Sequence and State Machine Diagrams

System Analysis and Design

Istanbul Arel University

Spring 2023

Sequence diagram

Remember that the most used UML diagram is the *class diagram*. The second is the **sequence diagram**.

It is an *interaction diagram* that models a single scenario executing in a system.

- It shows what messages are sent and when.

Sequence diagram

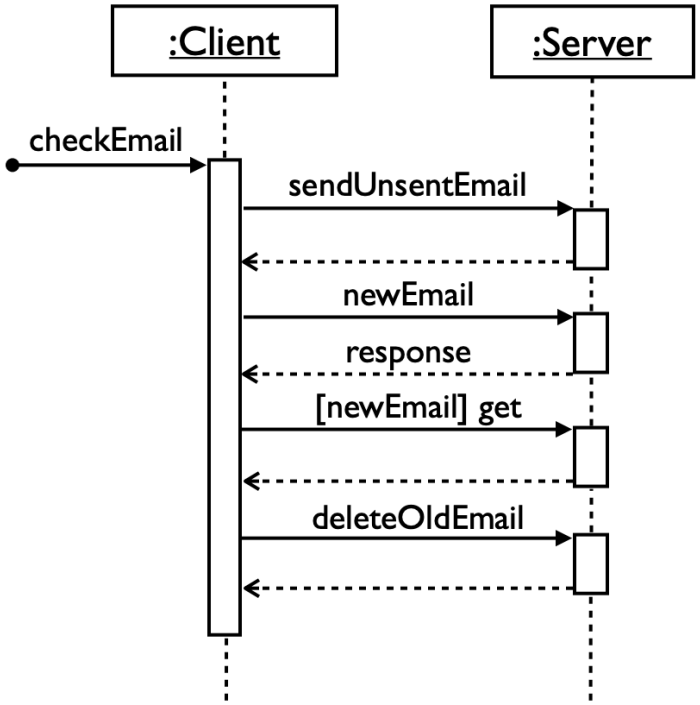
Type of a *behavior diagram* that illustrate how objects interact with each other in a particular scenario or use case.

They show the *order* in which messages are exchanged between objects and the *lifeline* of each object, which represents the time during which the object is active and involved in the interaction.

Why use them?

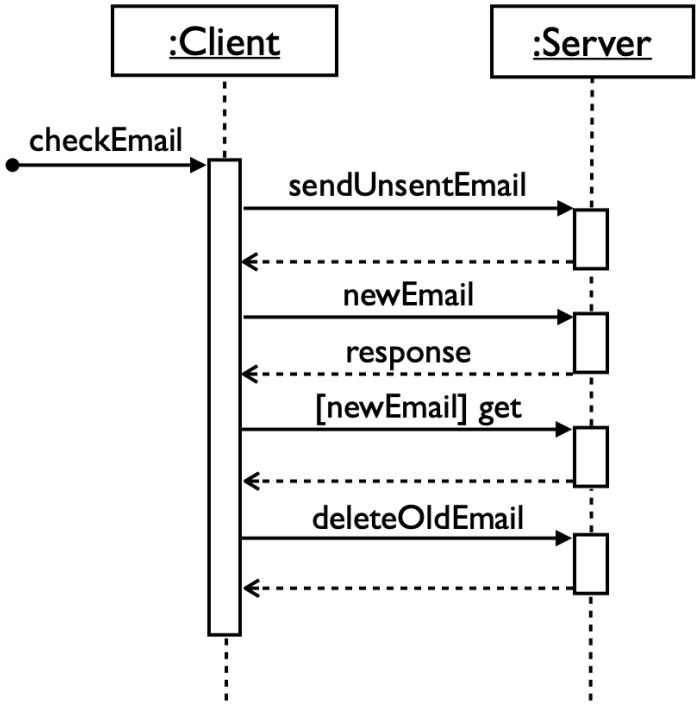
- They are language agnostic. They can be implemented in different languages.
- It is easy for non-coders to understand and create sequence diagrams.
- Can see many objects/classes on the same page.

Sequence diagram



- Vertical lines: Lifelines which represent the objects involved in the interaction.
- Horizontal arrows: Represent the messages exchanged between the objects.
  - Arrows are labeled with the name of the message and parameters passed. Can also show the return value.

Key parts

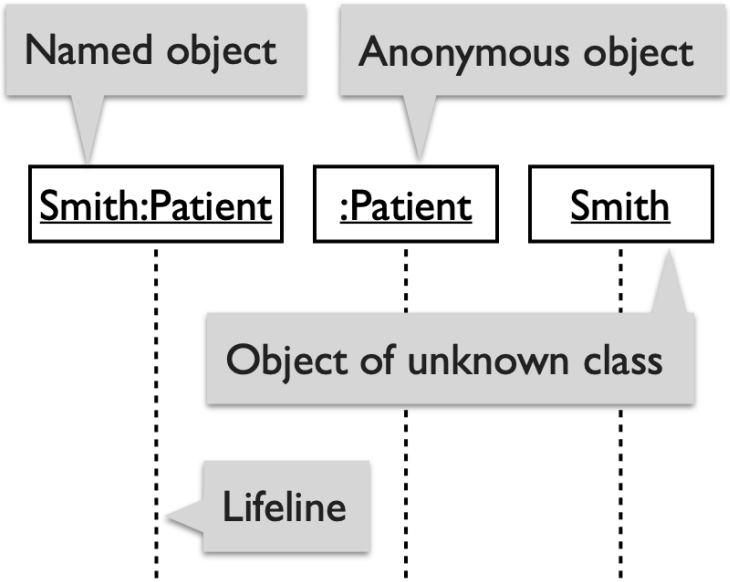


We have a **participant** and a **message**.

- Participant is the object or an entity. The *actor* in the sequence diagram.
- Message is the communication between the objects.
- Axes:
  - Horizontal: Which participant is acting
  - Vertical: Time (top to bottom - forward in time)

Object representation

- `objectname:classname`
- The lifeline is represented by the *dashed* line.
  - It represents the *life span* of the object during the scenario being modeled.



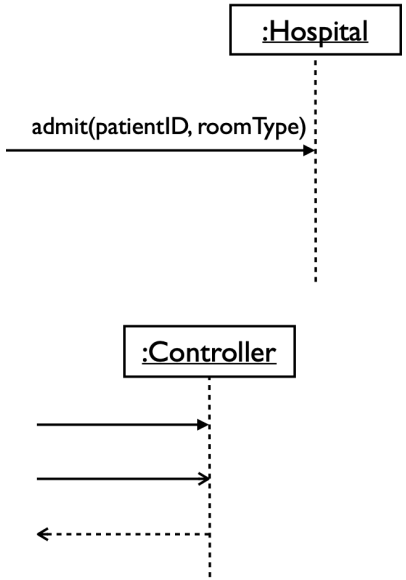
Message representation

The horizontal arrow to the receiving object represents a **message**.

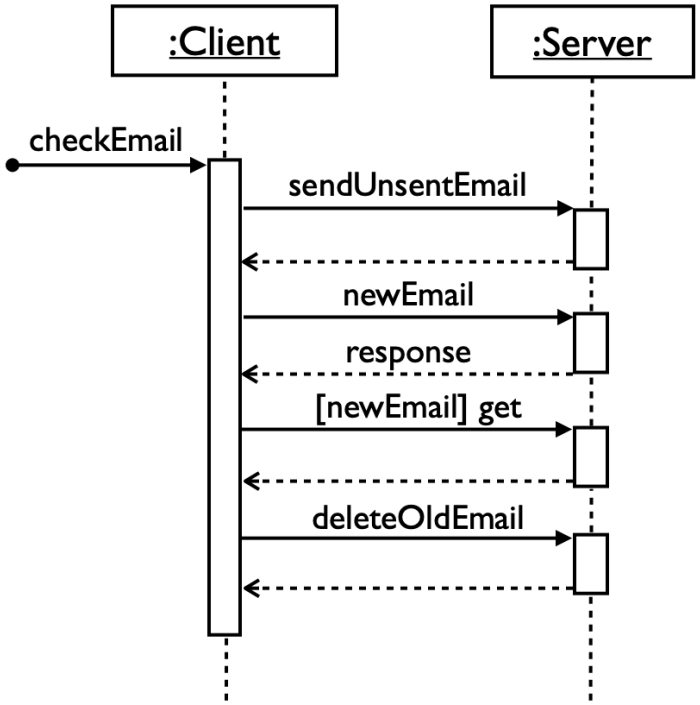
- Remember that a message is just a method(function) call.
- We write the message name and arguments on top of the arrow.

Different kinds of arrows tells different things.

- Synchronous message
- Asynchronous message
- Return message

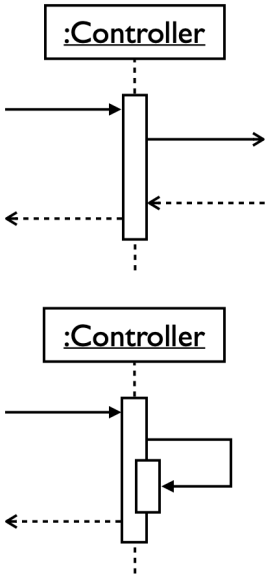


Key parts



1. The user presses *check email*
2. Client first sends all unsent email to the server.
3. After receiving an ACK, client asks the server if there is any new email.
4. If so, it downloads.
5. Next, it deletes old thrashed email from the server.

Method execution

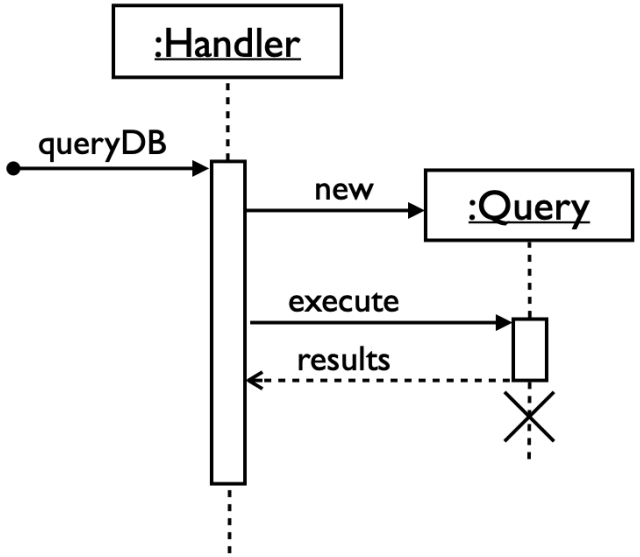


The thick box over the lifeline is drawn when an object's method is on the stack.

- Either that object is running its code, or it is on the stack waiting for another's method to finish.

Nest activations to indicate an object calling itself.

Lifetime of objects



When there is an arrow with **new** written over it, it means *object creation*.

- An object created after the start of the scenario appears lower than the others.

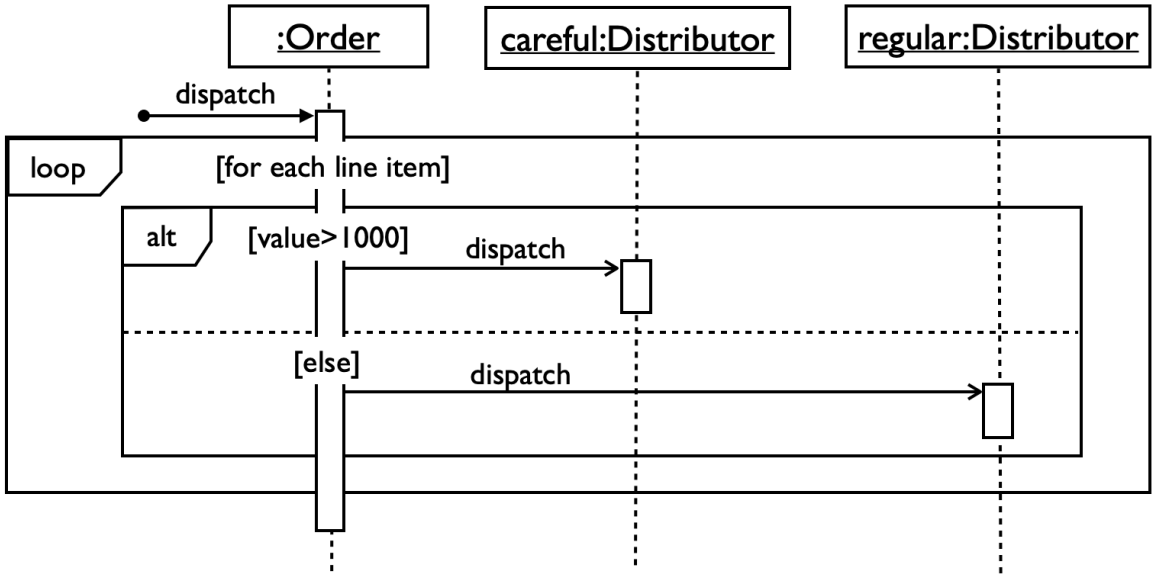
The **X** at the bottom represents object deletion.

- Some programming languages have *garbage collection*, so they may not explicitly delete objects. (Java)

Loops

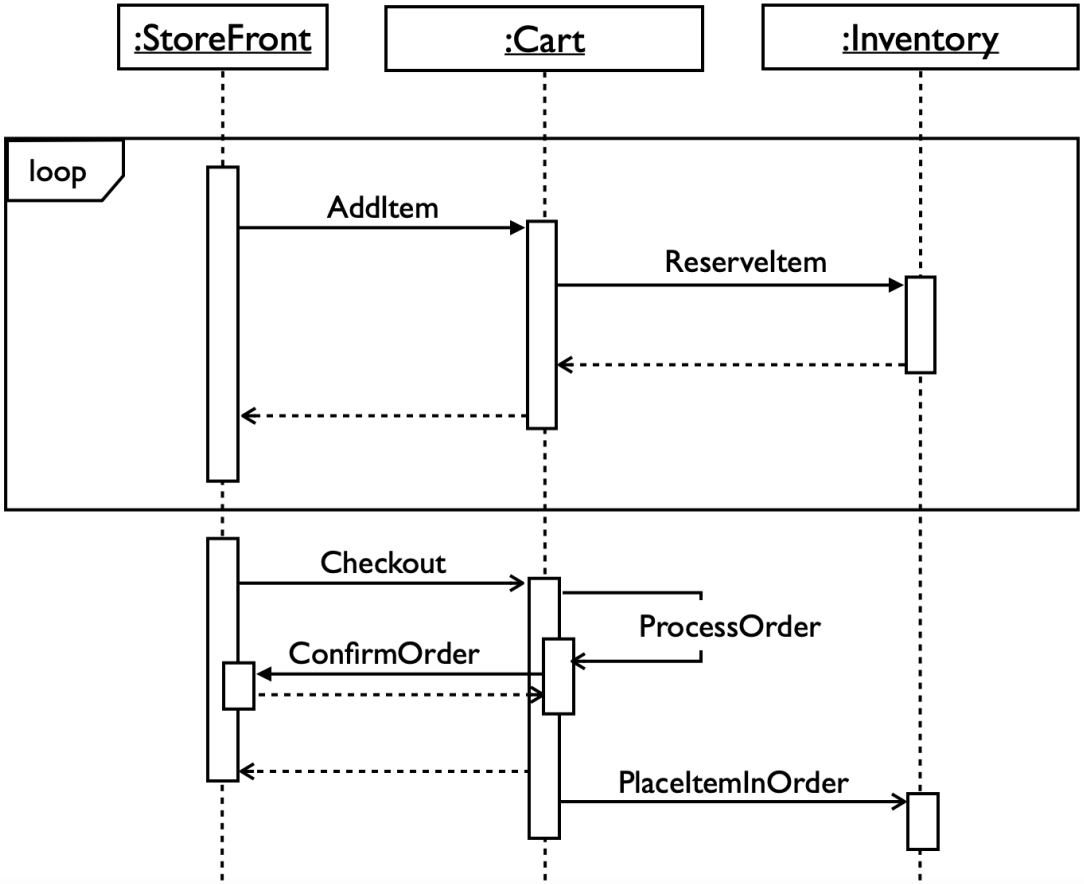
When there is a box around the part of a sequence diagram, it is called a **frame**.

- if -> opt [condition]
- if/else -> alt [condition], separated by horizontal dashed line
- loop -> loop [condition or items to loop over]



One can also refer to another sequence diagram by putting it in a *frame*.

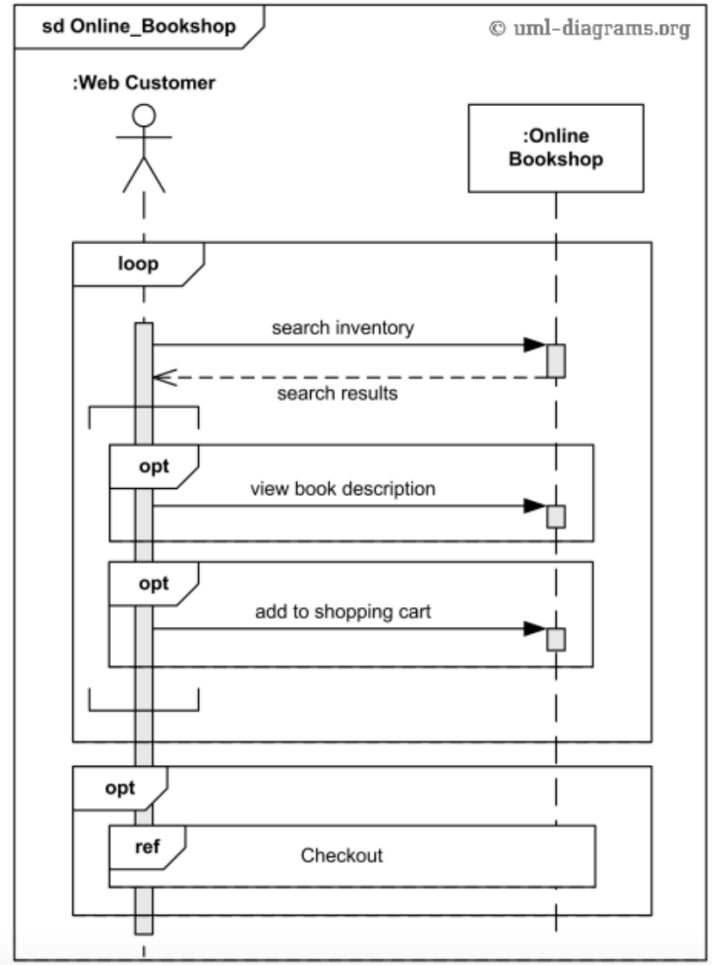
Example


:StoreFront
:Cart
:Inventory
loop
AddItem
ReserveItem
Checkout
ProcessOrder
ConfirmOrder
PlaceItemInOrder

Real life example

1. The customer begins the interaction by searching for a book by title.
2. The system will return all books with that title.
3. The customer can look at the book description.
4. The customer can place a book in the shopping cart.
5. The customer can repeat the interaction as many times as desired.
6. The customer can purchase the items in the cart by checking out.

Sequence diagram



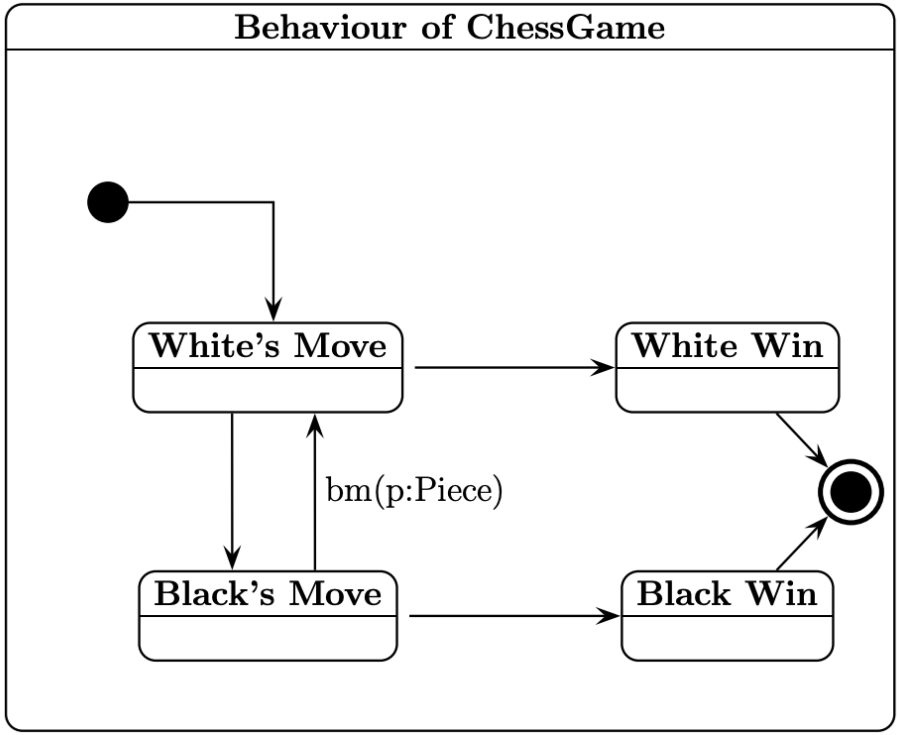
STATE MACHINE DIAGRAMS

Definition

State machine diagrams are used to specify *state machines*.

- It shows a *dynamic* flow.

It has **events**, **states** and **transition**.



Event, state, transition

Event: A significant or noteworthy occurrence.

- bm(p:Piece)

State: Condition of an object at a moment in time. Nodes on the graph.

- White's Move, Black's Move, etc.

Transition: The relationship between two states that indicates when an event occurs.

- Arrows

When to use

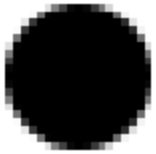
State machines are generally used at an early stage of software development.

When we don't fully grasp the behavior of an object or operation, we get help from these diagrams.

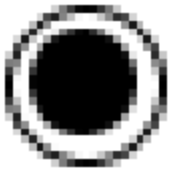
It is not very useful if there are *several* objects.

State

- Initial Pseudo State:
  - It represents a default vertex that is the source of a single transition to the default state.
  - There can be only one initial vertex in a region.



- Final state:
  - A special kind of state which states that the enclosing region is completed, leading to the entire state machine being completed.



State

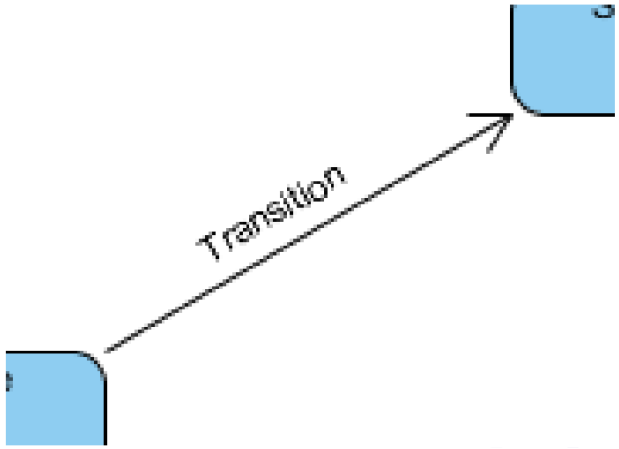
- A state models a situation during which some invariant condition holds.
- The invariant may represent a static situation such as an object waiting for some external event to occur.

Choice

- It allows splitting of transitions into multiple outgoing paths.
- One should always provide an *else* in these choice diagrams.

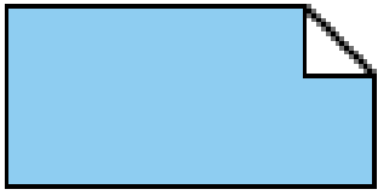
Transition

- A transition is a directed relationship between a source vertex and a target vertex.



Note

- We can use notes to attach various remarks to diagrams.
- A comment has no force but can include useful information.



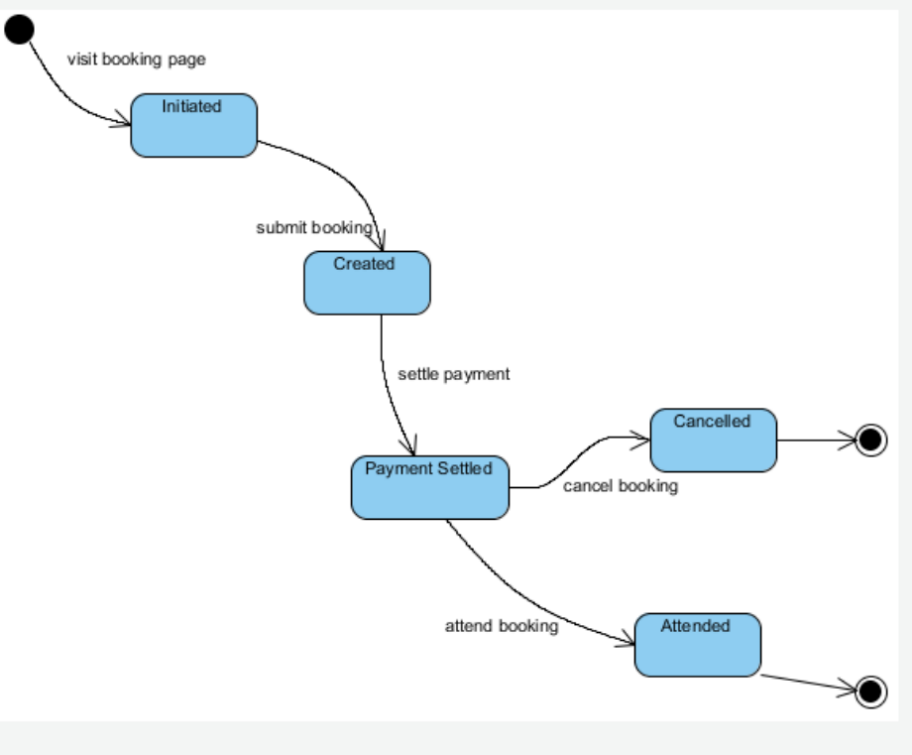
State dependent and state independent objects

- If an object always responds the same way to an event, it is considered **state-independent** with respect to that event.
- If for all events of interest an object always reacts the same way, it is a **state-dependent** object.
    - State-dependent objects react differently to events depending on their state.
- Business information systems have **few** state-dependent classes, so it is not helpful to apply a state machine modelling.

State dependent and state independent objects

- Process control, device control, protocol handlers, and telecommunication domains often have many state-dependent objects; state machine modelling would be useful in these cases:
  - A telephone
  - State-dependent, because the reaction changes depending on the current mode of the phone.
    - Yes and No buttons will act differently.

Example


visit booking page
Initiated
submit booking
Created
settle payment
Payment Settled
Cancelled
cancel booking
attend booking
Attended

References

- https://courses.cs.washington.edu/courses/cse403/15sp/lectures/L10.pdf
- https://warwick.ac.uk/fac/sci/physics/research/condensedmatt/imr_cdt/students/david_goodwin/teaching/modelling/l2_umlactivity.pdf
- https://formal.kastel.kit.edu/~beckert/teaching/Spezifikation-SS04/10StateCharts.pdf