UML - INTERACTION DIAGRAMS

Interaction Diagram is used in UML to establish communication between objects. It does not manipulate the data associated with the particular communication path.

Interaction diagrams mostly focus on message passing and how these messages make up one functionality of a system. Interaction diagrams are designed to display how the objects will realize the particular requirements of a system. The critical component in an interaction diagram is lifeline and messages.

The details of interaction can be shown using several notations such as sequence diagram, timing diagram and communication/collaboration diagram.

Following are the different types of interaction diagrams defined in UML:

- Sequence diagram
- Collaboration diagram
- Timing diagram

The purpose of a *sequence diagram* in UML is to visualize the sequence of a message flow in the system. The Sequence Diagram in Software Engineering shows the interaction between two lifelines as a time-ordered sequence of events.

The *Collaboration Diagram* in UML is also called a communication diagram. The purpose of a collaboration diagram is to emphasize structural aspects of a system, i.e., how various lifelines in the system connect.

Timing diagrams focus on the instance at which a message is sent from one object to another object.

Purpose of an Interaction Diagram

Interaction diagrams help you to visualize the interactive behavior of a system. Interaction diagrams are used to represent how one or more objects in the system connect and communicate with each other.

Interaction diagrams focus on the dynamic behavior of a system. An interaction diagram provides us the context of an interaction between one or more lifelines in the system.

In UML, the interaction diagrams are used for the following purposes:

- Observe the dynamic behavior of a system.
- Visualize the communication and sequence of message passing in the system.
- Represents the structural aspects of various objects in the system.
- Represents the ordered sequence of interactions within a system.
- Provides the means of visualizing the real time data via UML.
- Explain the architecture of an object-oriented or a distributed system.

Important terminology

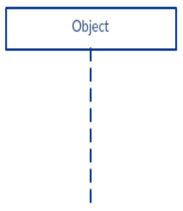
An interaction diagram contains lifelines, messages, operators, state invariants and constraints.

Lifeline

A lifeline represents a single participant in an interaction. It describes how an instance of a specific classifier participates in the interaction.

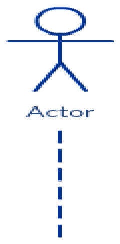
A lifeline represents a role that an instance of the classifier may play in the interaction.

Lifeline Notation

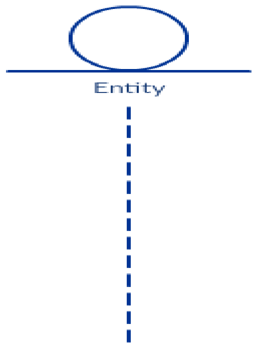


A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

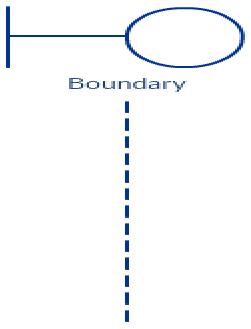
A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.



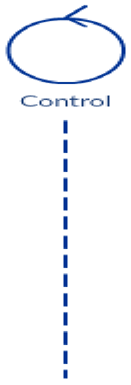
A lifeline with an entity element represents system data. For example, in a customer service application, the Customer entity would manage all data related to a customer.



A lifeline with a boundary element indicates a system boundary/ software element in a system; for example, user interface screens, database gateways or menus that users interact with, are boundaries.



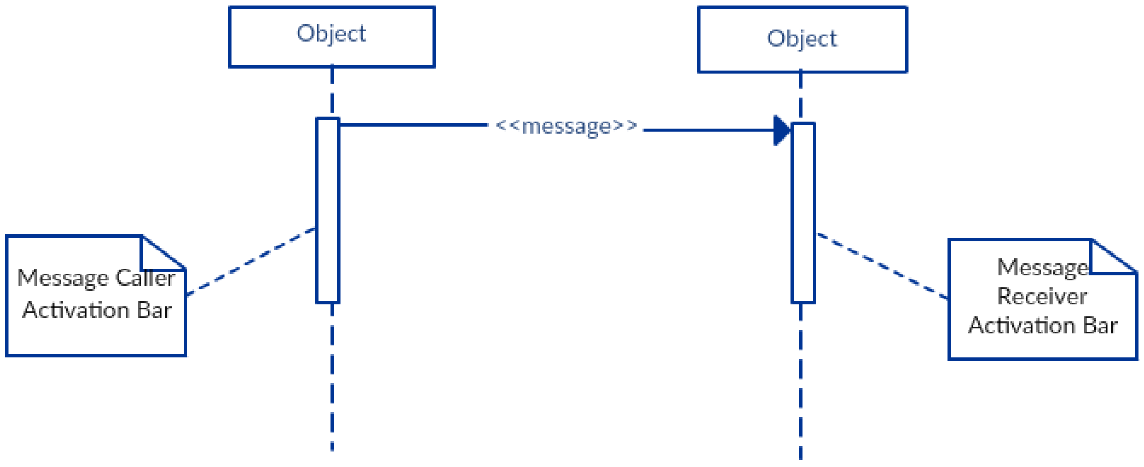
And a lifeline with a control element indicates a controlling entity or manager. It organizes and schedules the interactions between the boundaries and entities and serves as the mediator between them.



Activation Bars

Activation bar is the box placed on the lifeline. It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active.

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.


Object
Object
<<message>>
Message Caller Activation Bar
Message Receiver Activation Bar

Messages

A message is a specific type of communication between two lifelines in an interaction. A message involves following activities,

1. A call message which is used to call an operation.
2. A message to create an instance.
3. A message to destroy an instance.
4. For sending a signal.

When a lifeline receives a call message, it acts as a request to invoke an operation that has a similar signature as specified in the message. When a lifeline is executing a message, it has a focus of control. As the interaction progresses over time, the focus of control moves between various lifelines. This movement is called a flow of control.

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram. A message can flow in any direction; from left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received.

The message arrow comes with a description, which is known as a message signature, on it. The format for this message signature is below. All parts except the message_name are optional.

attribute = message_name (arguments): return_type

Following are the messages used in a System Interaction Diagram:

Message Name	**Meaning**
Synchronous message	The sender of a message keeps waiting for the receiver to return control from the message execution.
Asynchronous message	The sender does not wait for a return from the receiver; instead, it continues the execution of a next message.
Return message	The receiver of an earlier message returns the focus of control to the sender.
Object creation	The sender creates an instance of a classifier.

Object destruction	The sender destroys the created instance.
Found message	The sender of the message is outside the scope of interaction.
Lost message	The message never reaches the destination, and it is lost in the interaction.

State invariants and constraints

When an instance or a lifeline receives a message, it can cause it to change the state.

A state is a condition or a situation during a lifetime of an object at which it satisfies some constraint, performs some operations, and waits for some event.

In interaction diagram, not all messages cause to change the state of an instance. Some messages do not have the values of some attributes. It has no side effects on the state of an object.

Operator

An operator specifies an operation on how the operands are going to be executed.

The operators in UML support operations on data in the form of branching as well as iteration.

Various operators can be used to ensure the use of iteration and branching in the UML model.

Following are the operators used in an interaction diagram:

Operator	**Name**	**Meaning**
Opt	Option	An operand is executed if the condition is true. e.g., If else
Alt	Alternative	The operand, whose condition is true, is executed. e.g., switch
Loop	Loop	It is used to loop an instruction for a specified period.
Break	Break	It breaks the loop if a condition is true or false, and the next instruction is executed.
Ref	Reference	It is used to refer to another interaction.
Par	Parallel	All operands are executed in parallel.

Iteration

In an interaction diagram, we can also show iteration using an iteration expression. An iteration expression consists of an iteration specifier and an optional iteration clause. There is no pre-specified syntax for UML iteration.

In iteration to show that messages are being sent in parallel, parallel iteration specifier is used. A parallel iteration specifier is denoted by *//.

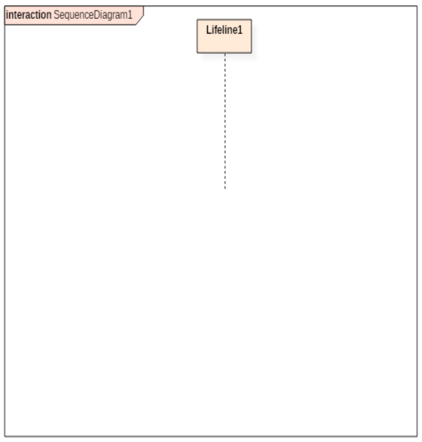
Iteration in UML is achieved by using the loop operator.

Branching

In an interaction diagram, we can represent branching by adding guard conditions to the messages. Guard conditions are used to check if a message can be sent forward or not. A message is sent forward only when its guard condition is true. A message can have multiple guard conditions, or multiple messages can have the same guard condition.

Branching in UML is achieved with the help of alt and opt, operators.

The basic notation of interaction is a rectangle with a pentagon in the upper left corner of a rectangular box.

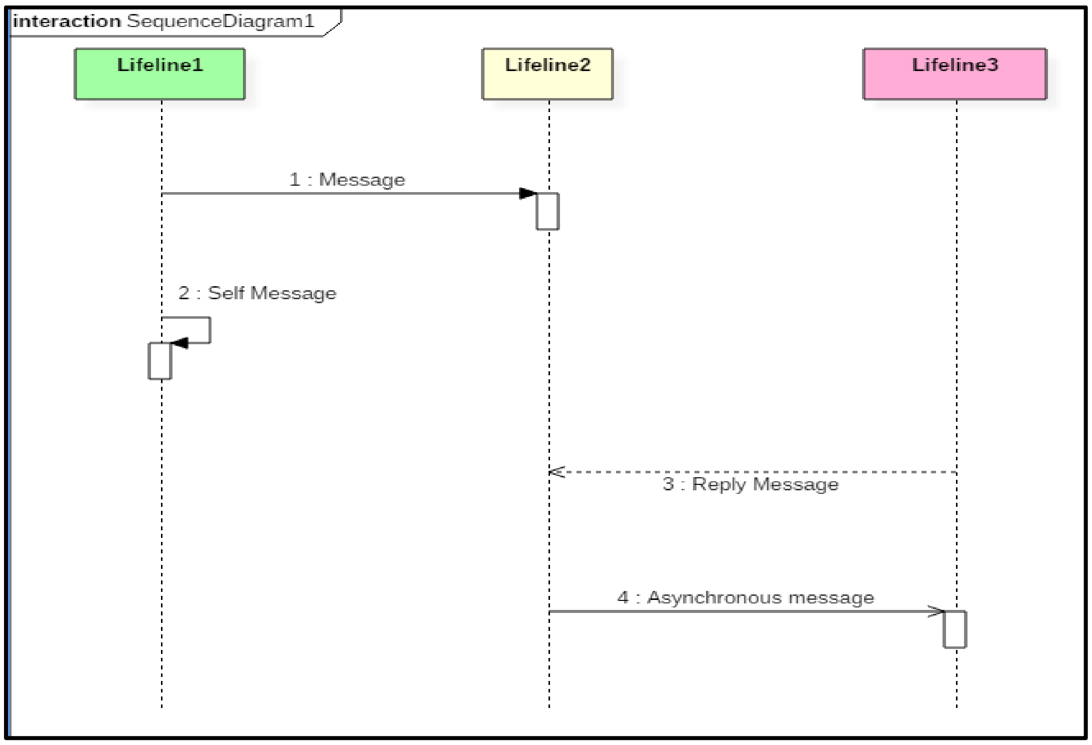


Notation of an Interaction Diagram

SEQUENCE DIAGRAM

A Sequence Diagram simply depicts interaction between objects in a sequential order. The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system. The sequence diagram shows the interaction between two lifelines as a time-ordered sequence of events.

- A sequence diagram shows an implementation of a scenario in the system. Lifelines in the system take part during the execution of a system.
- In a sequence diagram, a lifeline is represented by a vertical bar.
- In a sequence diagram, different types of messages and operators are used which are described above.
- In a sequence diagram, iteration and branching are also used.

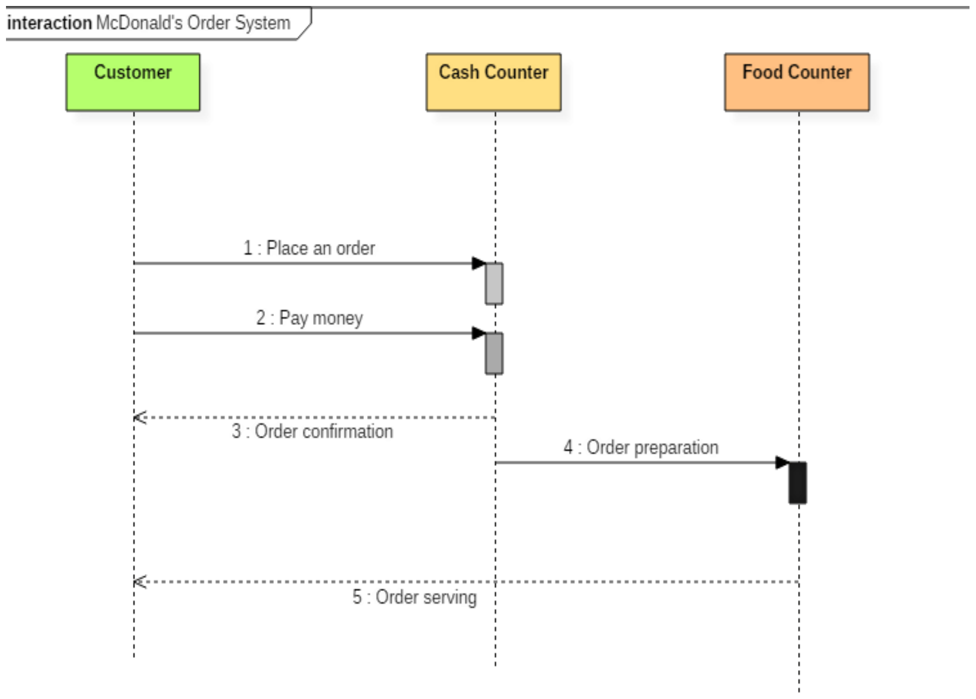


Notations in Sequence Diagram

The above sequence diagram contains lifeline notations and notations of various messages used in a sequence diagram such as create, reply, asynchronous message, etc.

Sequence diagram example

The following sequence diagram example represents McDonald's ordering system:



Sequence diagram of McDonald's ordering system

The ordered sequence of events in a given sequence diagram is as follows:

1. Place an order.
2. Pay money to the cash counter.
3. Order Confirmation.
4. Order preparation.
5. Order serving.

If one changes the order of the operations, then it may result in crashing the program. It can also lead to generating incorrect or buggy results. Each sequence in the above-given sequence diagram is denoted using a different type of message. One cannot use the same type of message to denote all the interactions in the diagram because it creates complications in the system.
You must be careful while selecting the notation of a message for any particular interaction. The notation must match with the particular sequence inside the diagram.

Benefits of a Sequence Diagram

- Sequence diagrams are used to explore any real application or a system.
- Sequence diagrams are used to represent message flow from one object to another object.
- Sequence diagrams are easier to maintain.
- Sequence diagrams are easier to generate.
- Sequence diagrams can be easily updated according to the changes within a system.
- Sequence diagram allows reverse as well as forward engineering.

Drawbacks of a sequence diagram

- Sequence diagrams can become complex when too many lifelines are involved in the system.
- If the order of message sequence is changed, then incorrect results are produced.
- Each sequence needs to be represented using different message notation, which can be a little complex.
- The type of message decides the type of sequence inside the diagram.