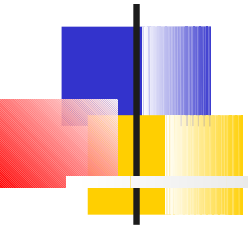
Information Systems Concepts

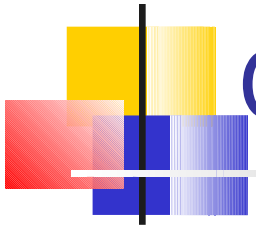
Object Interaction



Roman Kontchakov

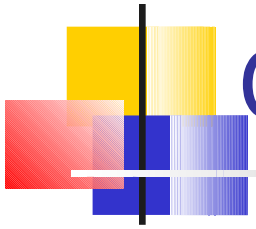
Birkbeck, University of London

Based on Chapter 9 of Bennett, McRobb and Farmer:
Object Oriented Systems Analysis and Design Using UML, (4th Edition), McGraw Hill, 2010



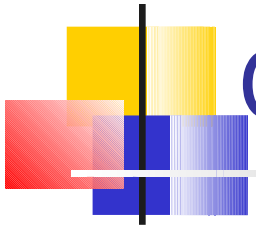
Outline

- Object Interaction and Collaboration
  - Section 9.2 (pp. 260 – 262)
- CRC Cards
  - Section 7.6 (pp. 215 – 218)
- Communication Diagrams
  - Section 7.4 (pp. 194 – 197)
  - Section 9.4 (pp. 280 – 284)
- Sequence Diagrams
  - Section 9.3 – 9.3.3 (pp. 262 – 276)
- Model Consistency
  - Section 9.7 (p. 289)



Object Interaction & Collaboration

- Message Passing
  - Objects communicate by sending messages
  - When an object sends a message to another object, an operation is invoked in the receiving object
  - The aim of modelling object interaction is to determine the most appropriate scheme of message passing between objects to support a particular user requirement

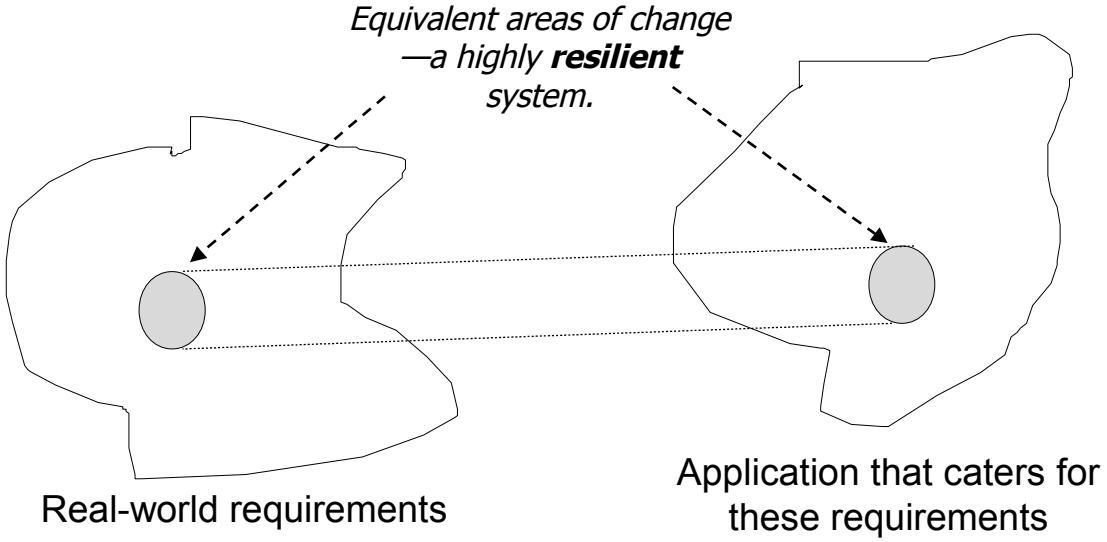


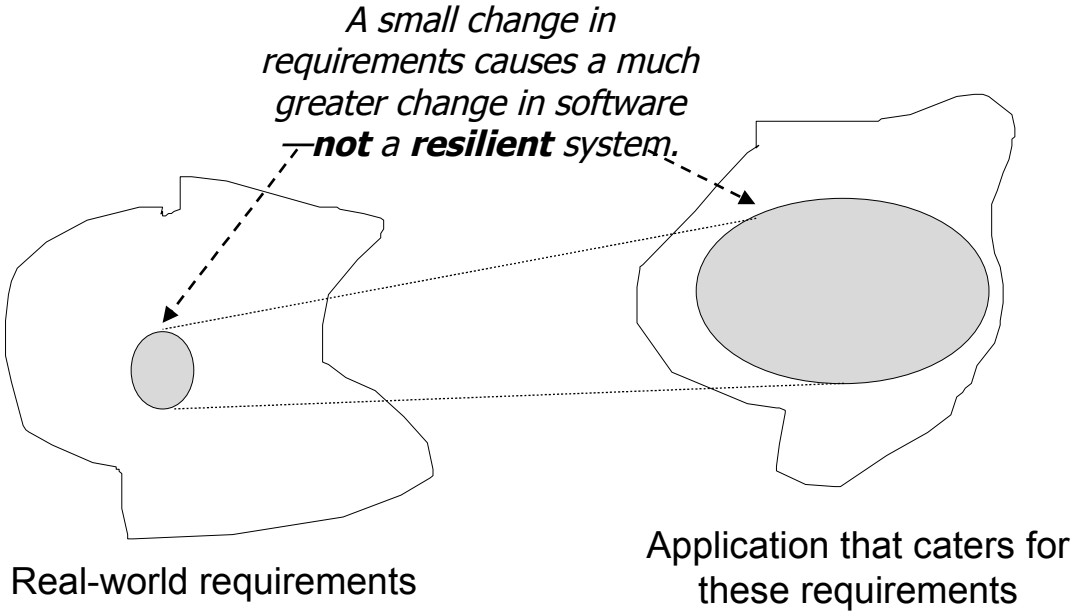
Object Interaction & Collaboration

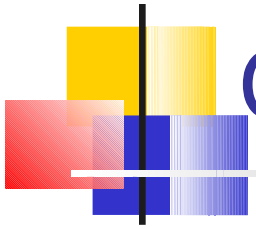
- Appropriate distribution of object responsibility improves software modularity
  - Each class tends not to be unduly complex, and as a result is easier to develop, to test and to maintain.
  - Each class is relatively small and self-contained, and as a result has a much greater potential for reuse.
  - The system is more resilient to changes in its requirements

(see next slide)

- A modular system is easier
  - to be maintained and upgraded
  - to achieve high reliability
  - to be implemented in small, manageable increments

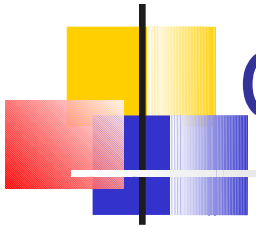

*Equivalent areas of change —a highly **resilient** system.*
Real-world requirements
Application that caters for these requirements


*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



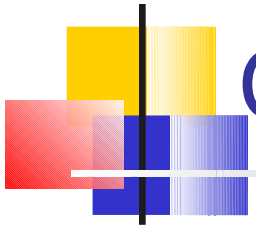
CRC Cards

- **C**lass–**R**esponsibility–**C**ollaboration (CRC) cards help to model interaction between objects
  - Brainstorm the classes
  - Allocate each class to a team members
  - For each use case, role play the interaction to distribute responsibilities among classes:
    - each object identifies the object that he/she thinks is most appropriate to take on a needed responsibility for collaboration
    - each object should be **as lazy as possible**, refusing to take on any responsibility unless persuaded by its fellow objects



CRC Cards

Class Name:	
Responsibilities	**C**ollaborations
Responsibilities of the class are listed in this section.	*Collaborations with other classes are listed here, together with a brief description of the purpose of the collaboration.*



CRC Cards: Use Case Example

Use Case: Add a new advert to a campaign

A campaign can consist of many adverts. Details of each advert are entered into the system with a target completion date and estimated cost.

Glossary:

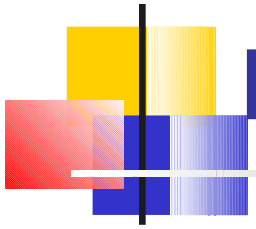
Campaign: Adverts are organized into campaigns in order to achieve a particular objective....

Client: A company or organization that wishes to obtain the services to develop and manage an advertising campaign, and design and produce adverts for the campaign

Class Name	Client
Responsibilities	**Collaborations**
Provide client information. Provide list of campaigns.	Campaign provides campaign details.

Class Name	Campaign
Responsibilities	**Collaborations**
Provide campaign information. Provide list of adverts. Add a new advert.	Advert provides advert details. Advert constructs new object.

Class Name	Advert
Responsibilities	**Collaborations**
Provide advert details. Construct adverts.	

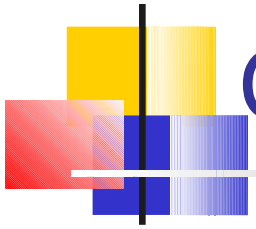


Dynamic Analysis with UML

- **Communication Diagrams**
- **Sequence Diagrams**
- Interaction Overview Diagrams
- Timing Diagrams

In UML 1.x,

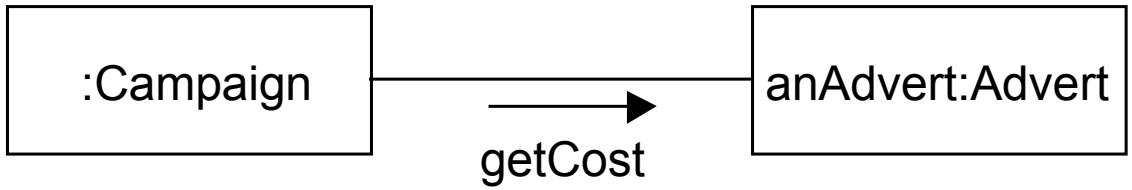
- communication diagrams are called collaboration diagrams
- interaction overview diagrams and timing diagrams do not exist



Communication Diagrams

- Communication diagrams hold the same information as sequence diagrams
- Communication diagrams show *links* between objects that participate in the collaboration
- *No time dimension*, sequence order is captured with sequence numbers

Communication Diagrams: Notation



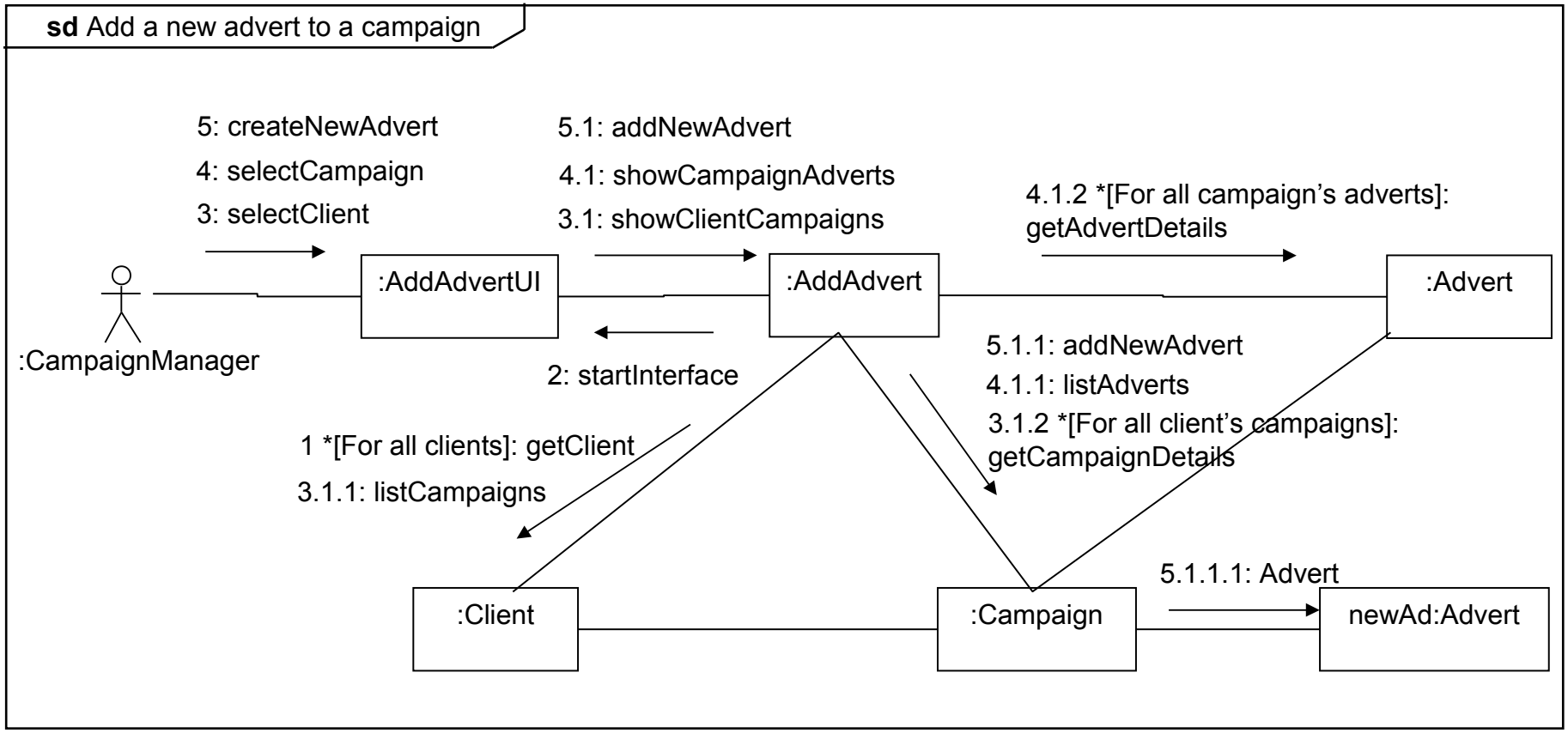
currentAdvertCost = anAdvert.getCost()

- Message order is captured with sequence numbers, which are written in a nested style to indicate the nesting of control within the interaction that is being modelled:

  e.g., message 3.1.1 is sent as part of reaction to message 3.1

Communication Diagrams: Example

Figure 9.21 on p. 282



Communication Diagrams: Message Labels

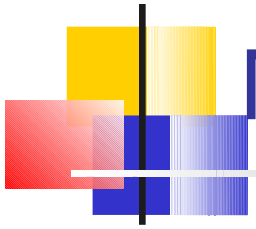
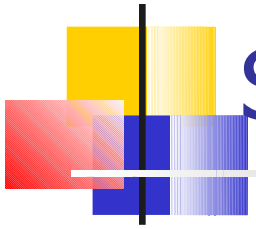


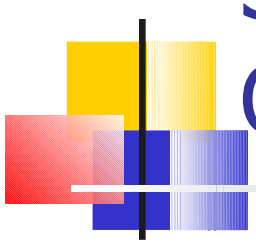
Figure 9.22 on p. 283

Type of message	**Syntax example**
Simple message.	**`4: addNewAdvert`**
Nested call with return value. *The return value is placed in the variable* `name`.	**`3.1.2: name = getName`**
Conditional message. *This message is only sent if the condition* `[balance > 0]` *is true.*	**`5 [balance > 0]: debit(amount)`**
Iteration	**`4.1 *[For all adverts]: getCost`**



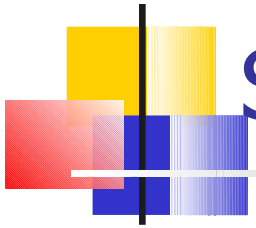
Sequence Diagrams

- A sequence diagram shows an interaction between objects arranged *in a time sequence*.
- Sequence diagrams can be drawn at *different levels of detail* and to meet different purposes at several stages in the development life cycle.
- Sequence diagrams are typically used to represent the detailed object interaction that occurs *for one use case or for one operation*.



Sequence Diagrams: Objects and Operations

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

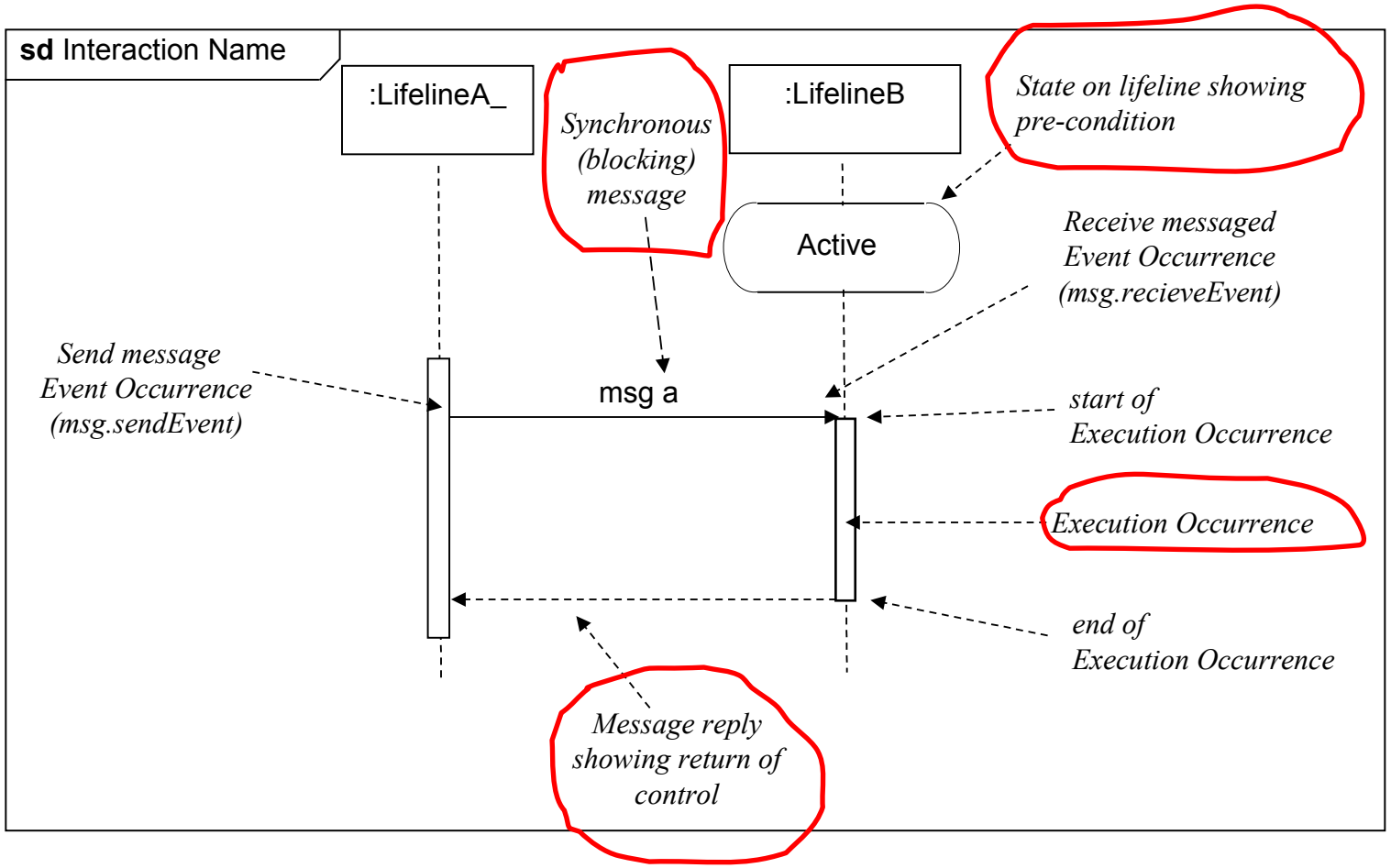


Sequence Diagrams: Messages

- **Messages** are usually shown by a solid horizontal arrow
  - A *synchronous message* or *procedural call* is shown with a full arrowhead, causes the invoking operation to suspend execution until the focus of control has been returned to it.
  - It is optional to show *reply messages* (with dashed arrows) because it can be assumed that control is returned to the originating object at the end of the activation in a destination object (except for *asynchronous messages*).
  - A *reflexive message* that an object sends to itself is shown by a message arrow that starts and finishes at the same object lifeline.

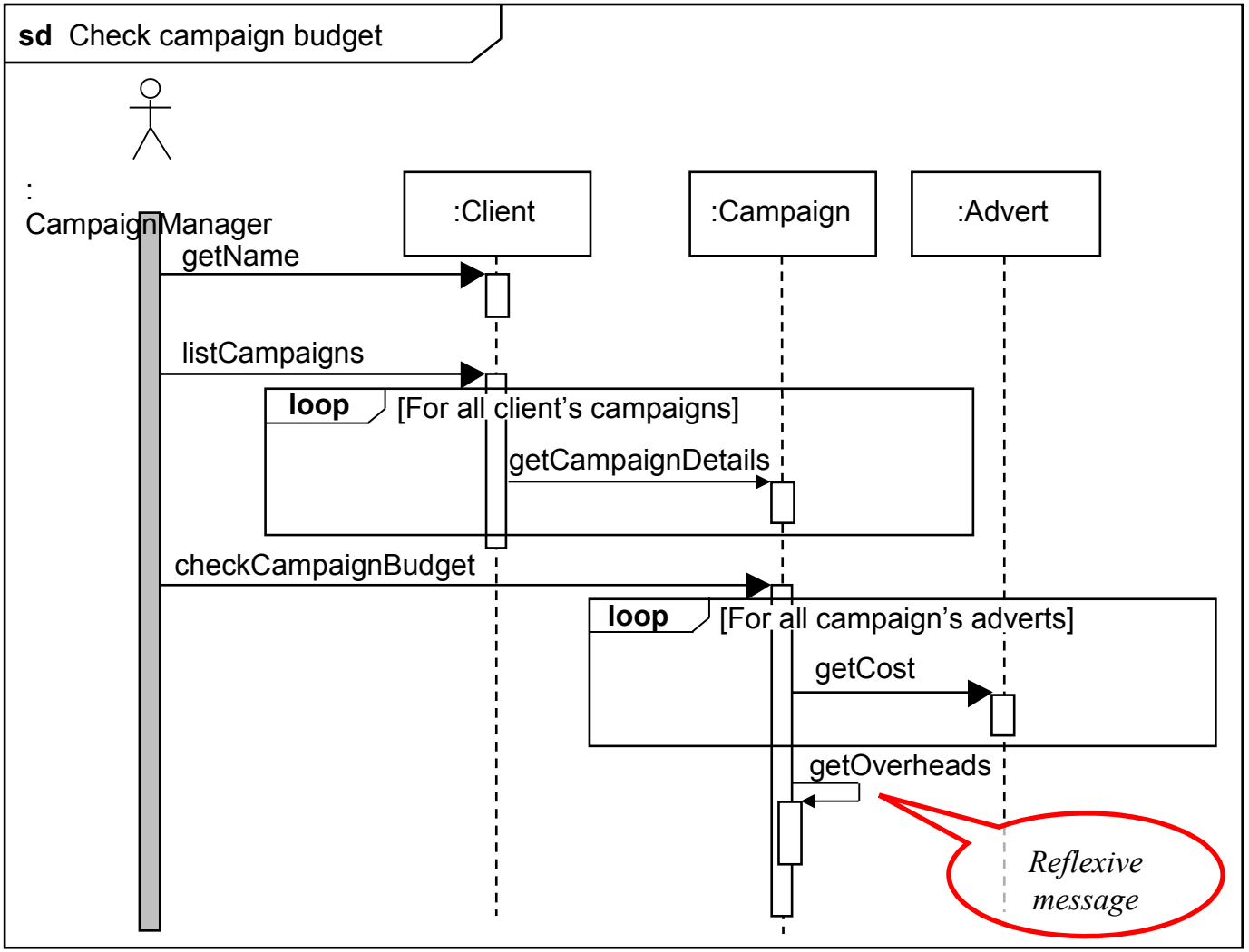
Sequence Diagrams: Example

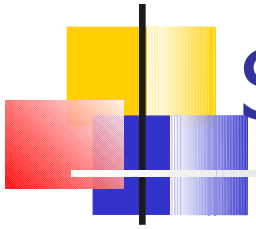
Figure 9.4 on p. 265



Sequence Diagrams: Example

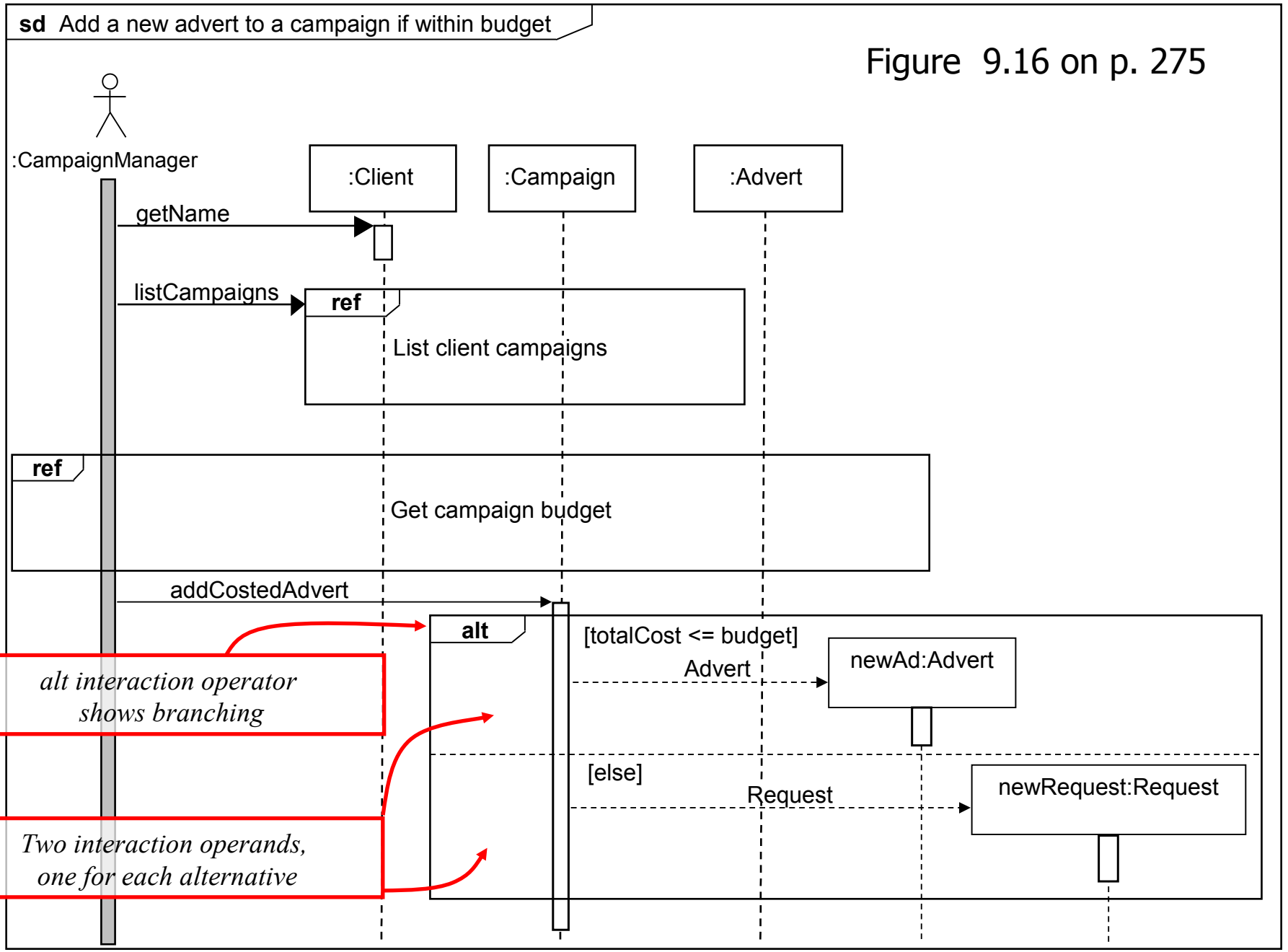
Figure 9.7
on p. 268





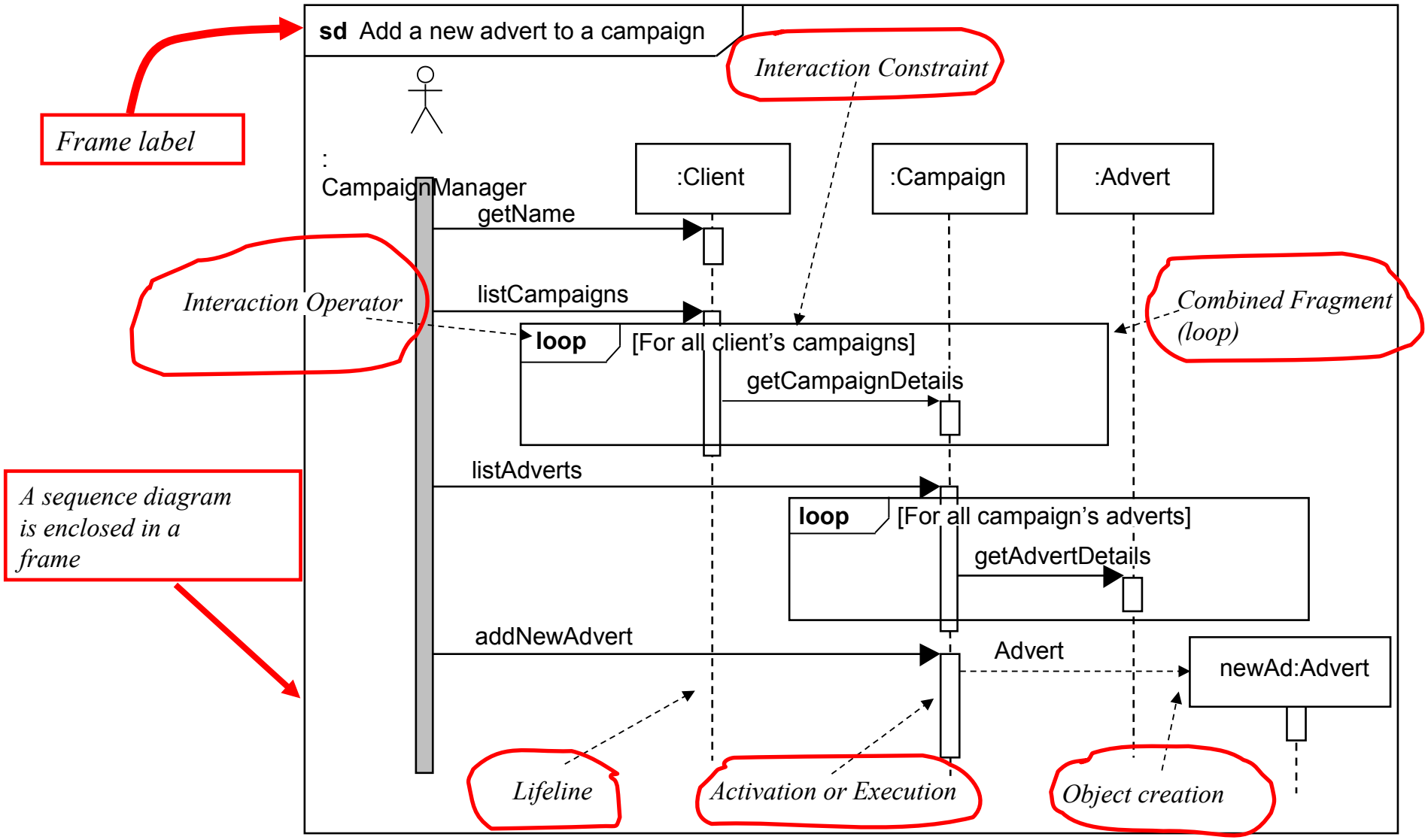
Sequence Diagrams: Flow Control

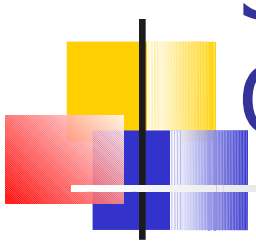
- **Sequencing** is shown by arranging messages vertically in the order of their occurrences
  - The vertical dimension represents time.
- **Selection (branching)** is shown by a combined fragment rectangle with the interaction operator 'alt' (a short form of alternatives)
  - The alt combined fragment has two (or more) compartments known as operands. Each operand corresponds to one of the alternatives in the combined fragment and each operand should have an interaction constraint to indicate under what conditions it executes.
- **Iteration (looping)** is shown by a combined fragment rectangle with the interaction operator 'loop'
  - The interaction in the loop combined fragment repeats as long as the *guard condition* in the interaction constraint evaluates as true.


sd Add a new advert to a campaign if within budget
Figure 9.16 on p. 275
:CampaignManager
:Client
:Campaign
:Advert
getName
listCampaigns
ref
List client campaigns
ref
Get campaign budget
addCostedAdvert
alt
[totalCost <= budget]
Advert
newAd:Advert
[else]
Request
newRequest:Request
alt interaction operator shows branching
Two interaction operands, one for each alternative

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 evaluted as true executes.
opt	Option describes a single choice of operand that will only execute if its interaction contraint 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.

Figure 9.3 on p. 263



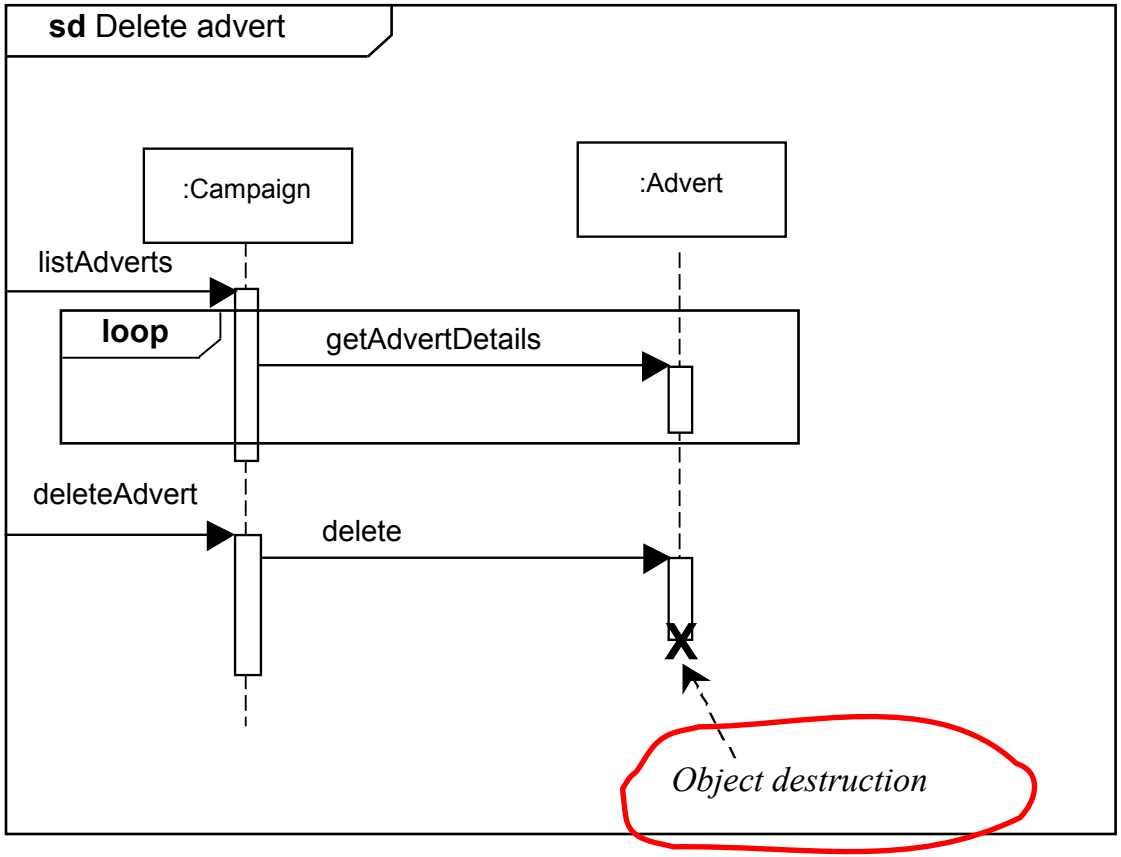


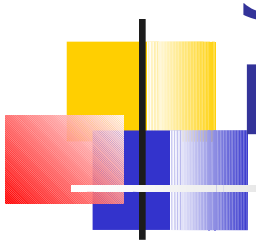
Sequence Diagrams: Object Creation and Destruction

- **Object creation** is shown with the construction message (dashed arrow) going to the object symbol.
- **Object destruction** is indicated by a large X on the lifeline on the destruction point.

Sequence Diagrams: Example

Figure 9.6 on p. 267



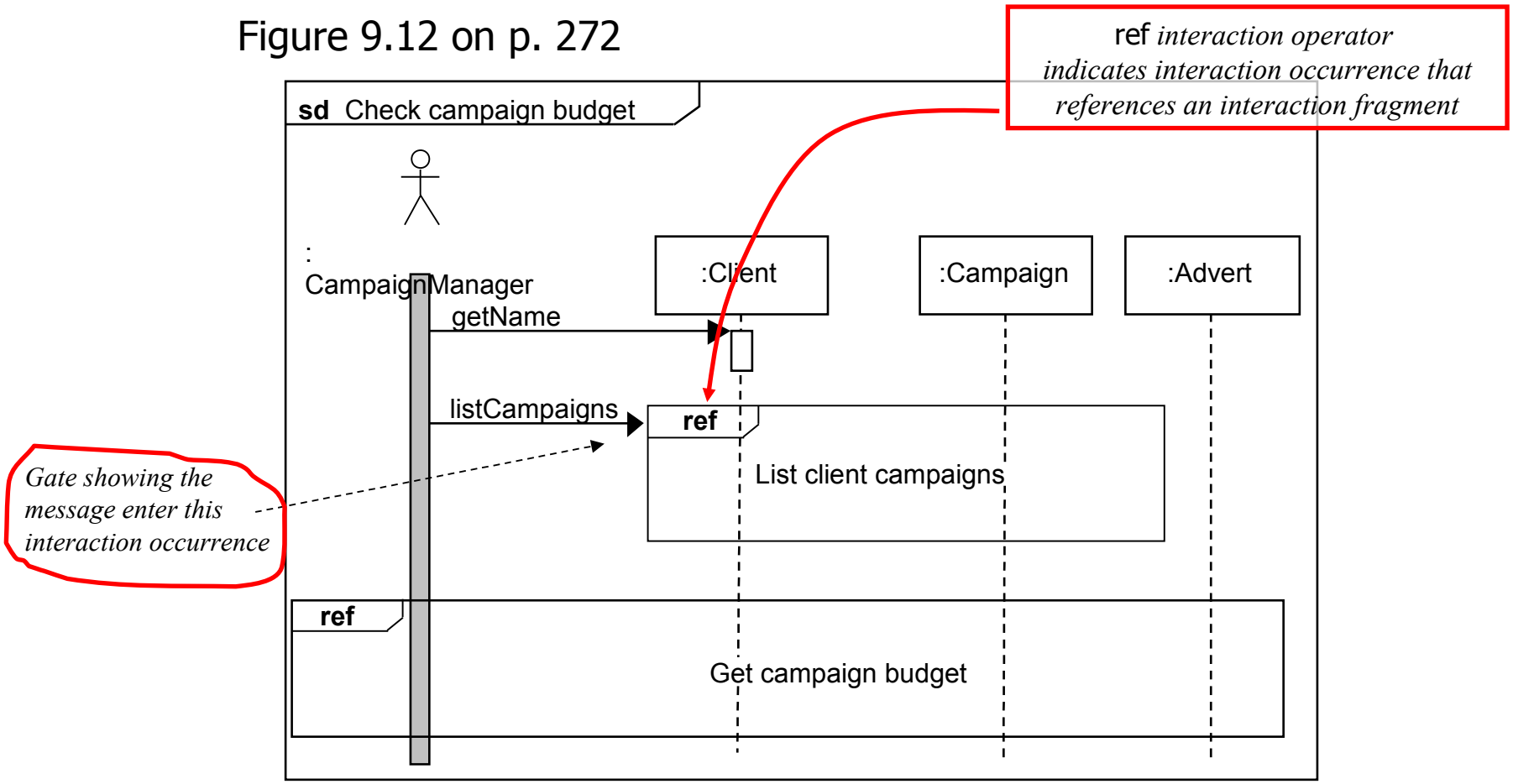


Sequence Diagrams: Handling Complexity

- Interaction occurrences and Interaction fragments
- Lifelines for subsystems or groups of objects
- Continuations
- Interaction Overview Diagrams

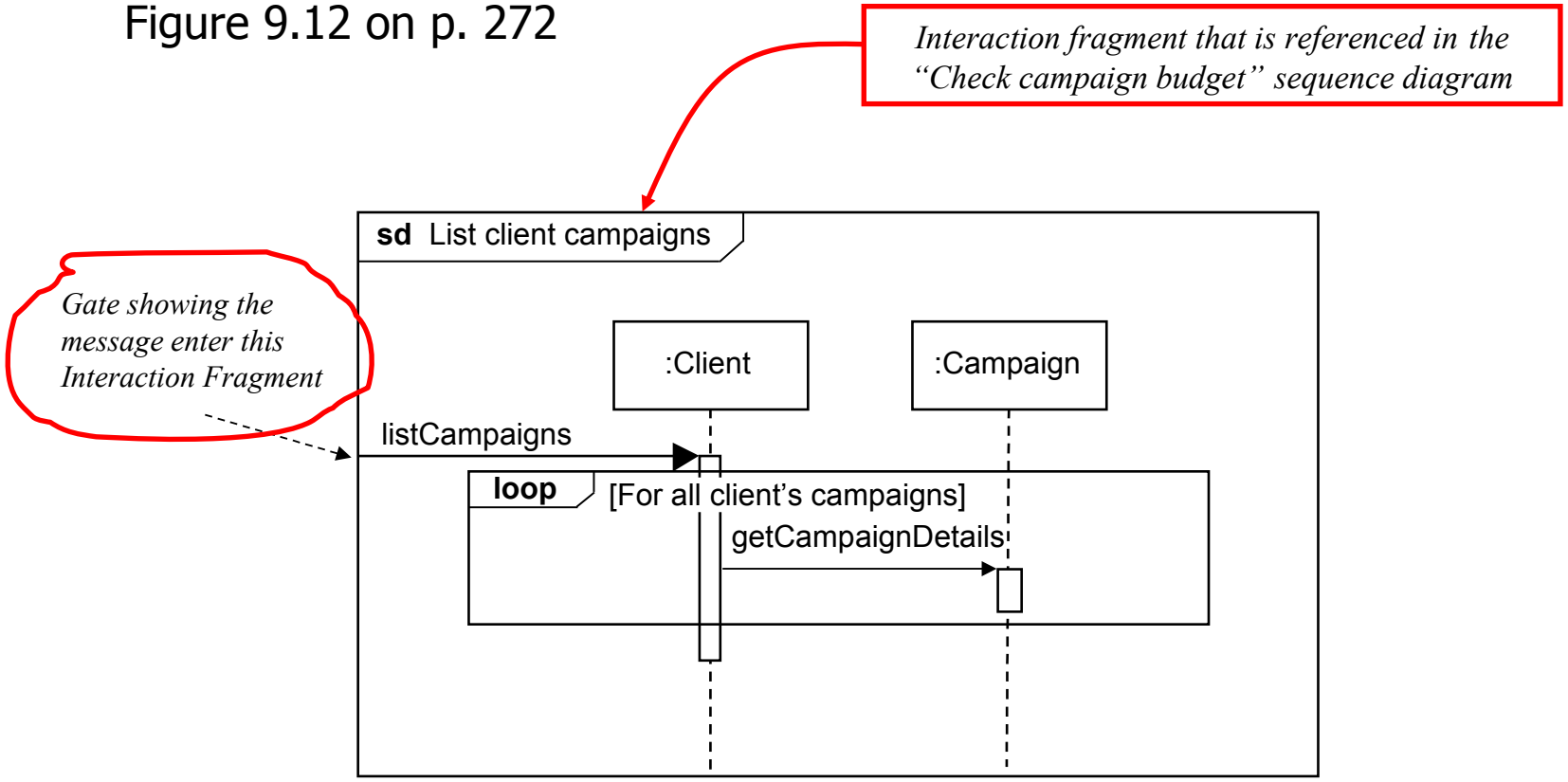
Sequence Diagrams: Example

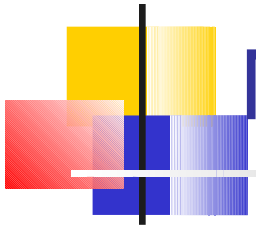
Figure 9.12 on p. 272



Sequence Diagrams: Example (cont.)

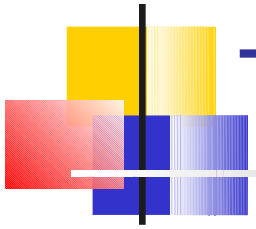
Figure 9.12 on p. 272





Model Consistency

- The communication/sequence diagrams should be mutually consistent with the class diagrams
  - 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
    - Message pathways should be carefully analysed.
    - This issue is key in determining association design



Take Home Messages

- Object Interaction and Collaboration
- CRC Cards
- Communication Diagrams
- Sequence Diagrams
- Model Consistency