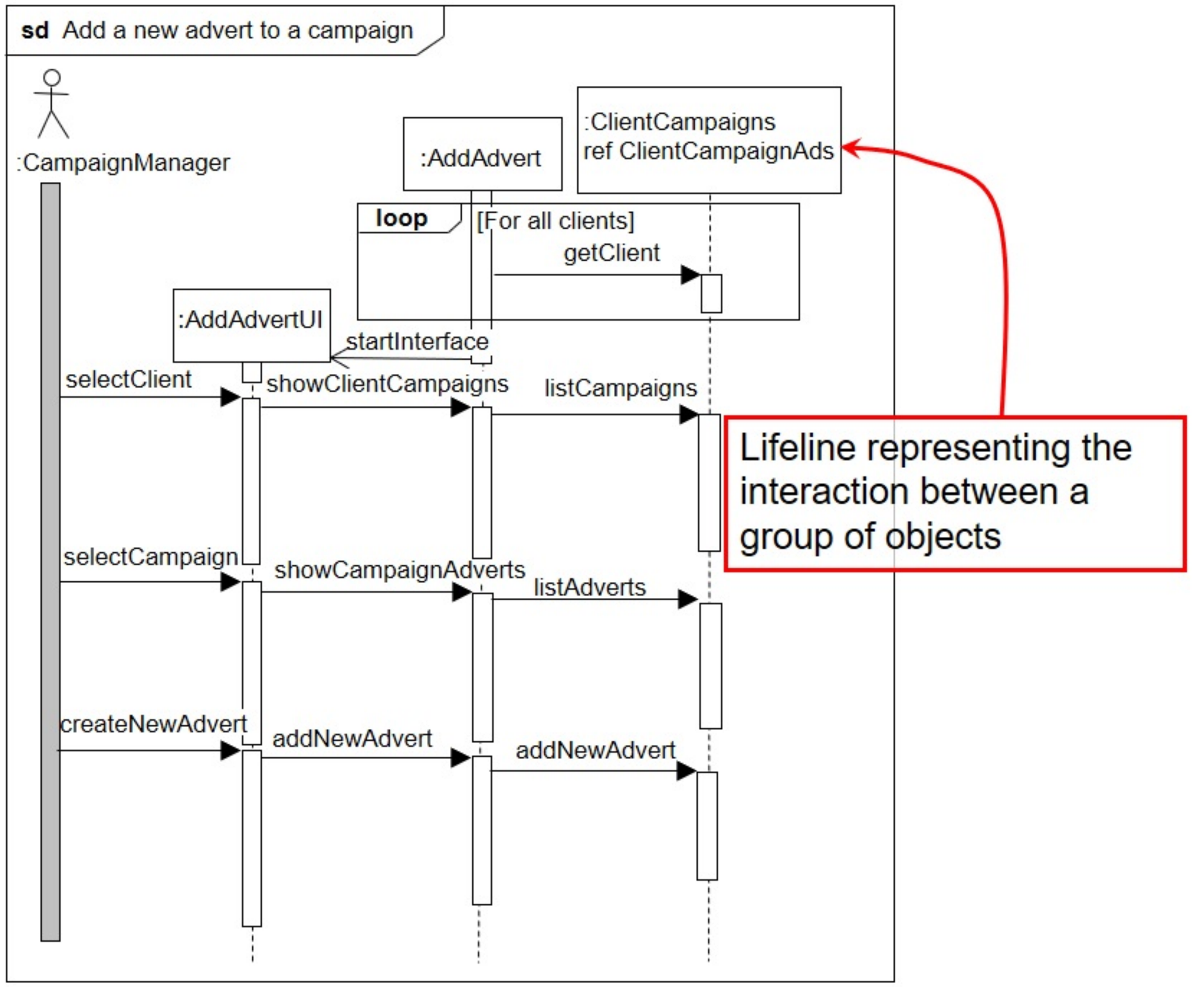

Interaction fragment that is referenced in `Check campaign budget` sequence diagram
Gate showing the message enter this Interaction Fragment
sd List client campaigns
:Client
:Campaign
listCampaigns
loop
[For all client's campaigns]
getCampaignDetails

Interaction Fragment

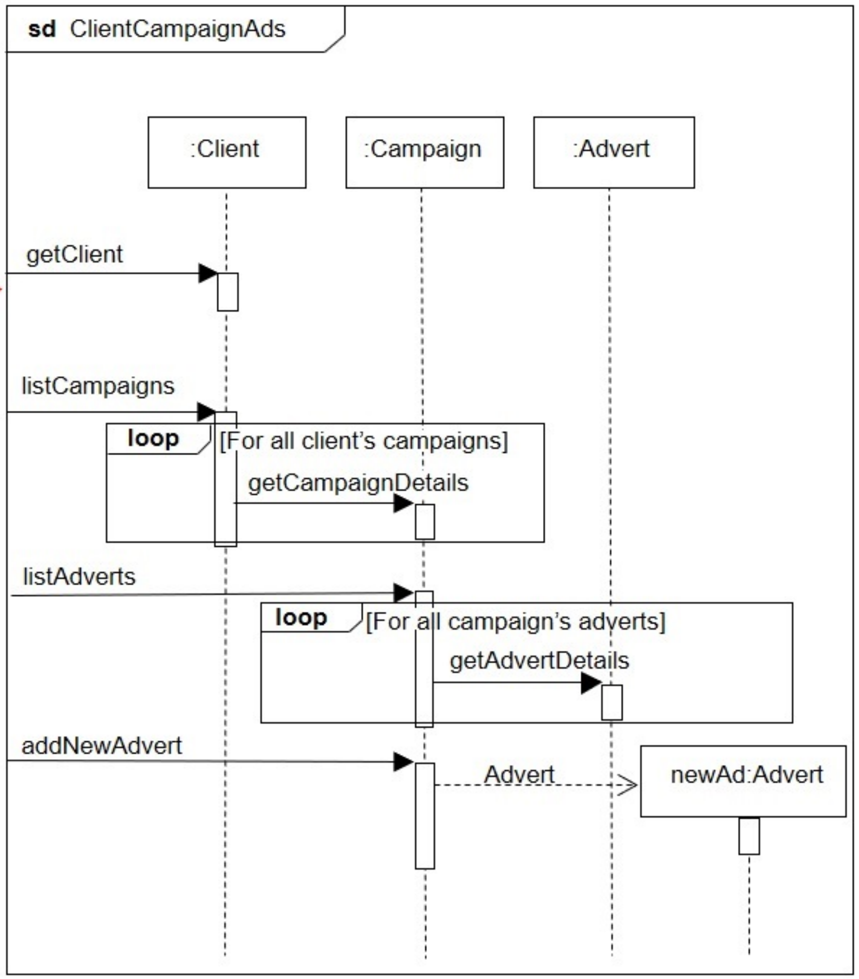


Interaction fragment that is also referenced in `Check campaign budget` sequence diagram

b. Using Lifelines for subsystems or groups of objects

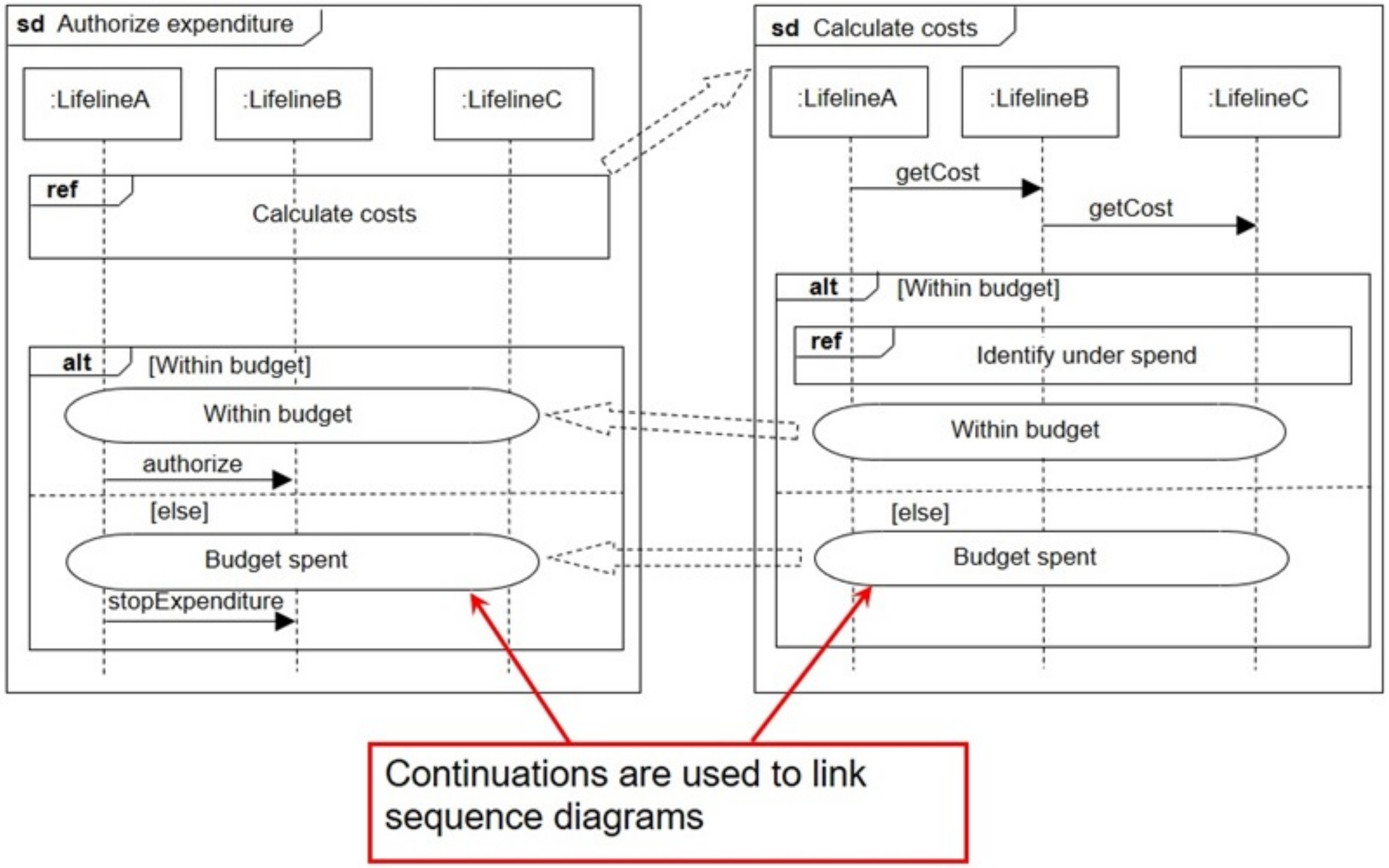


b. Using Lifelines for subsystems or groups of objects



Sequence diagram referenced in the `Add a new advert to a campaign` sequence diagram

c. Continuations

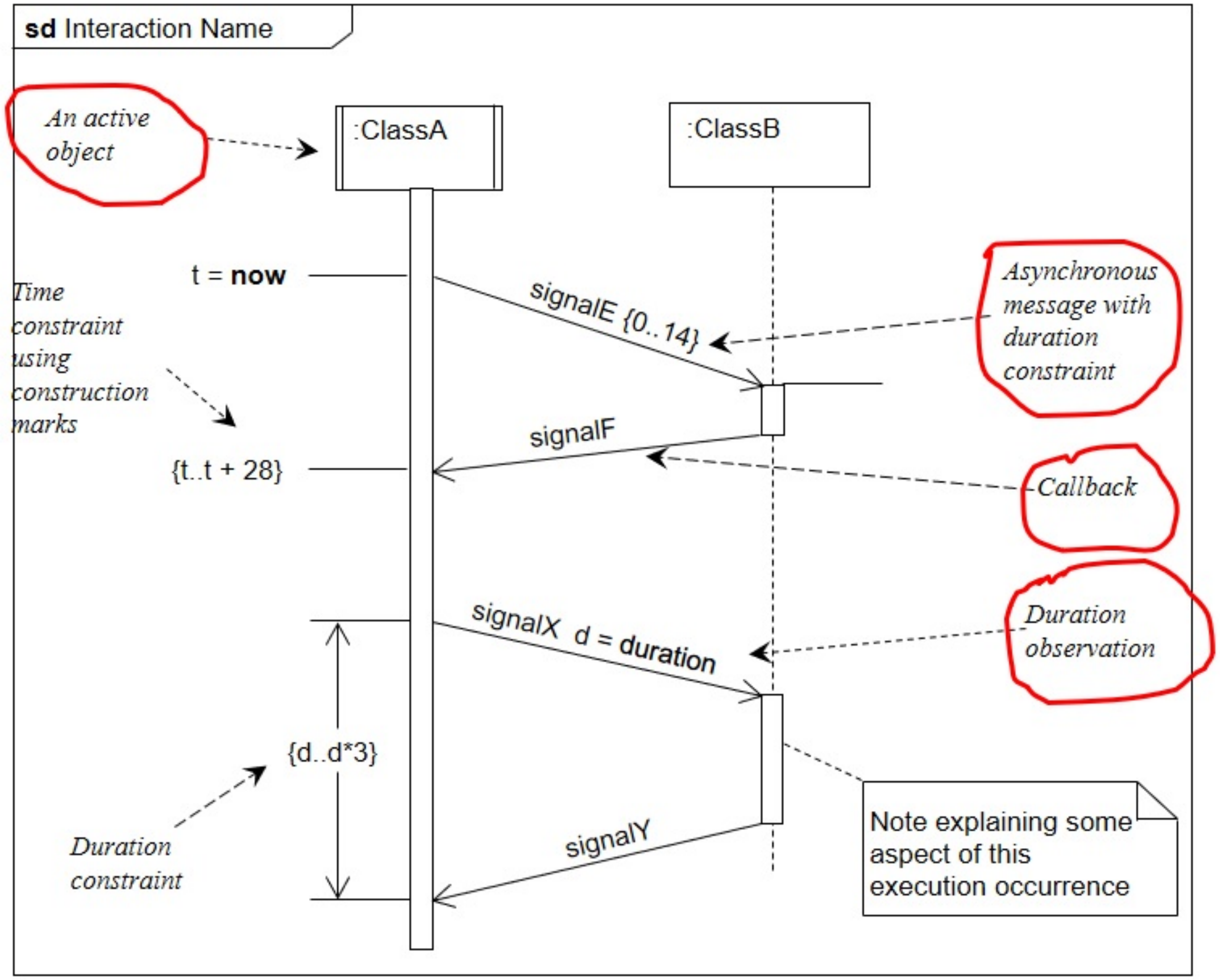


Continuations can be specified within an **alt** combined fragment to allow a link back to a referring sequence diagram.

Asynchronous Message

- An *asynchronous message*, drawn with an open arrowhead.
- Does not cause the invoking operation to halt execution while it awaits a return.
- When an asynchronous message is sent, operations in both objects may carry out processing at the same time.
- Frequently used in real-time systems when operations in different objects must execute concurrently.

Asynchronous Message (Notation)



Interaction Operators

Interaction Operator	Explanation and use
alt	Alternatives represents alternative behaviours, each choice of behaviour being shown in a separate operand. The operand whose interaction constraint is evaluated as true executes.
opt	Option describes a single choice of operand that will only execute if its interaction constraint evaluates as true.
break	Break indicates that the combined fragment is performed instead of the remainder of the enclosing interaction fragment.
par	Parallel indicates that the execution operands in the combined fragment may be merged in any sequence once the event sequence in each operand is preserved.
seq	Weak Sequencing results in the ordering of each operand being maintained but event occurrence from different operands on different lifelines may occur in any order. The order of event occurrences on common operands is the same as the order of the operands.
strict	Strict Sequencing imposes a strict sequence on execution of the operands but does not apply to nested fragments.
neg	Negative describes an operand that is invalid.
critical	Critical Region imposes a constraint on the operand that none of its event occurrences on the lifelines in the region can be interleaved.
ignore	Ignore indicates the message types, specified as parameters, that should be ignored in the interaction.
consider	Consider states which messages should be consider in the interaction. This is equivalent to stating that all others should be ignored.
assert	Assertion states that the sequence of messaging in the operand is the only valid continuation.
loop	Loop is used to indicate an operand that is repeated a number times until the interaction constraint for the loop is no longer true.

Guidelines for Sequence Diagrams

1. Decide at what level you are modelling the interaction.
2. Identify the main elements involved in the interaction. (CRC)
3. Consider the alternative scenarios that may be needed. (CRC)
4. Identify any existing interactions that have already been modelled as sequence diagrams.
5. Draw the outline structure of the diagram. (frame, lifelines)
6. Add the detailed interaction.

Guidelines for Sequence Diagrams

7. Check for consistency with linked sequence diagrams and modify as necessary.
8. Check for consistency with other UML diagrams or models (e.g. state machine diagrams).

Model Consistency

- The allocation of operations to objects must be consistent with the class diagram and the message signature must match that of the operation.
  - Can be enforced through CASE tools.
- Every sending object must have the object reference for the destination object.
  - Either an association exists between the classes or another object passes the reference to the sender.
  - This issue is key in determining association design.
  - Message pathways should be carefully analysed.

Model Consistency

- All forms of interaction diagrams used should be consistent.
- Messages on interaction diagrams must be consistent with the state machine for the participating objects.
- Implicit state changes in interactions diagrams must be consistent with those explicitly modelled in state machine.

Key points

- Object interaction can be develop from use cases.
- There are five types of messages and actions supported by UML interaction diagrams
  - Call and Return
  - Create and Destroy
  - Send

- Object interaction can be represented using an interaction sequence diagram.

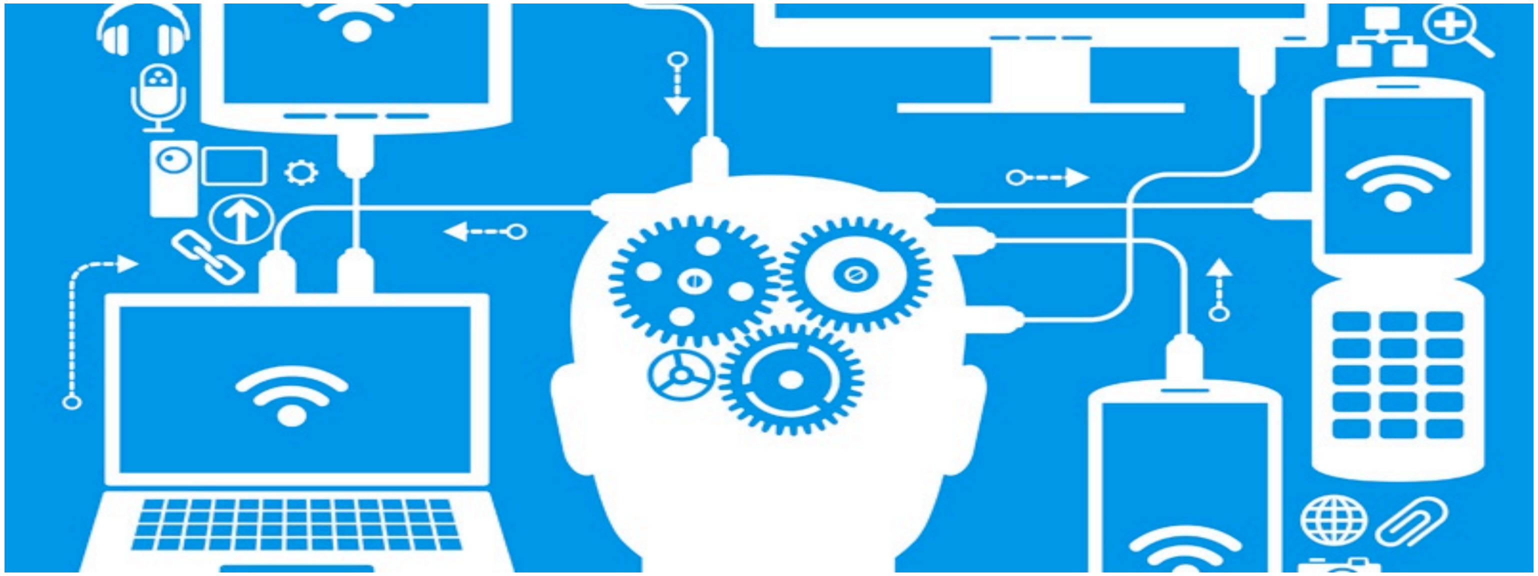
Key points

- Complex interaction can be modelled using four different techniques
  - Interaction fragments.
  - Lifelines for subsystems or groups of objects.
  - Continuations.
  - Interaction Overview Diagrams (see later lecture).
- Consistency between interaction diagrams and a class diagram can be checked via case tool.

References

- Alan Dennis, Barbara Haley Wixom & David Tegarden. 2015. Systems Analysis and Design with UML, 5th edition, Wiley.
- Simon Bennett, Steve McRobb & Ray Farmer. 2010. Object Oriented Systems Analysis and Design using UML 4th Edition, McGraw-Hill.
- UML Reference Manual (OMG, 2009)
- Bennett, Skelton and Lunn (2005)

In the next lecture..



Lecture 9: UML Interaction Diagrams – Part 2
(Communication, Interaction Overview and Timing Diagrams)