Object Interaction

Chapter 9

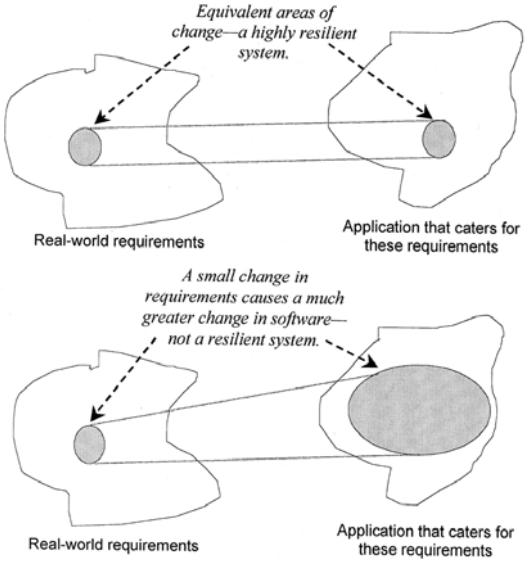
In this Lecture you will Learn:

- How to develop object collaboration from use cases
- How to model object collaboration using an interaction sequence diagram
- How to model object collaboration using an interaction collaboration diagram
- How to cross-check between interaction diagrams and a class diagram

Resilience of Design (設計彈性)

- An appropriate distribution of responsibility among classes has the important side-effect of producing a system that is more resilient to changes in its requirements.
- When the users' requirements for a system change it is reasonable to expect that the application will need some modification, but ideally the change in the application should be of no greater magnitude that the change in the requirements.
- An application that is resilient in this sense costs less to maintain and to extend than one that is not.

Resilience of Design

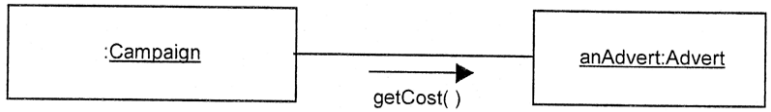


Interaction & Collaboration

- The structure of Instances playing roles in a behaviour and their relationships is called a **Collaboration**.
- An **Interaction** is defined in the context of a Collaboration. It specifies the communication patterns between the roles in the Collaboration.
- More precisely, it contains a set of partially ordered **Messages**, each specifying one communication, e.g. what Signal to be sent or what Operation to be invoked, as well as the roles played by the sender and the receiver, respectively.

Object Messaging

- Message passing is a metaphor used to describe object interaction.
- Objects communicate by sending messages
- Sending the message getCost() to an Advert object, might use the following syntax
  - advertCost = anAdvert.getCost()



Sequence Diagram

- Show an interaction between 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

Sequence Diagram Component

- **Vertical Dimension** represents time and all objects involved in the interaction are spread horizontally across the diagram.
- **Time** normally proceeds down the page. An **Object Lifeline** represents the existence of an object during an interaction represented in a sequence diagram.
- **Object** is represented by a vertical dashed line, called a lifeline, with an object symbol at the top.
- **Message** is shown by a solid horizontal arrow from one lifeline to another and is labeled with the message name. **Message Name** may optionally be preceded by a sequence number that represents the sequence in which the messages are sent, but this is not usually necessary on a sequence diagram since the message sequence is already conveyed by their relative positions along the time axis.

Sequence Diagram

- UML uses the general term **Stimulus** to describe an interaction between two objects that conveys information with an expectation of some action.
- A **Message** specifies the sender and receiver objects and the action of a stimulus.
- A message may correspond to calling an operation or raising a signal.
- An **Event** is the specification of an occurrence of significance and may for instance be the receipt of a message or a signal by an object.

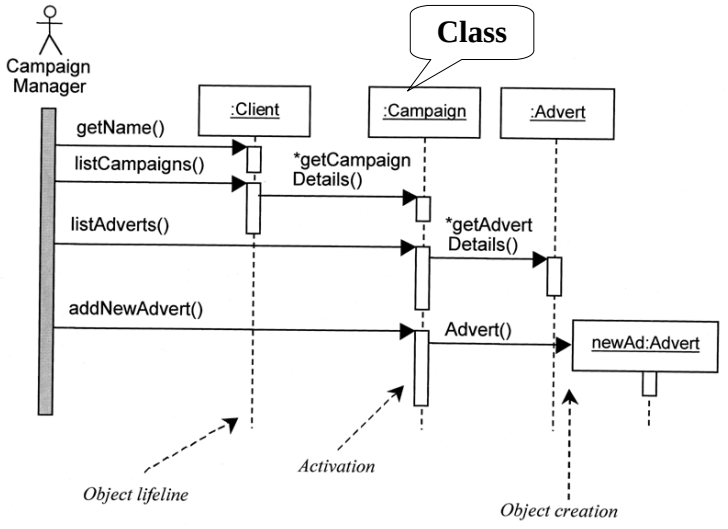
Sequence Diagram – Message

- In UML a signal is an asynchronous communication that may have parameters.
- From a pragmatic modeling perspective we need to distinguish between synchronous and asynchronous messages.
- When a message is sent to an object, it invokes an operation of that object.
- Once a message is received, the operation that has been invoked begins to execute.
- The message name is usually the same as the particular operation that is being invoked.

Sequence Diagram – Activation

- The period of time during which an operation executes is known as an **Activation**, and is shown on the sequence diagram by a rectangular block laid along the lifeline.
- The activation period of an operation includes any delay while the operation waits for a response from another operation that it has itself invoked as part of its execution.

Sequence Diagram

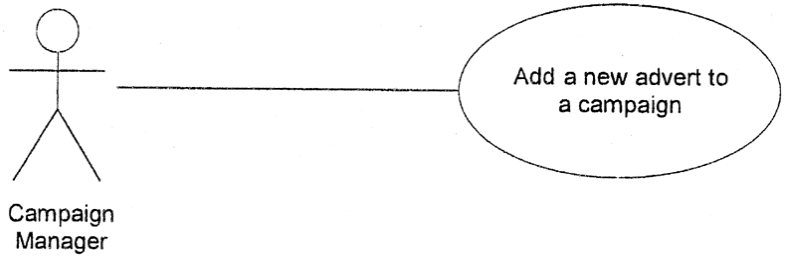


Example Explanation

- This sequence diagram drawn without boundary or control objects.
- The getName() message is first message received by the Client and is intended to correspond the Campaign Manager requesting the name of the selected Client.
- The Client object then receives a listCampaigns() message and a second period of operation activation begins.
- This is shown by the tall thin rectangle that begins at the message arrowhead

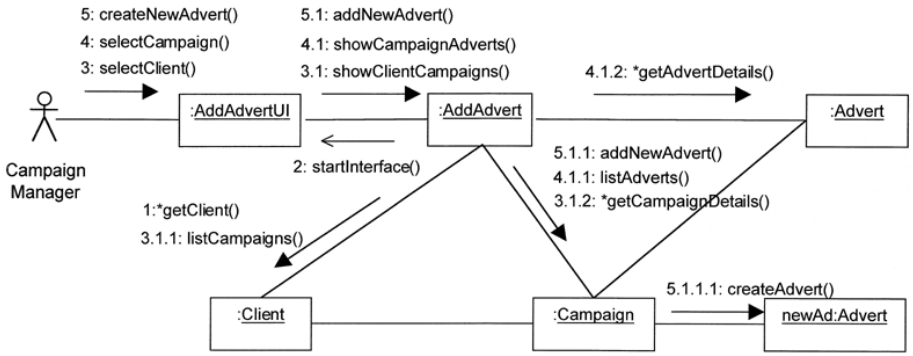
Boundary & Control Classes

- Consider the User Case for the "Add a new advert to a campaign"



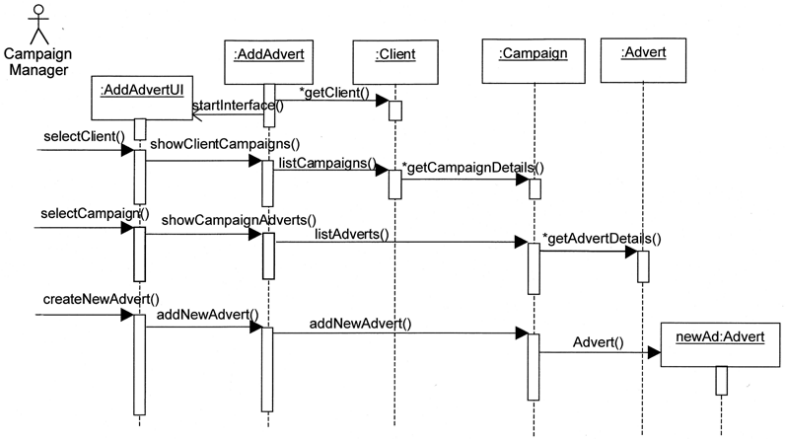
Boundary & Control Classes

- The collaboration for the use case "Add a new advert to a campaign"



Boundary & Control Classes

- Sequence Diagram for the use case "Add a new advert to a campaign" with boundary and control class.



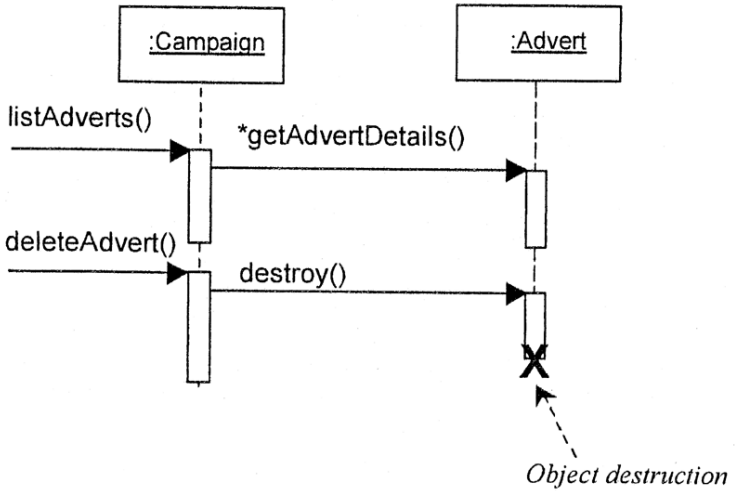
Boundary & Control Classes

- Most use cases imply at least one boundary object that manages the dialogue between the actor and the system – in the sequence diagram it is **:AddAdvertUI**
- The control object is **:AddAdvert** and this manages the overall object communication.

Object Destruction

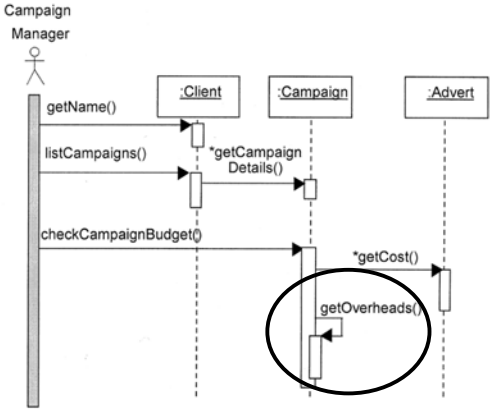
- Objects may created or destroyed at different stages during an interaction.
- On a sequence diagram the destruction of an object is indicated by a large X on the lifeline at the point in the interaction when the object destroyed.
- An object may either be destroyed when it received a message or it may self-destruct at the end of an activation if this is required by the particular operation that is being executed.

Object Destruction



Reflexive Messages

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



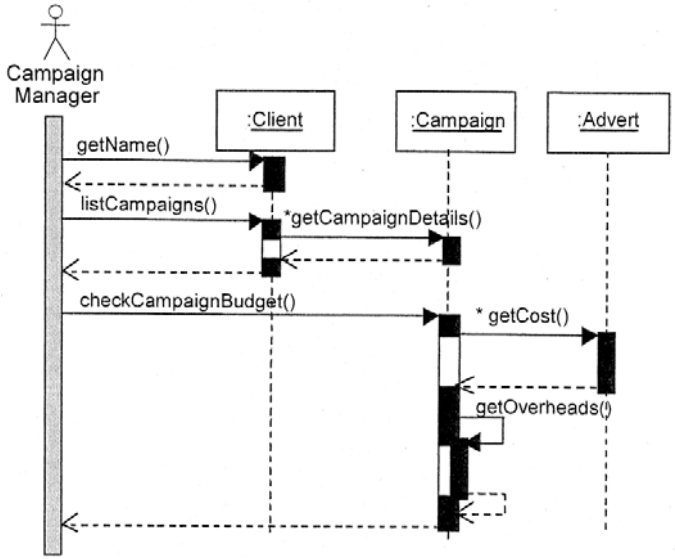
Focus of Control

- The **Focus of Control** indicates which operation is executing at a particular stage in an interaction represented in a sequence diagrams.
- 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

Return

- A **Return** is a return of control to the object that originated the message that began the activation.
- This is not a new message, but is only the conclusion of the invocation of an operation.
- Returns 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 at the end of the activation

Example: Focus of Control and Return



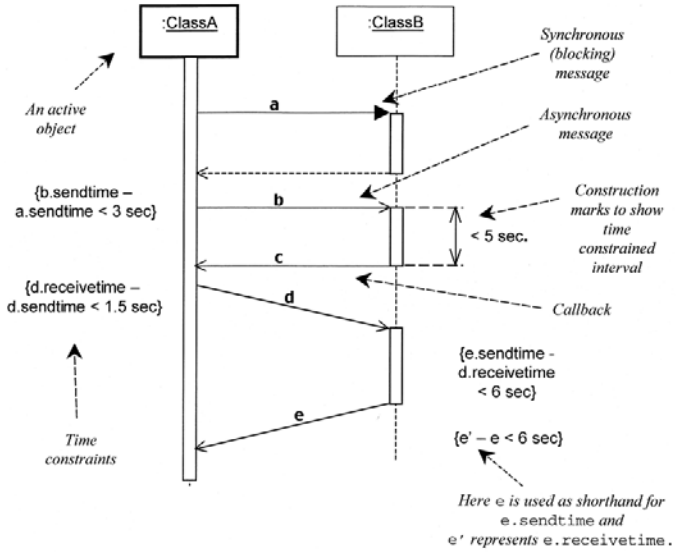
Message Types – Synchronous

- A **Synchronous Message** also called **Procedural Call**,
- A synchronous message is a blocking call
- The sending operation waits for the receiving object to complete execution before resuming
- It causes the invoking operation to suspend execution until the focus of control has been returned to it
- Shown with a full arrowhead.

Message Types – Asynchronous

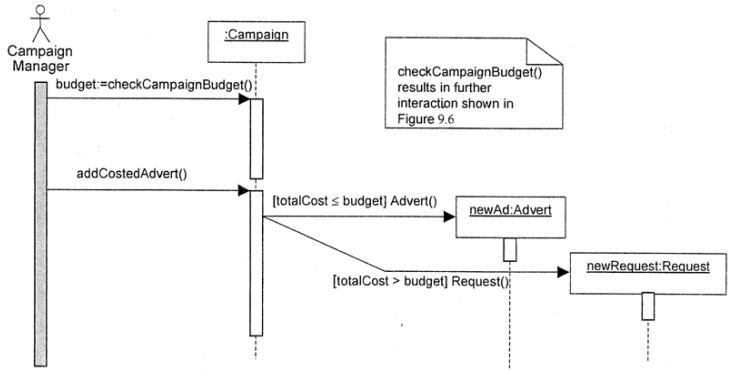
- An **Asynchronous Message** is not a blocking call.
- Does not cause the invoking operation to halt execution while it awaits a return
- The operation in the sending object continues to execute after sending the asynchronous message that invokes an operation in the destination object, which executes concurrently.
- Drawn with an open arrowhead

Example of Different Message Types



Message Branching

- Some interactions have two or more alternative execution pathways.
- Each reflects a branch in the possible sequence of events for the use case it represents.

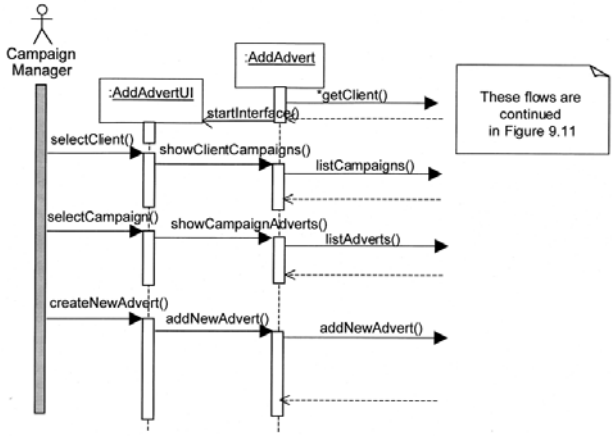


Handling Complexity

- Complex interactions may be represented by managed by one of the following:
  - Complex diagrams can be split into two or more smaller diagrams suitably annotated
  - Alternatively a group of objects can be represented by a single lifeline, and interaction among these objects is shown on a different diagram
  - Representing the interaction on several linked sequence diagrams
  - Using object grouping to simplify the representation by hiding that part of the interaction within the group.
  - Not showing all the messages explicitly but annotating the diagram with a note to make this clear.

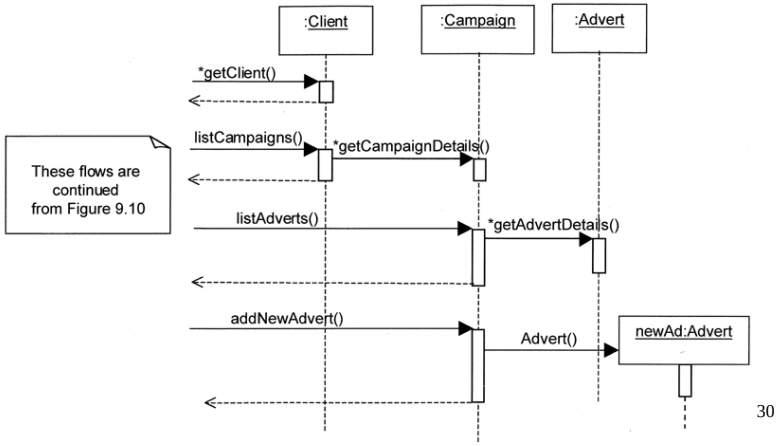
Handling Complexity – Diagram Separation

- First part of interaction for use case Add a advert to a campaign



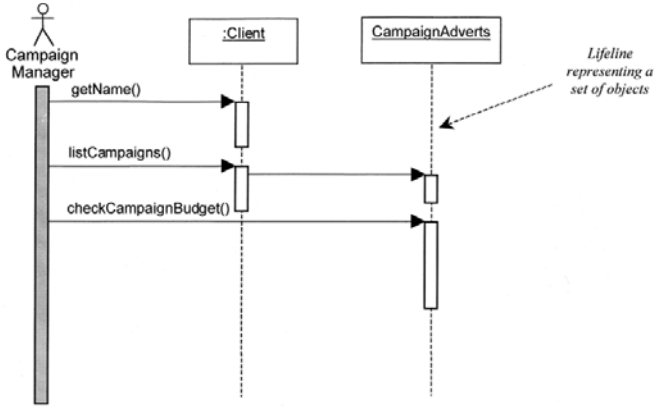
Handling Complexity – Diagram Separation

- Second part of interaction for use case Add a advert to a campaign



Handling Complexity – Object Grouping

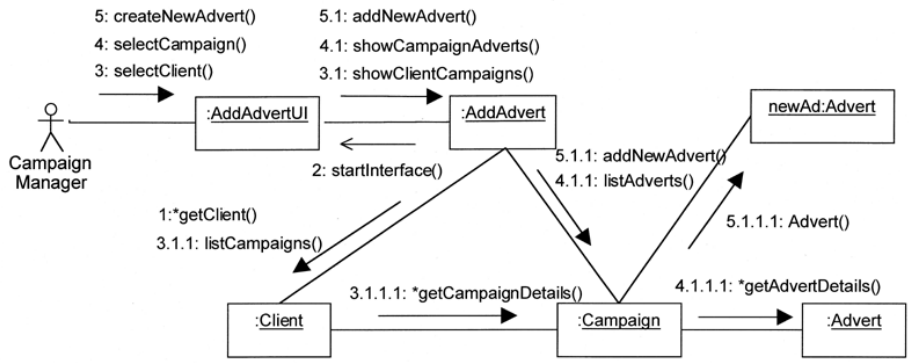
- Alternative Sequence diagram representing a group of objects by a single lifeline for use case Check campaign budget.



Collaboration Diagrams

- Hold the same information as sequence diagrams
- Show links between objects that participate in the collaboration
- No time dimension, sequence is captured with sequence numbers
- Sequence numbers are written in a nested style (e.g. 3.1 & 3.1.1) in collaboration diagram to represent nested procedural calls.

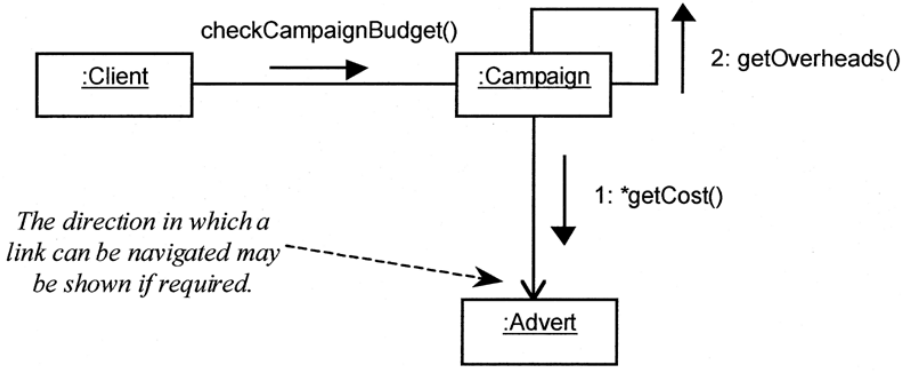
Collaboration Diagrams



Various Type of Message Labels

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.*	`[balance > 0] 5: debit(amount)`
Synchronization with other threads. *Message* `4: playVideo()` *is invoked only once the two concurrent messages* `3.1a and 3.1b` *are completed.*	`3.1a, 3.1b / 4:playVideo()`

Navigating Links



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

- If both sequence and collaboration diagrams are prepared they should be consistent
- Messages on interaction diagrams must be consistent with the statecharts for the participating objects
- Implicit state changes in interaction diagrams must be consistent with those explicitly modelled in statecharts

Class Exercise

- What consistency checks should be applied to interaction diagrams?

Sequence Diagrams vs. Collaboration Diagrams

- Sequence Diagrams have a time dimension (normally vertically down the page) while Collaboration Diagrams do not.
- Collaborations show the links between objects, which are not shown on sequence diagrams.

Sequence Diagrams vs. Collaboration Diagrams

- The essential part of the message label in an interaction sequence diagram is the signature of the operation being invoked. If the name of the operation is sufficient to uniquely identify it then this may be sufficient though CASE tools may report a consistency error if the message label does not match the signature of the operation in the class.
- In addition each message on a collaboration diagram also requires a sequence expression to show the sequence of the messages. This is normally not necessary on a sequence diagram as the sequence is represented by the order of the messages along the time dimension (normally down the page).

Class Exercise

- List two specific features of bad object-oriented modeling that are discouraged by the use of collaboration diagrams.

Class Exercise

- What are the benefits of keeping all classes reasonably small and self-contained?