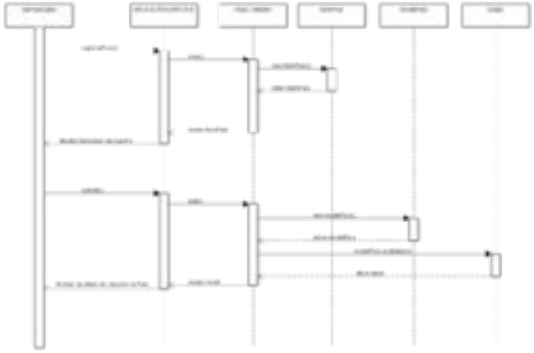
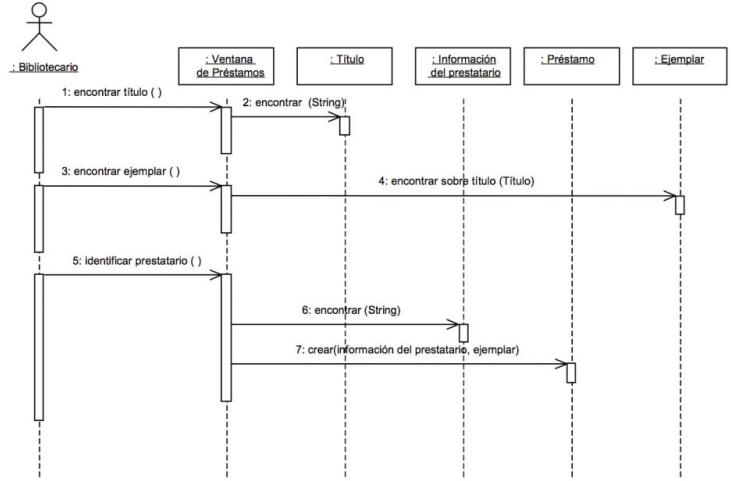
# UML Diagramas de secuencia





# Índice

# ¿Qué son?

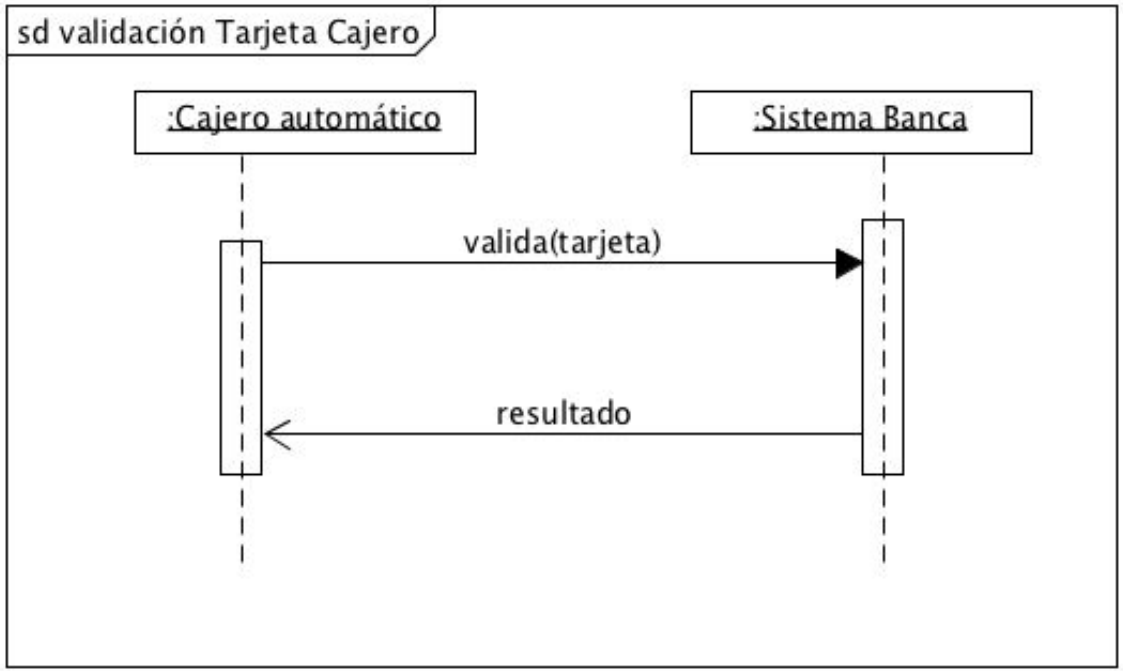
Muestra la forma en la que un grupo de objetos se comunica (interactúan) entre sí a lo largo del tiempo.

En el cual se indicarán los módulos o clases que formarán parte del programa y las llamadas que se hacen cada uno de ellos para realizar una tarea determinada, por esta razón permite observar la perspectiva cronológica de las interacciones.

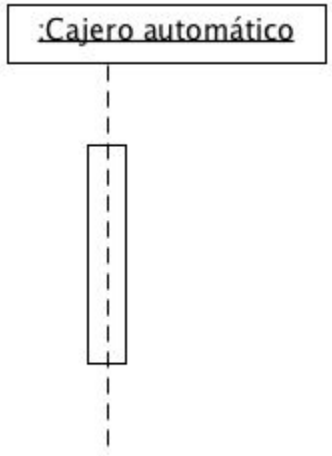
Es importante recordar que el diagrama de secuencias se realiza a partir de la descripción de un caso de uso.

# ¿Qué muestra un diagrama de secuencia?

- Los objetos(actores) que participan en la interacción
- La secuencia de mensajes intercambiados.



# Participantes



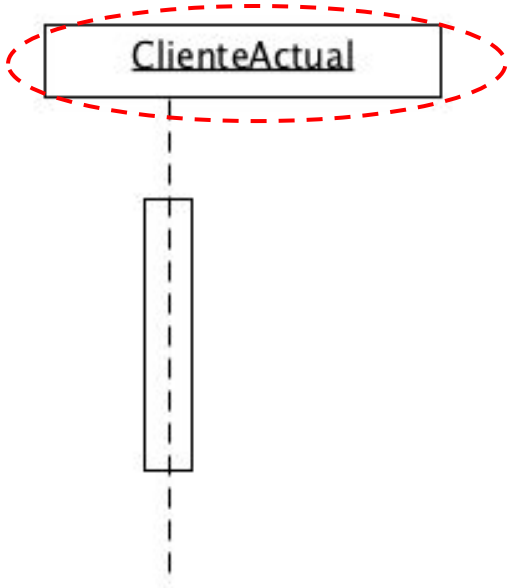

Los diagramas de secuencia están compuestos por un conjunto de participantes que interactúan con otros participantes durante la secuencia.

La sintaxis es la siguiente

***nombre_objeto[selector]:nombre_clase ref descomposición***

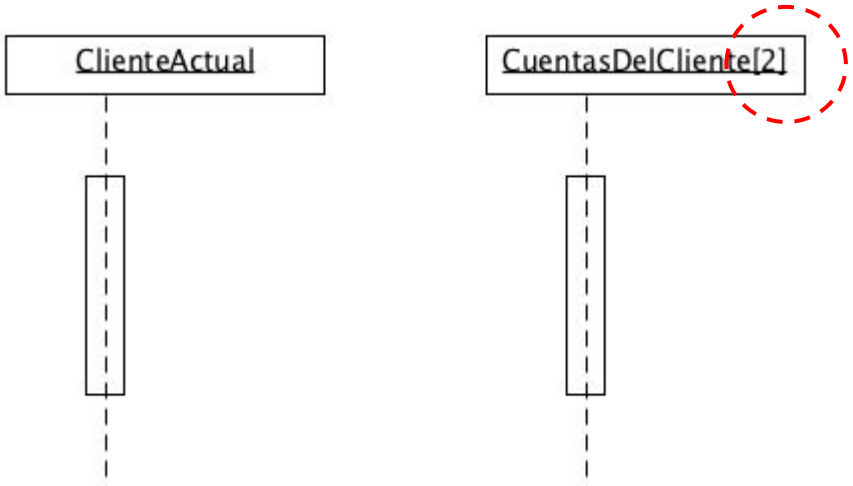
# Participantes: nombre_objeto

Especifica el nombre de la instancia involucrada en la interacción.



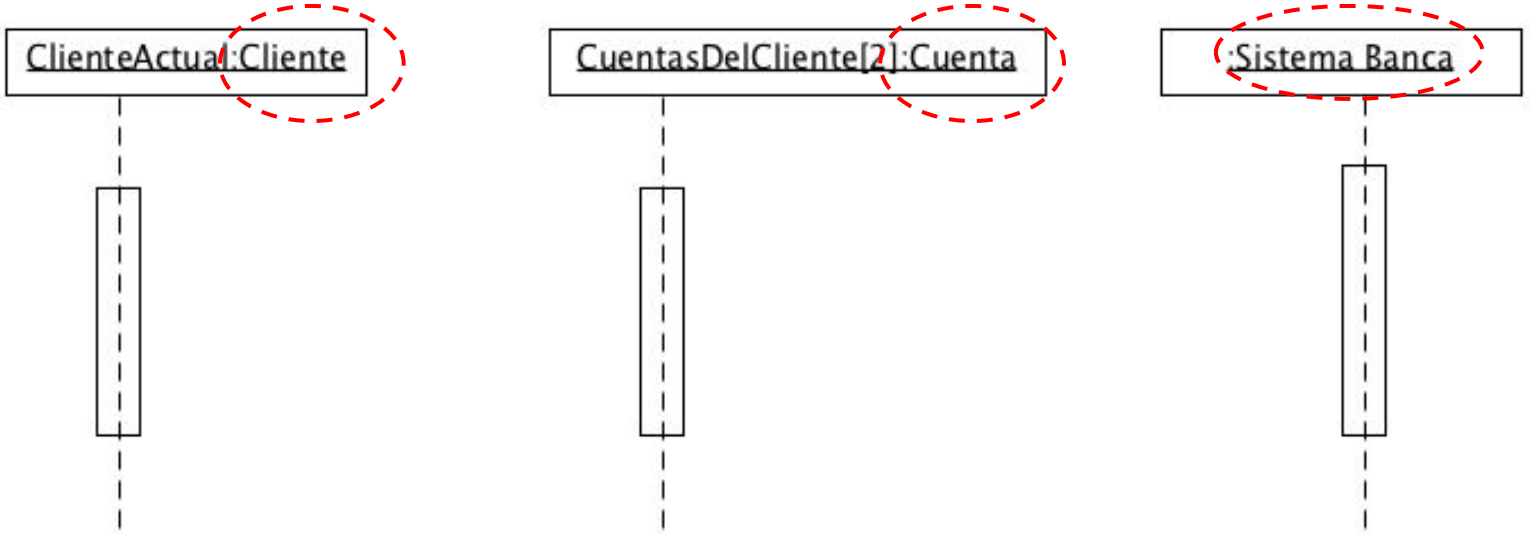
# Participantes: Selector

Es opcional y se usará cuando la instancia sea una colección, y queramos indicar cuál elemento de esa colección es el que se está usando.



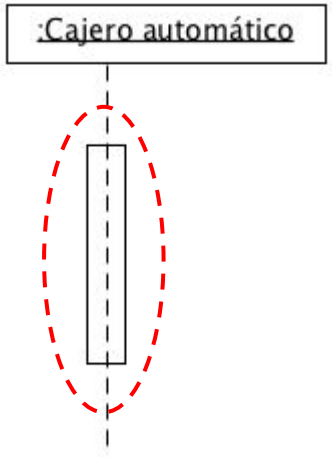
# Participantes: Nombre_clase

Si fuera necesario podemos especificar el nombre de la clase



# Barras de activación

La barra de activación representa la duración de la acción en tiempo y la relación de control entre la activación y sus llamantes

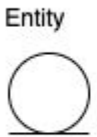


# Más símbolos usados en los participantes

Ángel González M.

# Entidad / Entity

Podemos usar símbolos de persistencias: habitualmente se encargan de guardar datos en ficheros o en bases de datos

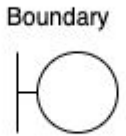



# Control / lógica de negorio

Podemos usar símbolos de lógica de negocio: se corresponden con tareas que nuestra aplicación va a realizar. Ej realizar cálculos, validaciones, etc.

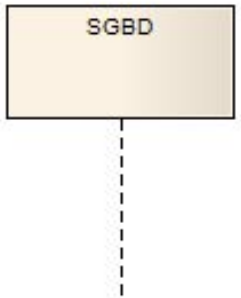

# Interfaces gráficas / boundary

Podemos usar símbolos de presentación: habitualmente se corresponden con las interfaces gráficas de nuestro proyecto (la vista)




# También podemos usar actores

# O simplemente podemos usar el símbolo de una caja



# Mensajes

Ángel González M.

# Mensajes

La comunicación entre los participantes puede ser:

- llamadas a métodos
- envíos de señales
- creación de instancias de objetos
- destrucción de objetos
- etc.

La sintaxis es la siguiente:

***atributo = señal_o_nombre_operacion(argumentos): valor_retorno***

# Mensajes: atributos

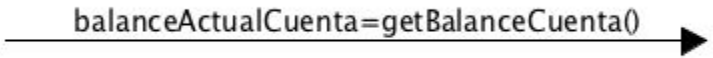
***atributo* = *señal_o_nombre_operacion(argumentos): valor_retorno***

Es opcional, indica donde se **almacena el valor devuelto** por el mensaje. (como si fuera una variable/atributo de java)

# Mensajes: señal_o_nombre_operacion

***atributo* = *señal_o_nombre_operacion*(*argumentos*): *valor_retorno***

Indica el nombre de la **operación** a invocar, o la señal que se está emitiendo. (como si fuera una procedimiento/funcion de java)



# Mensajes: argumentos

***atributo = señal_o_nombre_operacion(argumentos): valor_retorno***

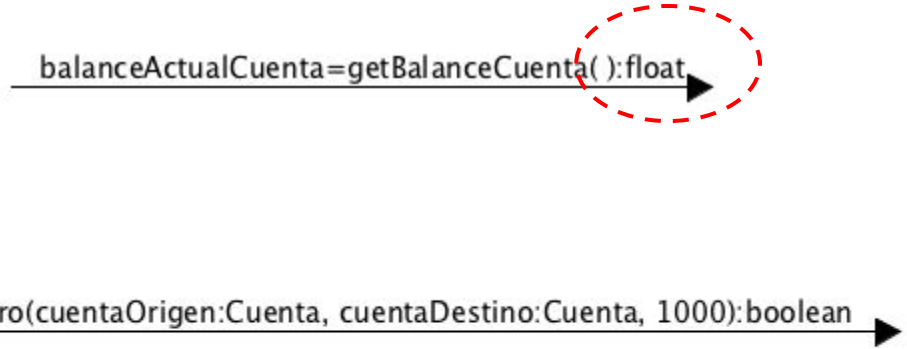
Los **parámetros** que se envían en el mensaje.


*Nota: Observa que el 3er parámetro (1000) es una constante.*

# Mensaje: valor_retorno

***atributo = señal_o_nombre_operacion(argumentos): valor_retorno***

Si queremos clarificar aún más la operación podemos especificar el tipo de dato o valor que se va a retornar



# Tipos de mensajes

# Tipos de mensajes

- Simples
- Síncronos
- Asíncronos
- De retorno
- De creación
- De borrado
- Perdidos y encontrados
- Self / Recursivos
- De rechazo
- Con restricciones de tiempo y duración

# Mensaje simple

Es la transferencia de control de un objeto a otro.

Se representa por una línea continua con flecha sin rellenar.

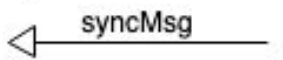
# Mensaje síncrono

En este tipo de mensajes, cuando se envían hay que esperar por la respuesta.

El programa suspende la ejecución mientras espera por la respuesta.

La llamada se representa por una line continúa con punta rellena. La respuesta se representa por una línea discontinua sin punta rellena

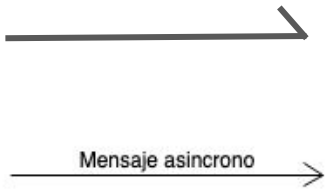
También lo podemos encontrar así, con el decorador syncMsg encima de la flecha.



# Mensaje asíncrono

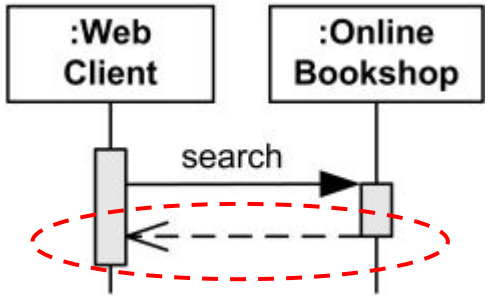
Por el contrario en este tipo de mensajes, se envía el mensaje y no se espera la respuesta, sino que se continúa con las demás llamadas. El flujo del programa continúa sin esperar la respuesta.

Podemos representarlo mediante cualquiera de estos 2 símbolos:



# Mensaje de retorno (reply message)

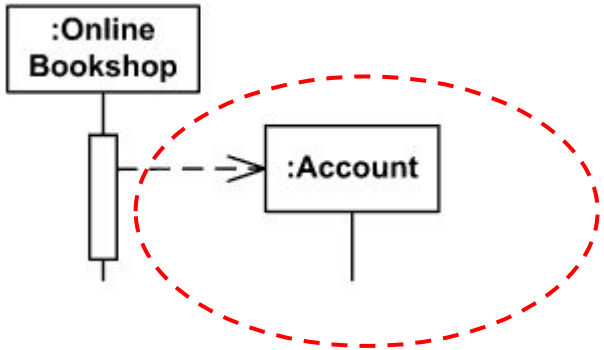
Si necesitamos representar un mensaje de retorno (tanto para mensajes síncronos como asíncronos), posterior al envío de un mensaje usaremos una línea discontinua sin punta rellenada.



# Mensaje de creación

Envía un mensaje para que se cree un elemento. Básicamente esto representa la instanciación de un nuevo objeto.

El símbolo al inicio de la línea de vida se muestra en un nivel más bajo de la página que el símbolo del objeto que causó la creación

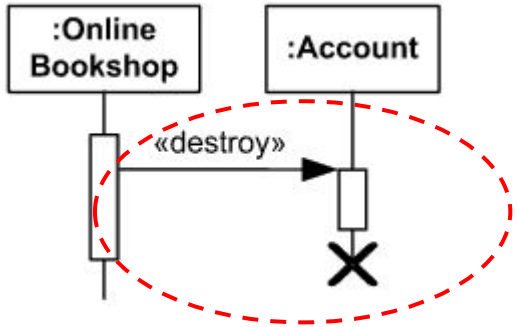


# Mensaje de borrado (delete mensaje)

En versiones anterior de uml se llamaba *stop message*.

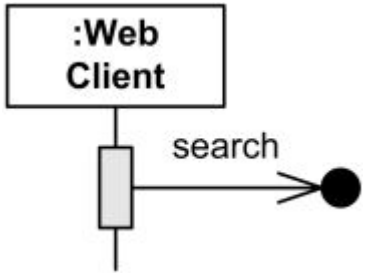
Es enviado para terminar el ciclo de vida de otro elemento.

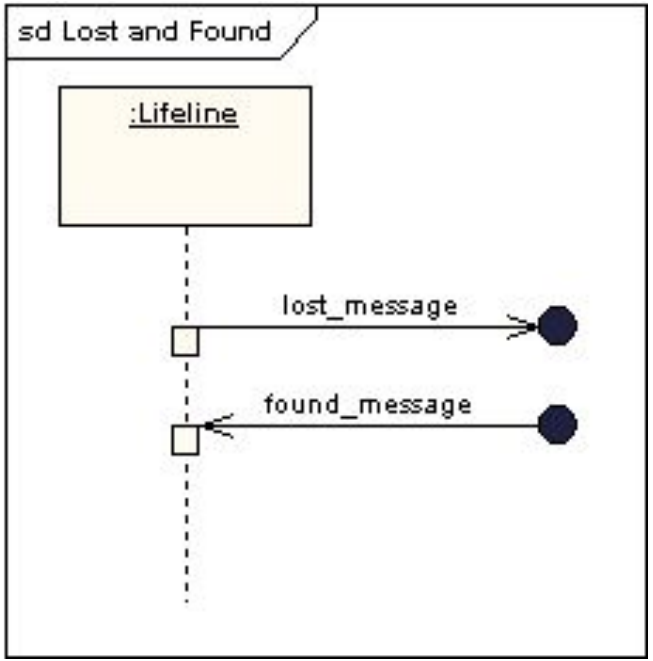
Opcionalmente se puede usar el estereotipo <<destroy>> encima de la línea.



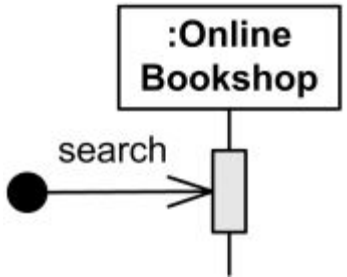
# Mensajes perdidos y encontrados

Los **mensajes perdidos** son aquellos que han sido enviados pero que no han llegado al destino esperado, o que han llegado a un destino que no se muestra en el diagrama actual.



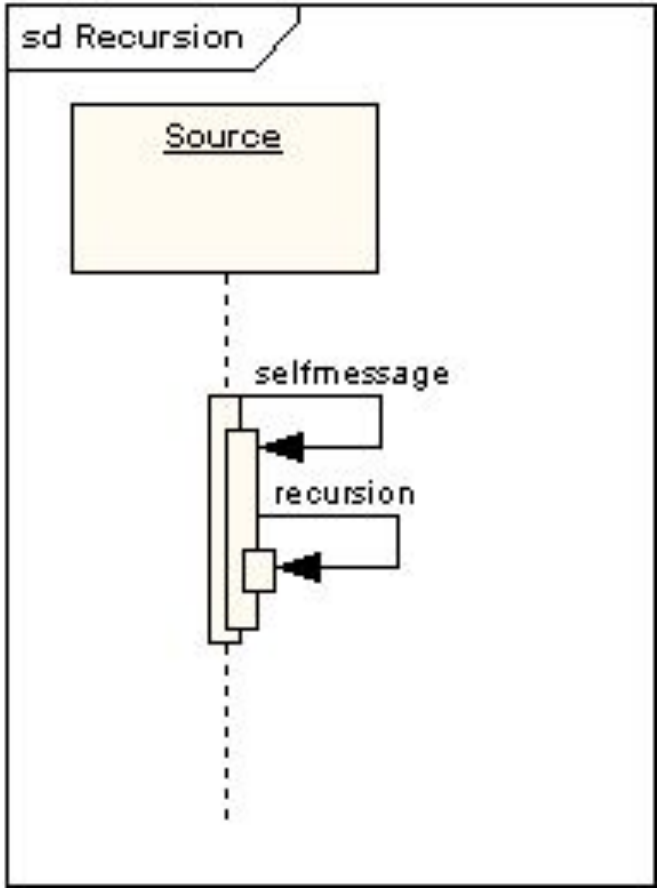

Los **mensajes encontrados** son aquellos que llegan de un remitente no conocido, o de un remitente no conocido en el diagrama actual. Ellos se denotan yendo o llegando desde un elemento




Ángel González M.

# Mensaje Self / Recursividad

Un **mensaje self** puede representar una llamada recursiva de una operación, o un método llamando a otro método perteneciente al mismo objeto. Este se muestra como cuando crea un foco de control anidado en la ocurrencia de ejecución de la línea de vida.



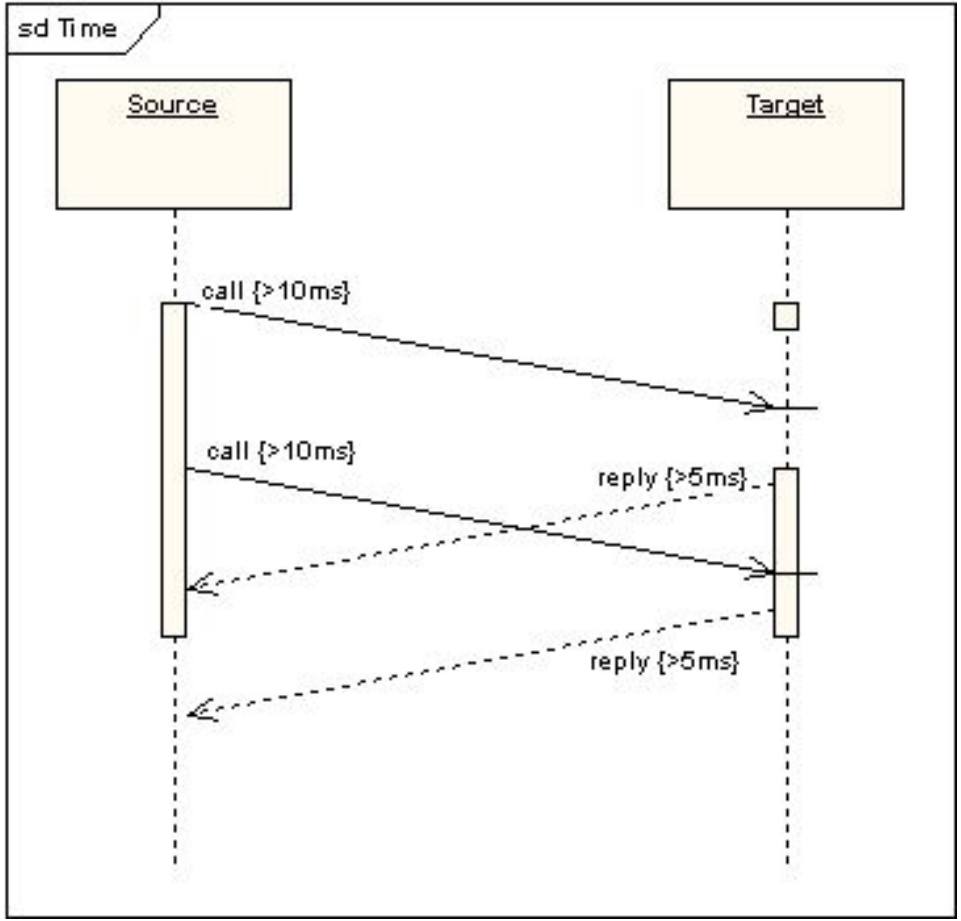

Ángel González M.

# Rechazo

Podemos representar un mensaje que se ha rechazado de esta forma:

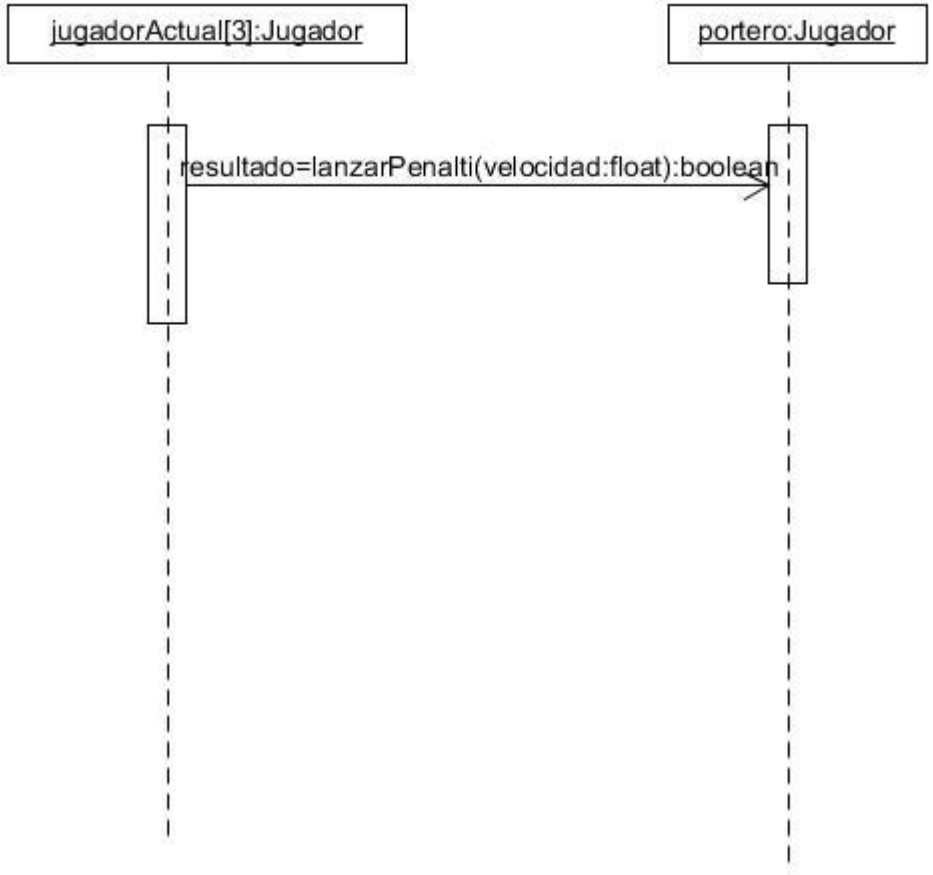
# Con restricciones de tiempo y duración

En forma predeterminada, un mensaje se muestra como una línea horizontal. Ya que la línea de vida representa el pasaje de tiempo hacia abajo, cuando se modela un sistema en tiempo real, o incluso un proceso de negocios en tiempo límite, puede ser importante considerar el tiempo que toma realizar las acciones. Al configurar una restricción de duración para un mensaje, el mensaje se mostrará como una línea inclinada.



# Ejemplo

Representamos en este diagrama de secuencia la acción de lanzar un penal



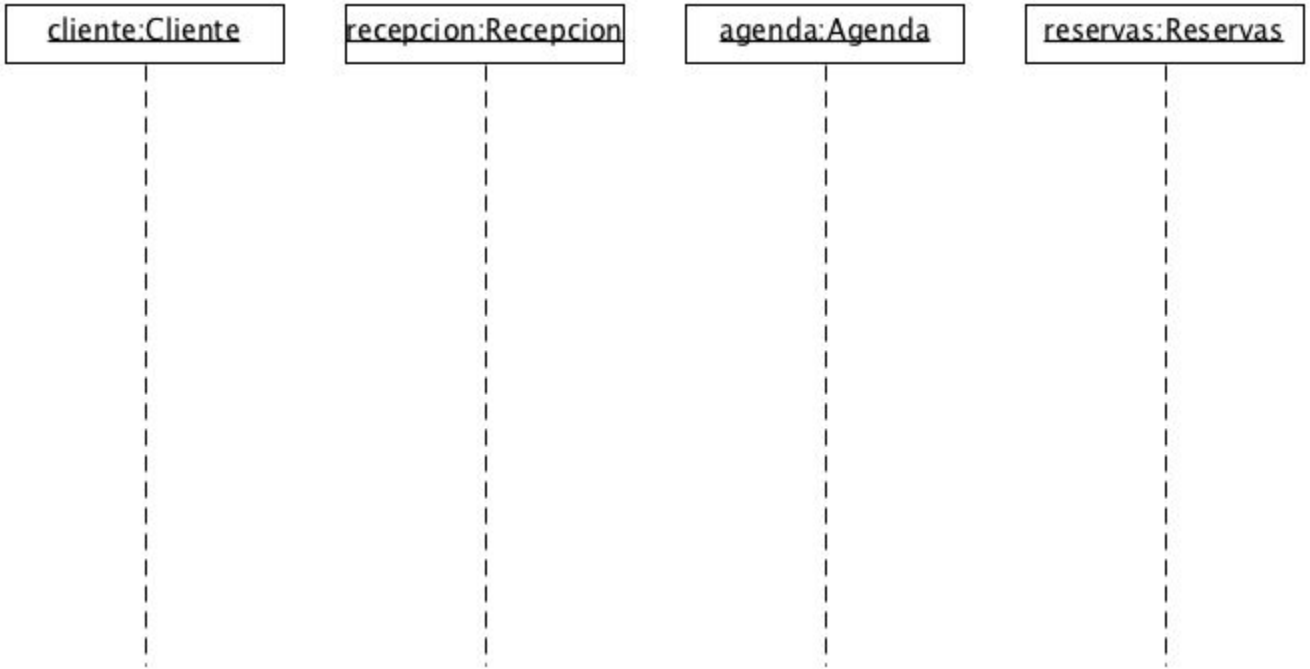
# Ejemplo: Reserva de habitación en hotel

Ángel González M.

# Ejemplo

Veamos el ejemplo de un diagrama de secuencia donde un cliente reserva una habitación en un hotel.

# Creando los participantes



# Trazado de líneas de mensaje/estímulo entre objetos

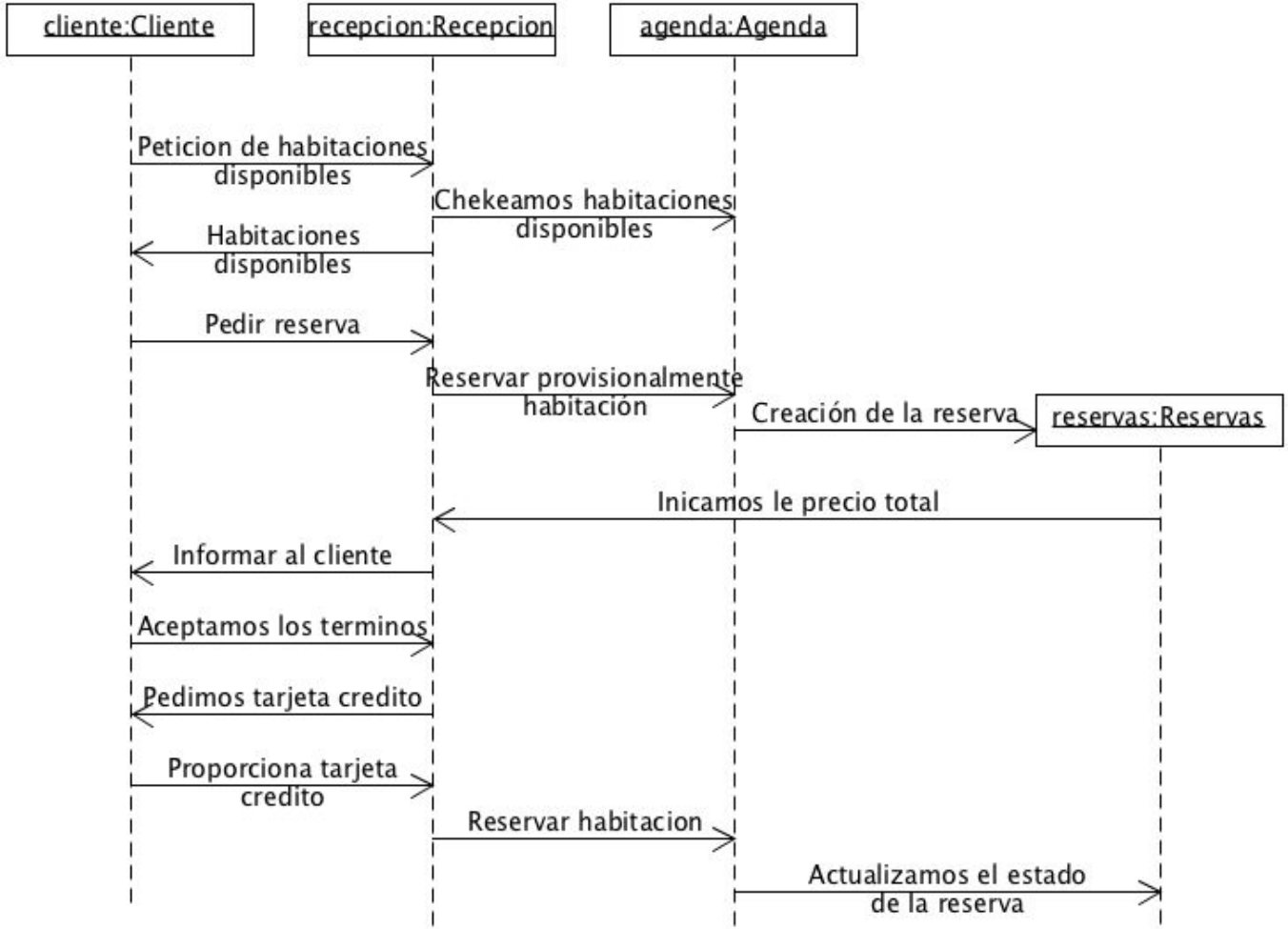
Ahora el cliente puede solicitar reservar la habitación. La recepción (Recepción) reservará la habitación en la agenda, lo que hará que se cree la reserva.

# Mostrar la creación de objetos

La notación UML para mostrar que se está creando un objeto en un diagrama de secuencia, consiste en trazar una línea de mensaje a la cabecera del cuadro de nombre, en la parte superior de la línea de vida de objeto. Para mostrar que se está creando la reserva, moverá el objeto Reservas hacia abajo en la línea de vida para poder trazar una línea dentro del recuadro de cabecera

# Continuación

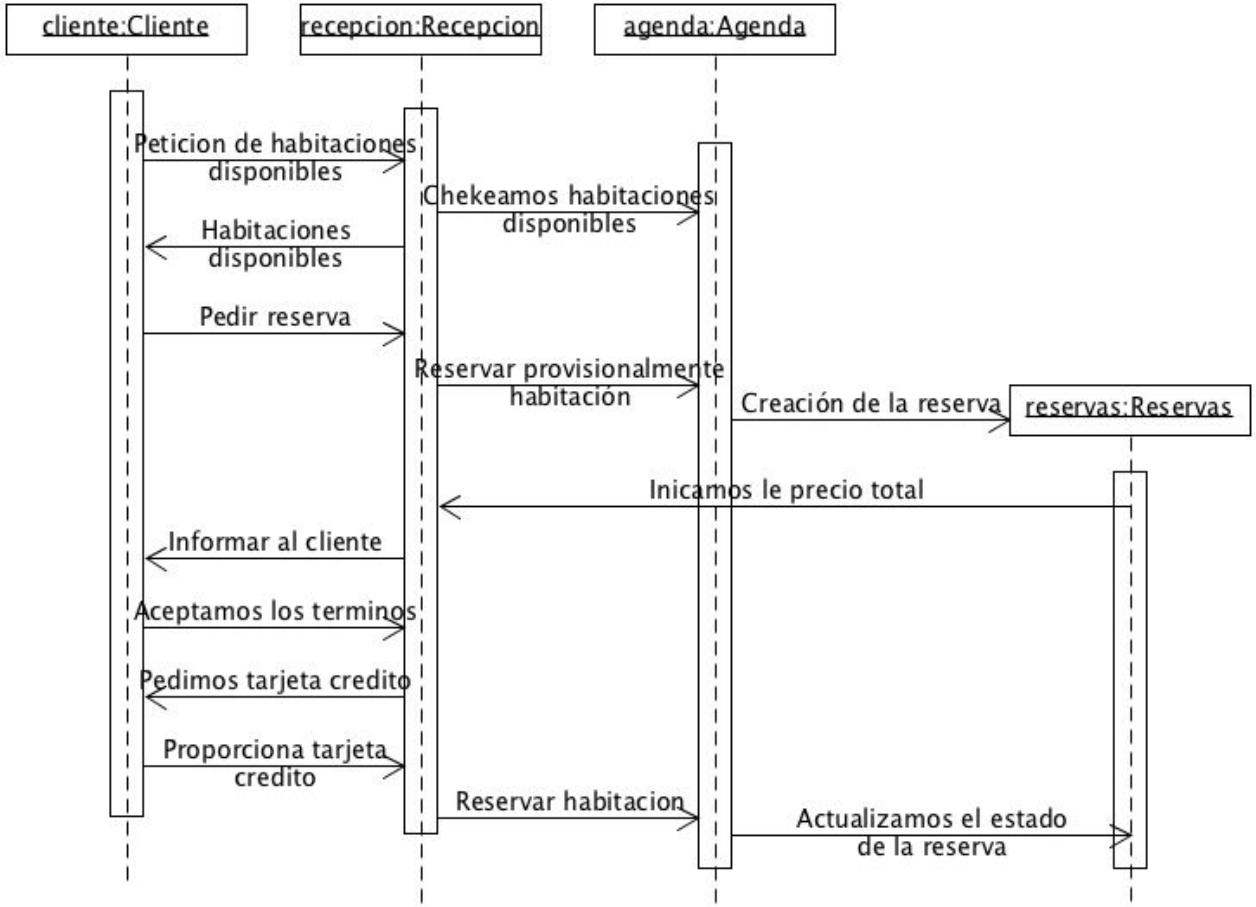
El objeto Reservas calculará su precio y lo notificará a la Recepcion que a su vez informará al cliente (la agenda no está implicada en este proceso de cálculo de precio). Si el cliente acepta los términos, lo notifica a Recepcion qué reserva la habitación en la agenda. La agenda actualiza el estado de la reserva = (reservado).



# Añadir barras de activación

Un rectángulo de foco de control o barra de activación, se traza sobre una línea de vida de objeto para mostrar el periodo durante el que el objeto realiza una acción.

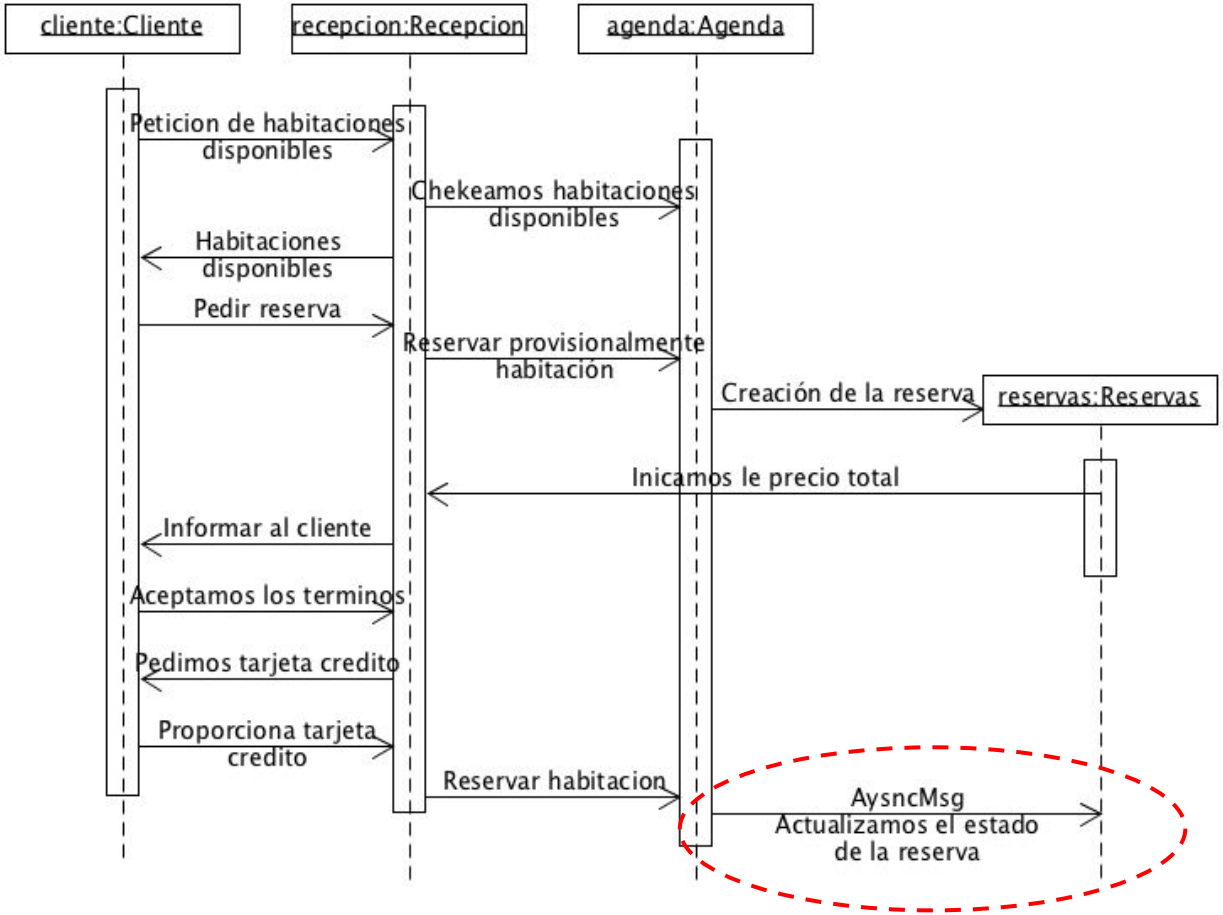
La barra de activación representa la duración de la acción en tiempo y la relación de control entre la activación y sus llamantes.



# Mensajes asíncronos

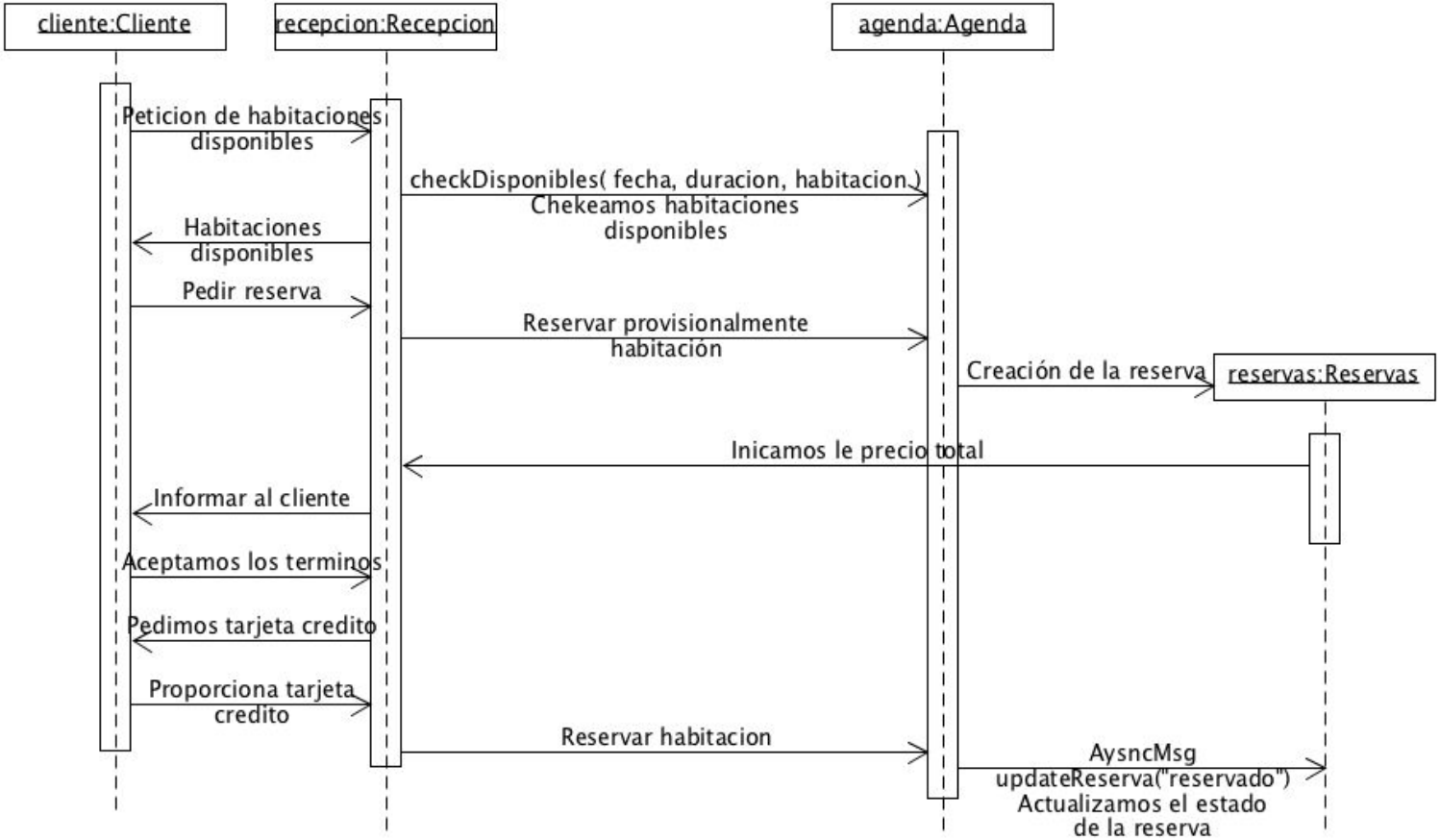
Podemos pensar que hay ciertos mensajes que son asíncronos.

Se envía el mensaje y no se espera la respuesta, sino que se continúa con las demás llamadas.



# Añadir un método a una línea de mensaje

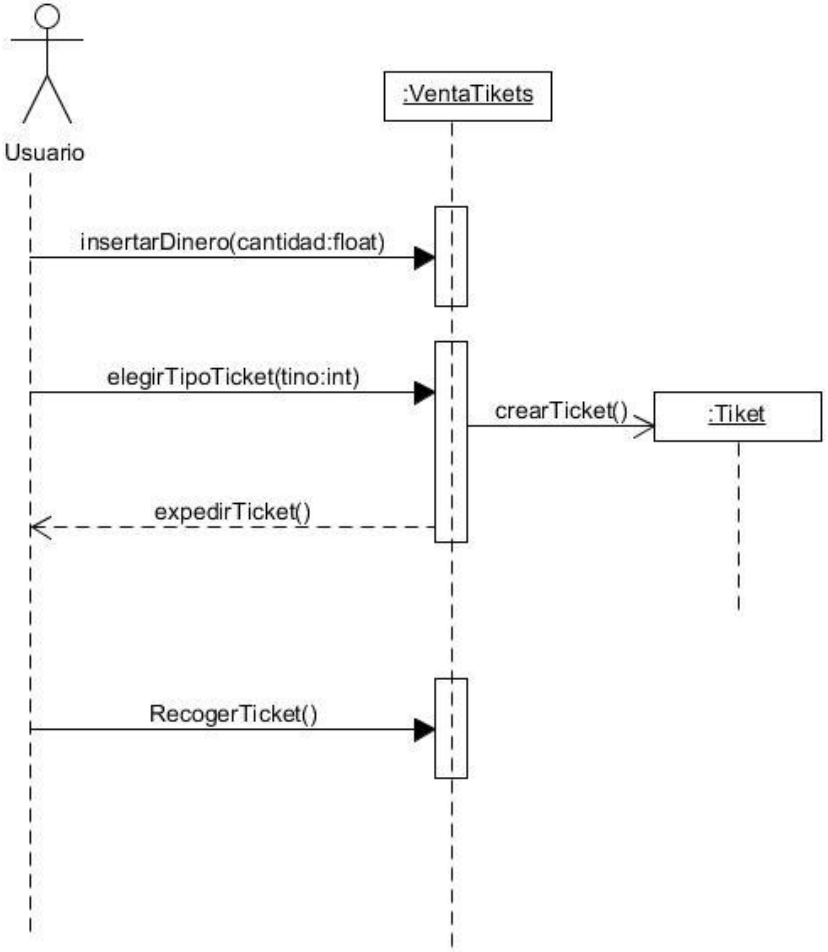
Podemos especificar los métodos que serán ejecutados



# Ejemplo: Coger un ticket de metro

# Ejemplo

Ejemplo de la secuencia de un usuario del metro para comprar un ticket



# Profundizando en los diagramas de secuencia

Ángel González M.

# Fragmentos combinados

Ángel González M.

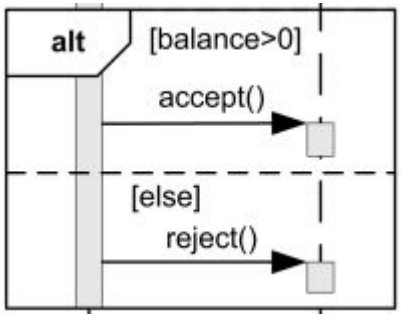
# Fragmentos combinados

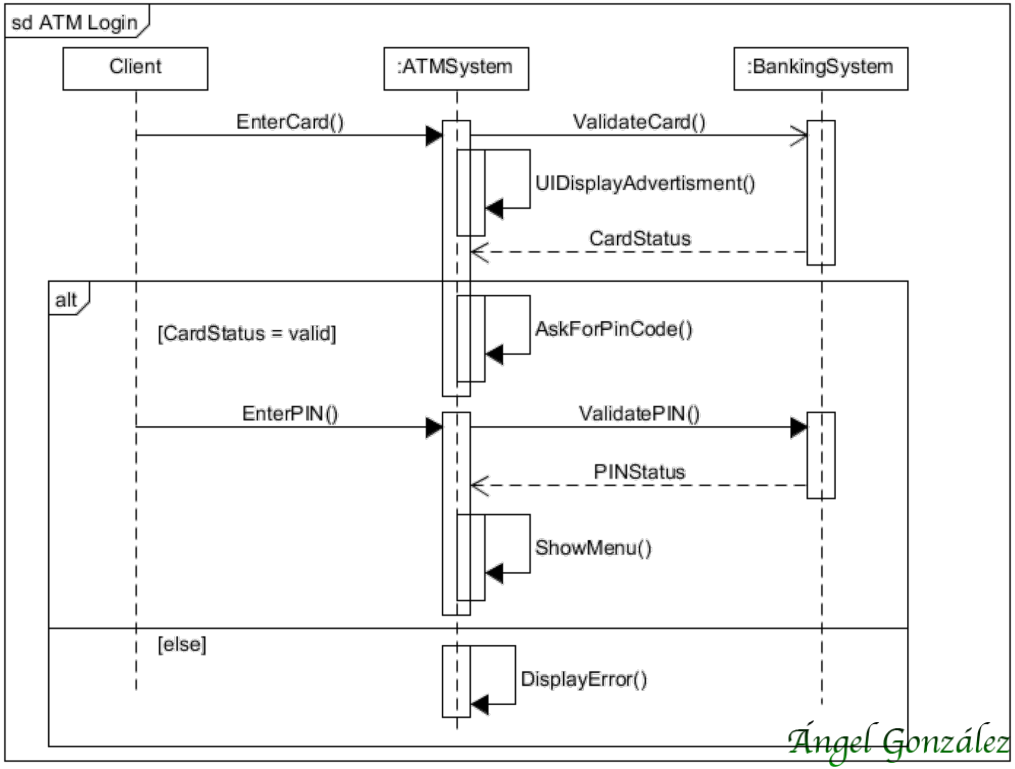
Se estableció anteriormente que no se espera que los diagramas de secuencia muestren lógicas de procedimientos complejos. Siendo este el caso, hay un número de mecanismos que permiten agregar un grado de lógicas de procedimientos a los diagramas y que a la vez vienen bajo el encabezado de fragmentos combinados. Un fragmento combinado es una o más secuencias de procesos incluidas en un marco y ejecutadas bajo circunstancias nombradas específicas. Los fragmentos disponibles son:

- Fragmento Alternative (denotado “alt”)
- Fragmento Option (denotado “opt”)
- Fragmento Break
- Fragmento Parallel
- Fragmento Loop
- Fragmento de secuenciado Weak (denotado “seq”)
- Fragmento de secuenciado Strict (denotado “strict”)
- Fragmento Negative (denotado “neg”)
- Fragmento Critical
- Fragmento Ignore
- Fragmento Consider
- Fragmento Assertion (denotado “assert”)

# Alternative (alt)

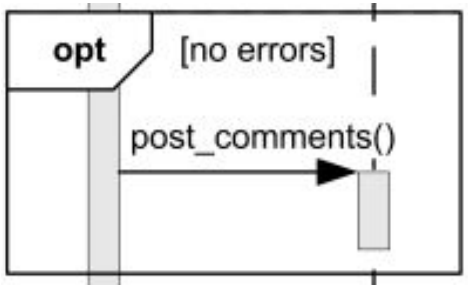
El fragmento Alternative (denotado “alt”) modela estructuras if…then…else.

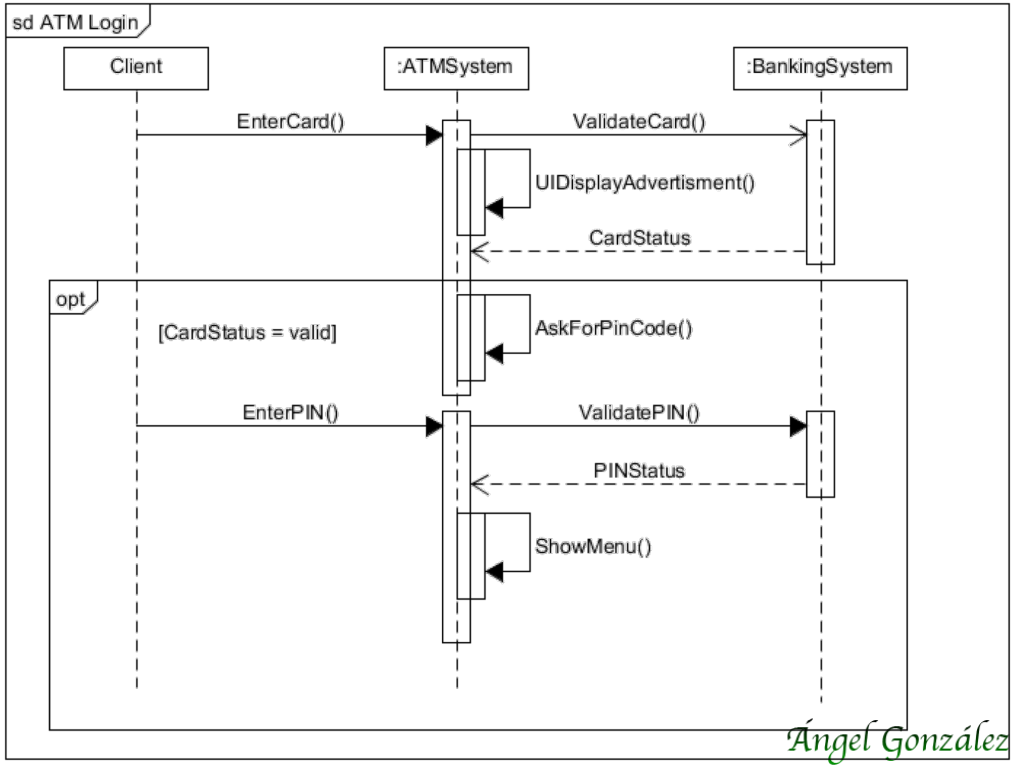


# Option (opt)

El fragmento Option (denotado “opt”) modela estructuras switch.

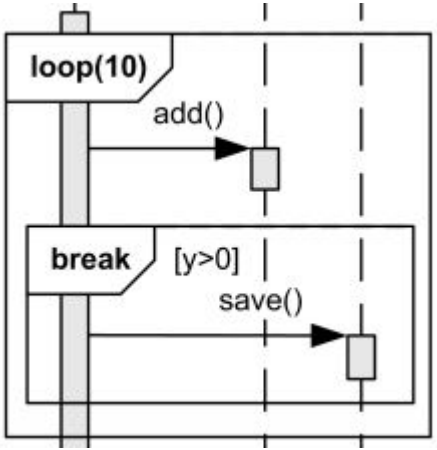


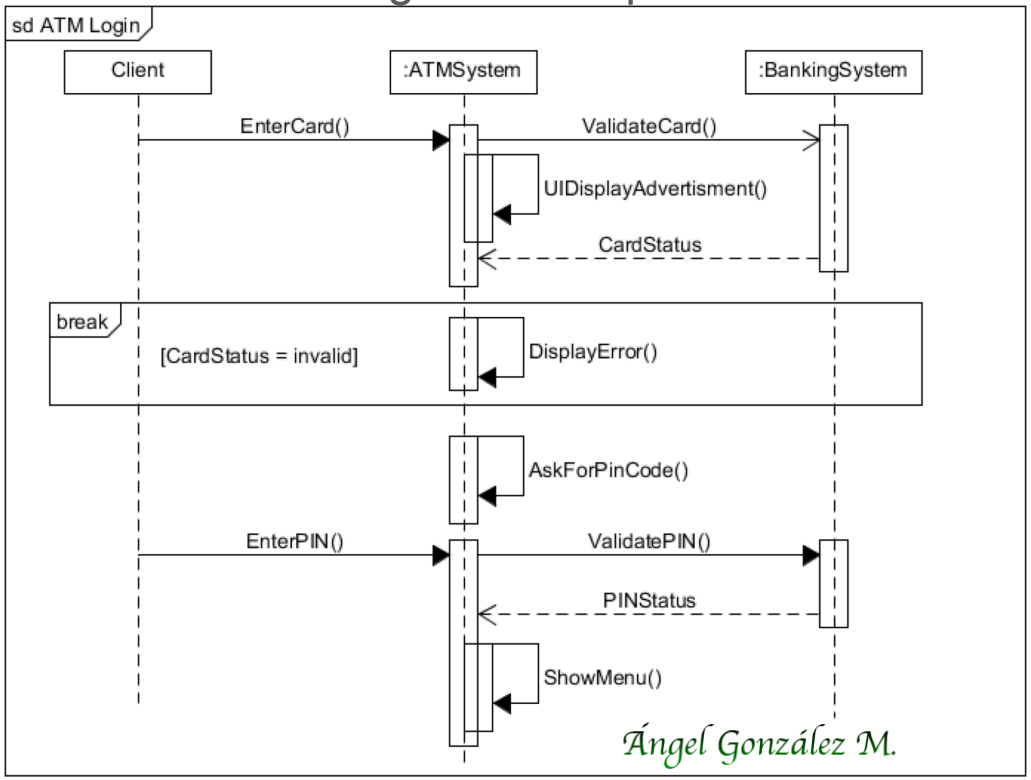



# Break (break)

El fragmento Break indica que se abandona el resto de la secuencia de ejecución.

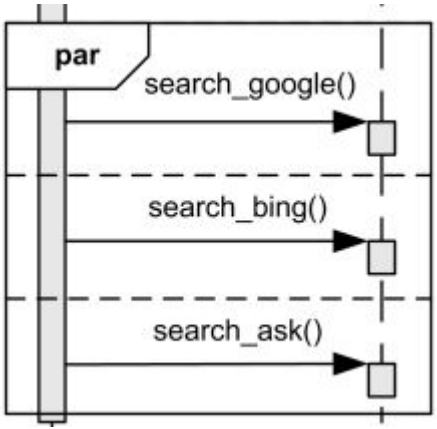
Es muy común que lo encontremos dentro de un fragmento loop

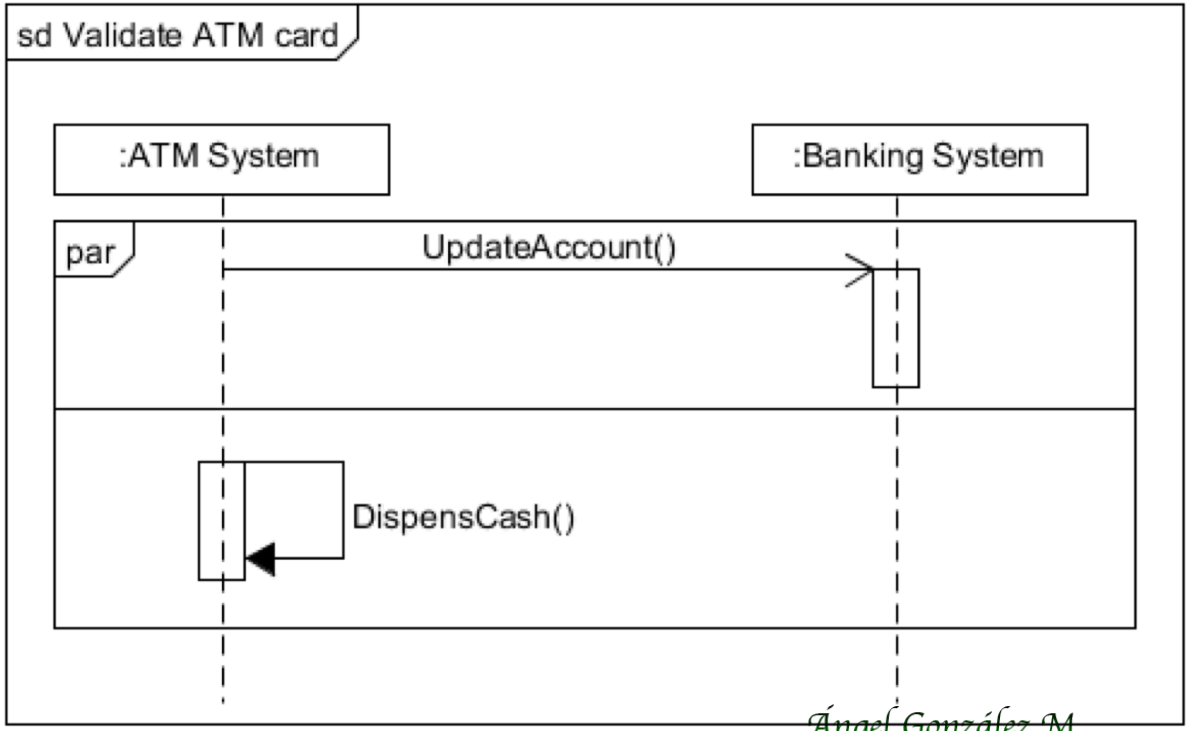


# Parallel (par)

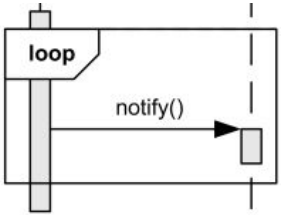
El fragmento Parallel (denotado “par”) modela procesos concurrentes. Es decir los eventos de los fragmentos se pueden intercalar.

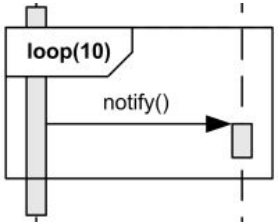


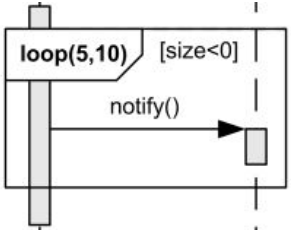



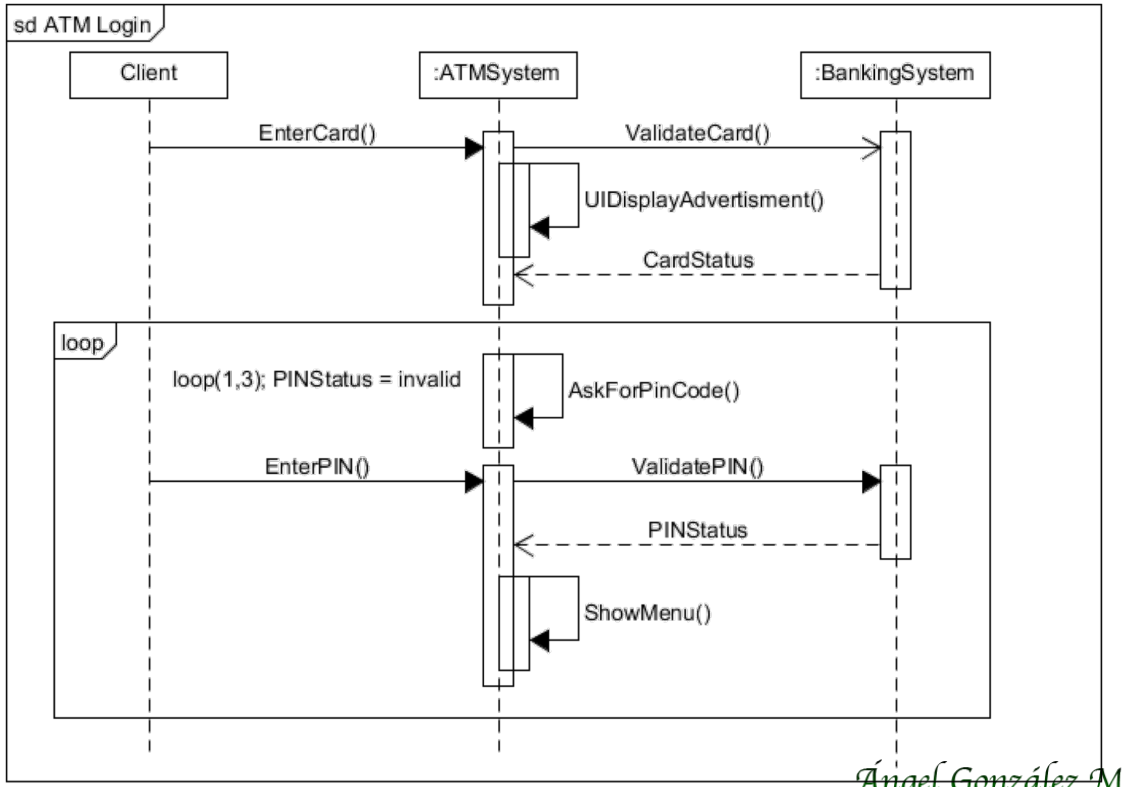
# Loop

El fragmento Loop incluye una serie de mensajes que están repetidos.



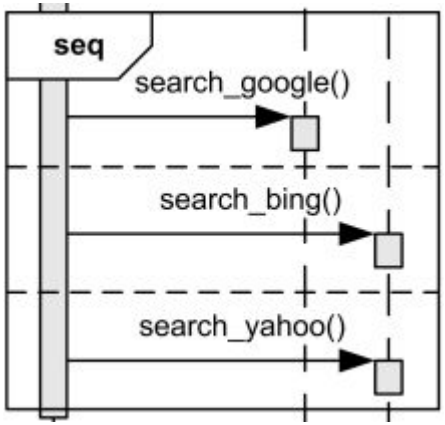



# Weak (seq)

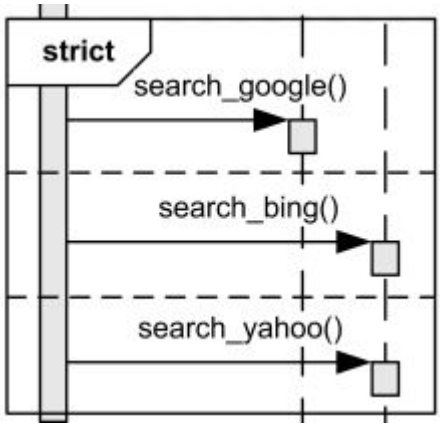
El fragmento de secuenciado Weak (denotado “seq”) incluye un número de secuencias para las cuales todos los mensajes se deben procesar en un segmento anterior, antes de que el siguiente segmento pueda comenzar, pero que no impone ningún secuenciado en los mensajes que no comparten una línea de vida.




- Los mensajes relacionados con la misma línea de vida deben aparecer en el orden de los fragmentos.
- Si no implican las mismas líneas de vida, es posible intercalar en paralelo los mensajes de fragmentos diferentes.

# Strict

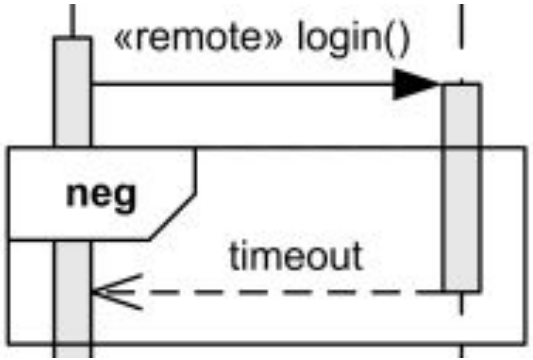
El fragmento de secuenciado Strict (denotado “strict”) incluye una serie de mensajes que se deben procesar en el orden proporcionado.



# Negative (neg)

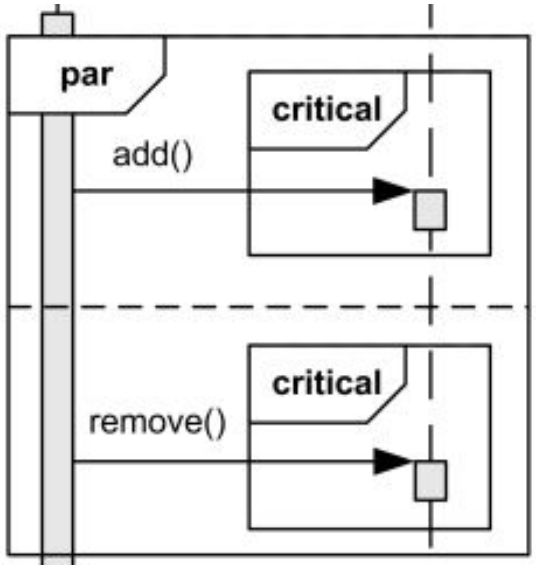
El fragmento Negative (denotado “neg”) incluye una serie de mensajes inválidos.

Suele usarse dentro de un fragmento Consider o Ignore.



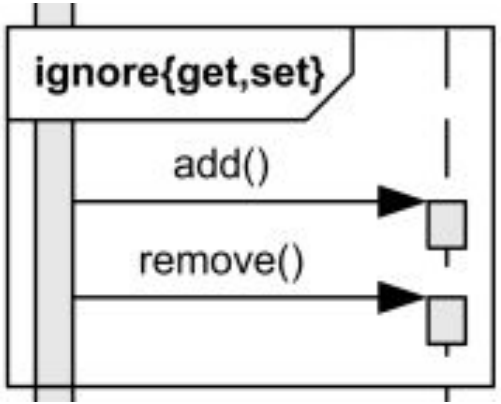
# Critical

El fragmento Critical incluye una sección crítica. Una sección crítica es aquella que no puede ser intercalada con otro proceso concurrente. La operación debe ser atómica. Se usa dentro de un fragmento Par o Seq



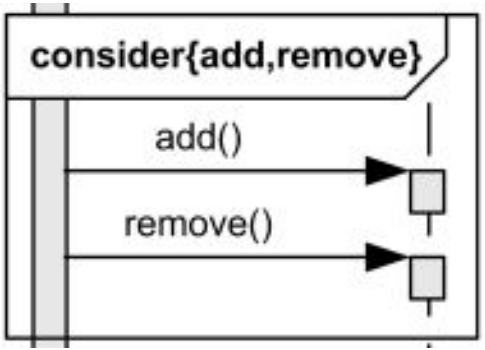
# Ignore

El fragmento Ignore declara un mensaje o mensajes que no son de ningún interés si este aparece en el contexto actual.



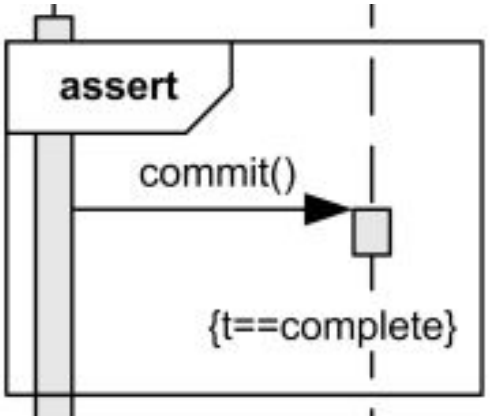
# Consider

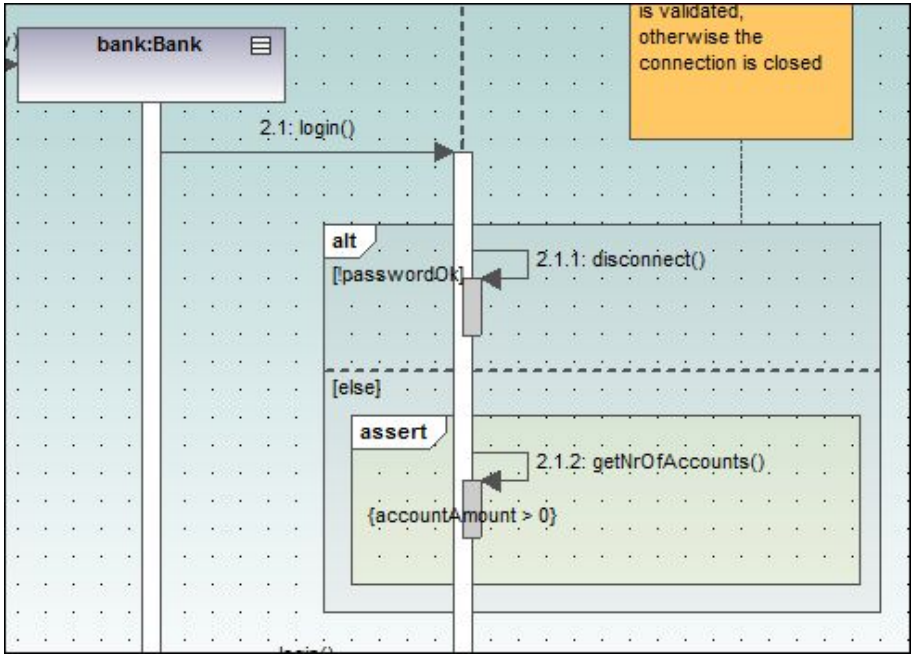
El fragmento Consider es el opuesto del fragmento Ignore: cualquier mensaje que no se incluya en el fragmento Consider se debería ignorar.



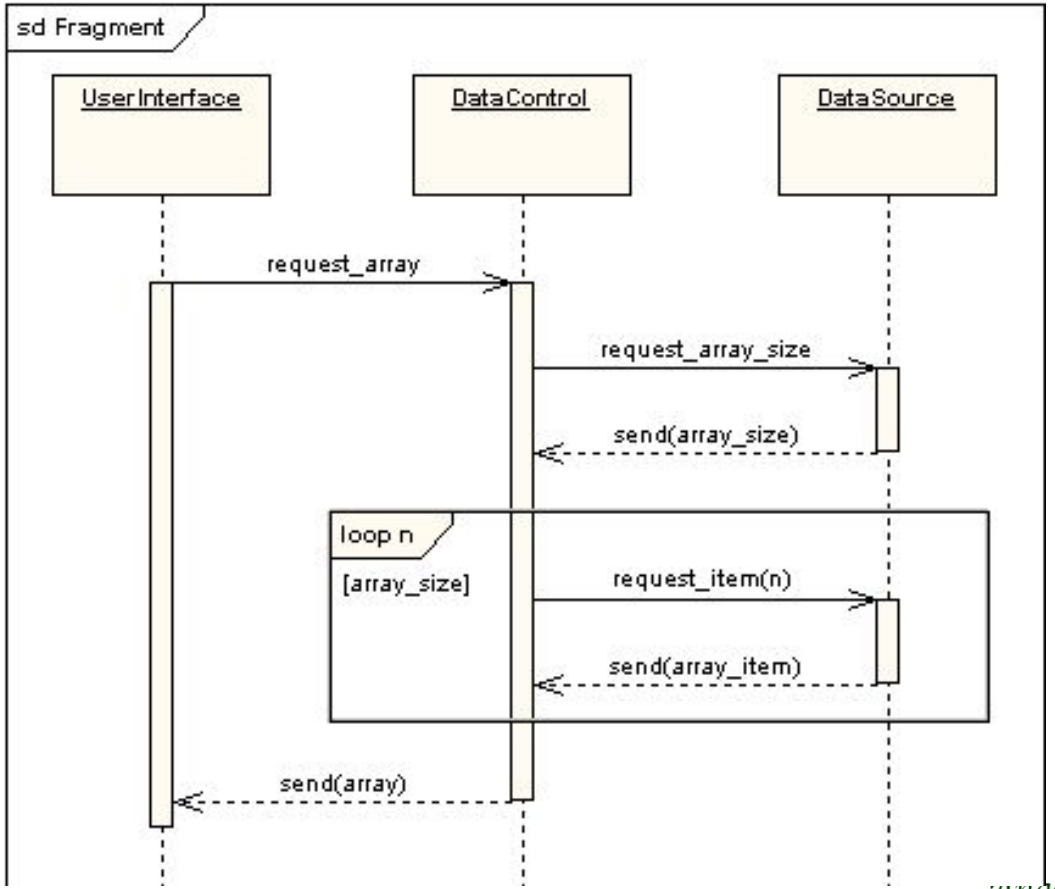
# Assertion (assert)

El fragmento Assertion (denotado “assert”) designa que cualquier secuencia que no se muestra como un operando de la aserción es inválida. Suele usarse dentro de un fragmento Consider o Ignore.

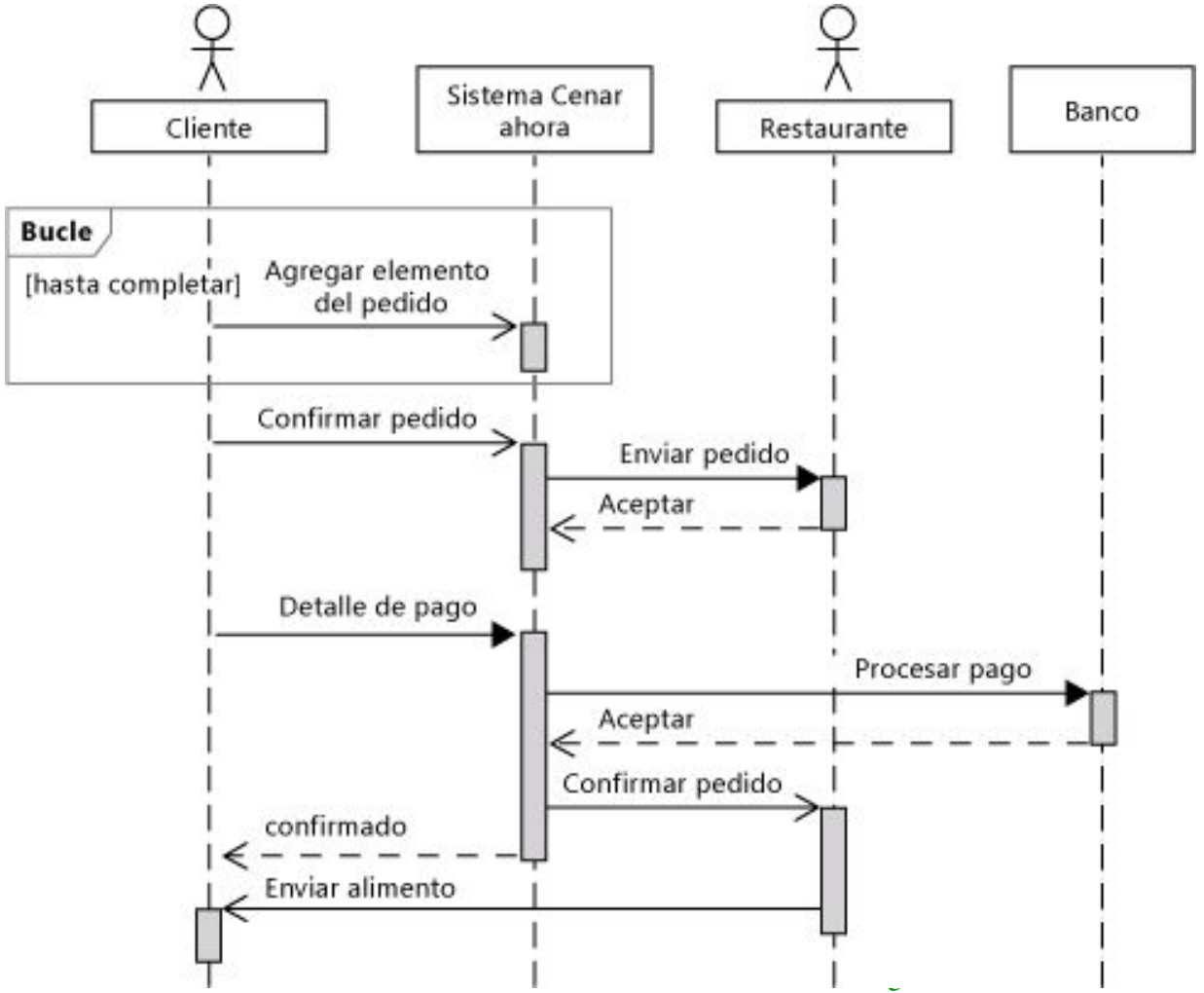


# Ejemplo fragmento combinado

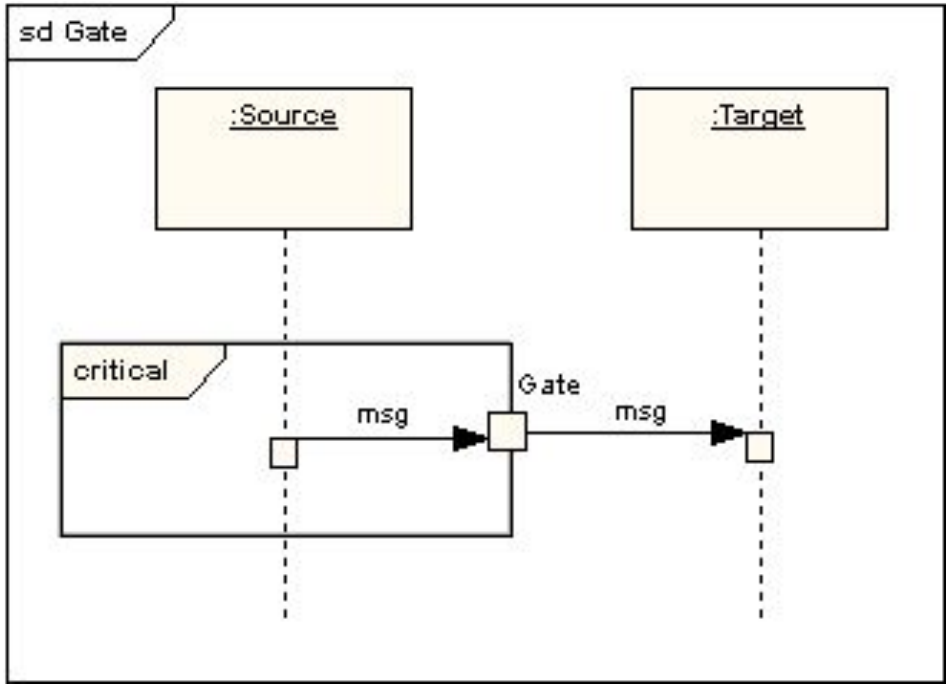


# Ejemplo:


Cliente
Sistema Cenar ahora
Restaurante
Banco
Bucle
[hasta completar]
Agregar elemento del pedido
Confirmar pedido
Enviar pedido
Aceptar
Detalle de pago
Procesar pago
Aceptar
Confirmar pedido
confirmado
Enviar alimento

# Puerta

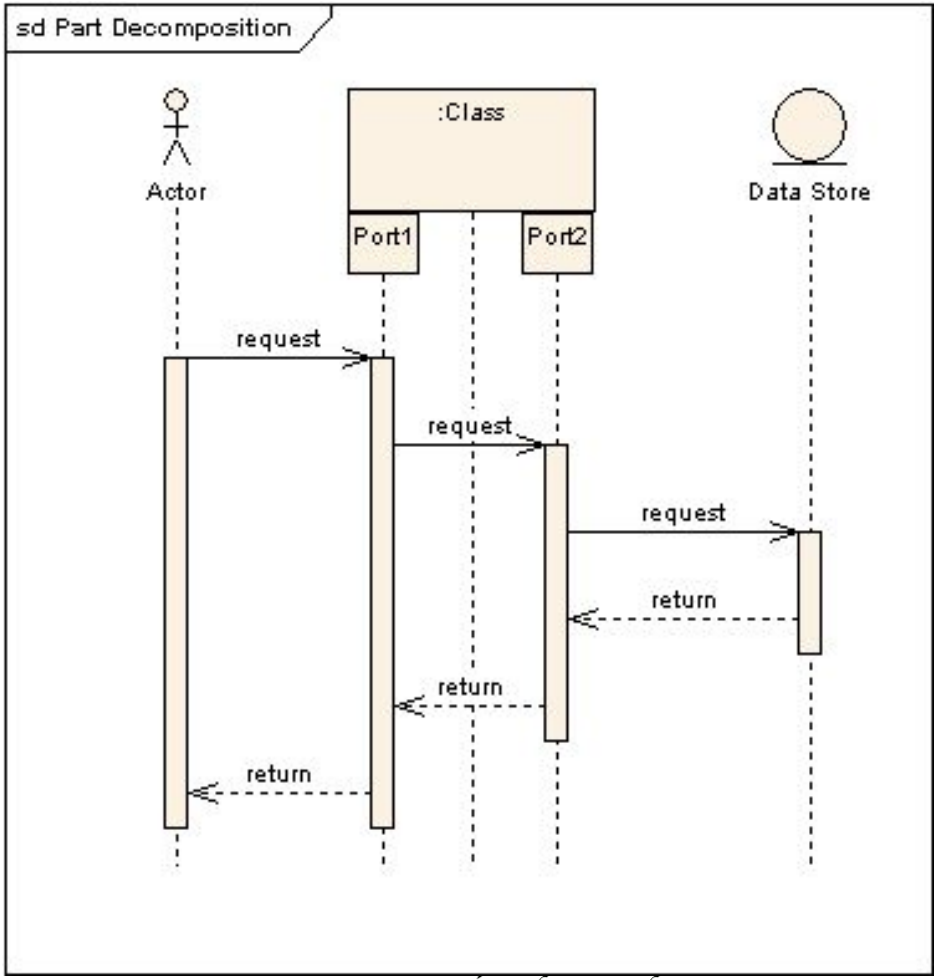
Una puerta es un punto de conexión para conectar un mensaje dentro de un fragmento con un mensaje fuera del fragmento. Esta muestra una puerta como un cuadro pequeño en un marco del fragmento.




Ángel González M.

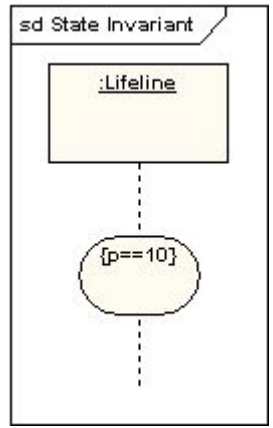
# Descomposición en parte

Un objeto puede tener más de una línea de vida que viene de ésta. Esto permite mensajes de entre e intra objetos para que se muestren en el mismo diagrama.



# Continuaciones / Invariantes de Estado

Una invariante de estado es una restricción ubicada en una línea de vida que debe ser verdadera en el tiempo de ejecución. Esta se muestra como un rectángulo con los extremos en semi-circulos.




Una continuación tiene la misma notación que una invariante de estado pero se usa en fragmentos combinados y puede extenderse a través de más de una línea de vida.