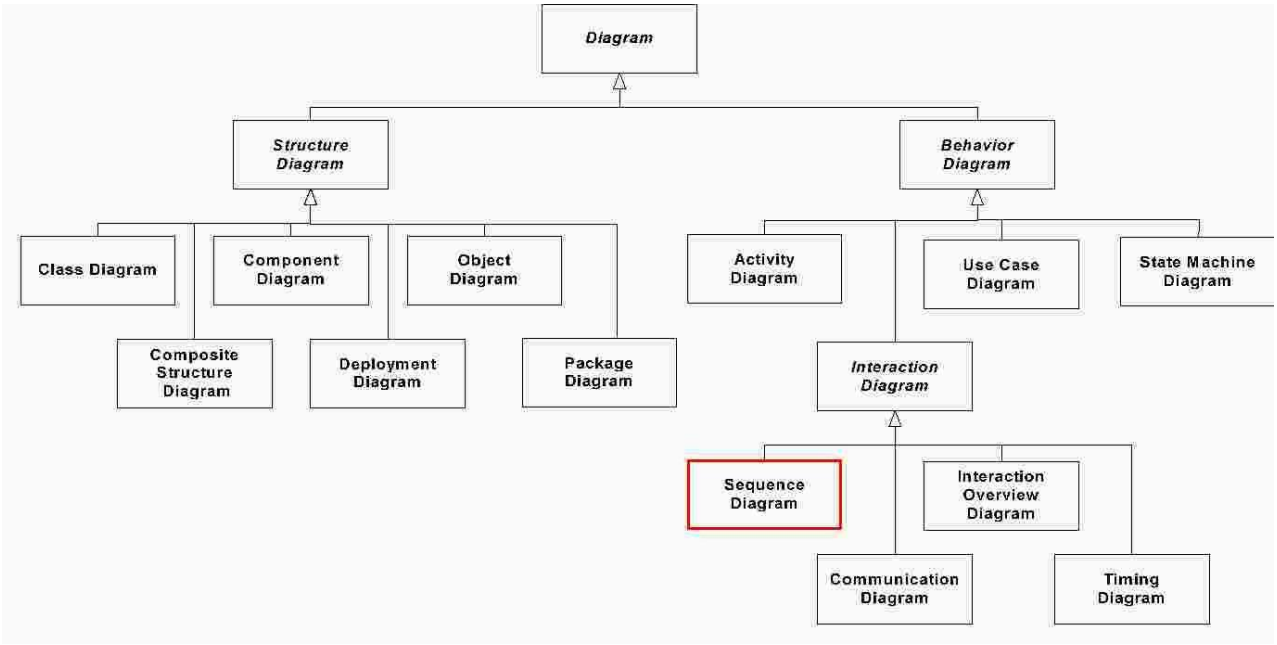
Indice

9 – Diagrammi di interazione 1
9.1 – Elementi dei diagrammi d'interazione 4
9.2 – I diagrammi di sequenza 4
9.3 – Messaggi 5
9.4 – Stati 8
9.5 – Realizzare i casi d'uso 9
9.6 – Operatori 10
9.7 – Conclusioni 13

9 – Diagrammi di interazione

Nella visione d'insieme, i diagrammi di interazione si trovano:



I **diagrammi di interazione** sono diagrammi di comportamento che modellano le interazioni tra varie entità di un sistema.

Visualizzano lo **scambio di messaggi** tra entità nel tempo. Il loro **obbiettivo** è mostrare come un certo comportamento viene realizzato dalla collaborazione delle entità in gioco.

La parola chiave è **realizzare**: questi diagrammi mostrano la realizzazione di un comportamento offerto. In altri termini, mostrano il comportamento del sistema da una **prospettiva interna**.

Come altri diagrammi, possono avere **diversi livelli di astrazione**.

Gli **elementi essenziali** che per costruire un diagramma di sequenza sono due:

1. Un comportamento da realizzare trattato da un classificatore (classificatore di contesto), ad esempio un caso d'uso o un'operazione di una classe;
2. Una serie di elementi che realizzano il comportamento, ad esempio attori o istanze di classe.

Tuttavia, solitamente questi ingredienti provengono da diagrammi creati precedentemente.

Esistono **4 tipi di diagrammi di interazioni**. Vengono presentati per completezza, ma solo il primo (diagramma di sequenza) verrà studiato:

- **Diagramma di sequenza**: adatto a mostrare la sequenza temporale degli avvenimenti per ogni entità del diagramma.
- **Diagramma di comunicazione**: adatto a mostrare le relazioni strutturali e i collegamenti tra le entità e i messaggi che vi transitano.
- **Diagramma di interazione generale**: adatto a mostrare la costruzione di interazioni complesse a partire da interazioni più semplici.
- **Diagramma di temporizzazione**: adatto a mostrare l'evoluzione dell'interazione in tempo reale.

9.1 – Elementi dei diagrammi d'interazione

Nei diagrammi di interazione generalmente non compaiono direttamente classificatori come le classi. Al loro posto ci sono **due elementi**:

- Le **istanze** di classificatori, come oggetti, istanze di attori, ecc.
- Le **linee di vita** (*lifeline*) di classificatori (esprimono ruoli, non specifici oggetti).

La **differenza** tra istanze e linee di vita è molto sottile, ma in linea generale è **preferibili usare le linee di vita**.

Questi diagrammi includono i **messaggi**, elemento principale, ossia i segnali che si scambiano le istanze di classificatori e/o linee di vita.

9.2 – I diagrammi di sequenza

I **diagrammi di sequenza** evidenziano l'ordine temporale delle invocazioni dei metodi. Una linea verticale tratteggiata indica una sequenza temporale. Gli eventi hanno luogo nel loro ordine sulla linea e le distanze sono irrilevanti. Più linee di vita interagiscono per realizzare il comportamento offerto, scambiandosi messaggi.

9.3 – Messaggi

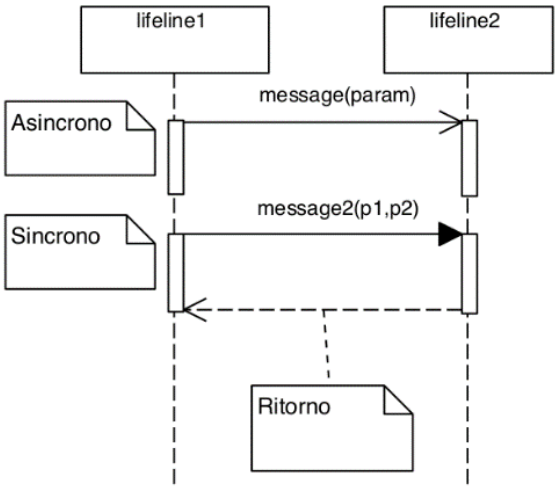
I **messaggi** rappresentano comunicazioni tra linee di vita: segnali, chiamate di operazioni, creazione e distruzione di oggetti.

Esistono **sette tipi** di messaggi:

1. Chiamata sincrona;
2. Chiamata asincrona;
3. Ritorno da chiamata;
4. Creazione;
5. Distruzione;
6. Messaggio trovato;
7. Messaggio perso.

Chiamata e ritorno da chiamata

Esistono due tipi di chiamata: sincrona e asincrona. Nell'immagine viene presentato un esempio e i rettangoli di attivazione, che indicano un focus di controllo, sono opzionali.



I messaggi **sincroni** fanno bloccare il mittente (chi li manda) fino al messaggio di ritorno del destinatario.

I messaggi **asincroni** consentono al mittente di continuare la sua esecuzione poiché esso non si aspetta un messaggio di ritorno.

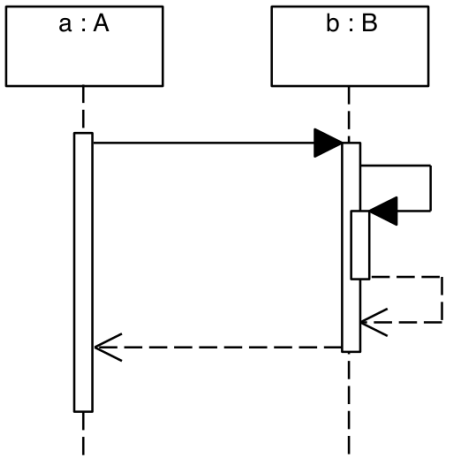
I **messaggi di ritorno** sono **opzionali** e generalmente inclusi solo quando non ostacolano la leggibilità del diagramma oppure per segnalare valori di ritorno.

La distinzione tra messaggi sincroni e asincroni in genere emerge **in fase di progettazione**.

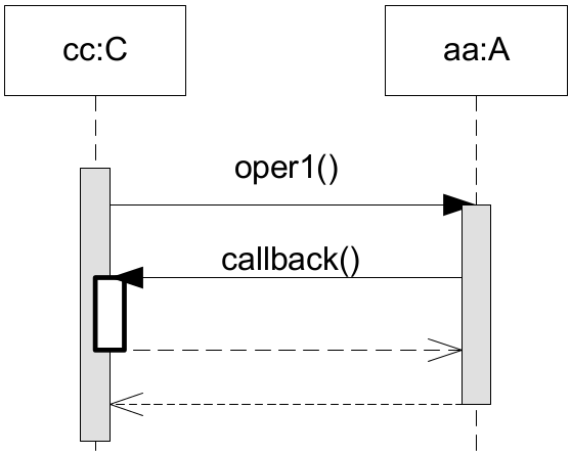
In fase di analisi, conviene indicare tutti i messaggi come sincroni (è il caso più vincolante) e includere il nome del messaggio. Nel caso in cui si desiderasse maggiore dettaglio, si potrebbe includere una lista di **parametri** tra parentesi e utilizzare **guardie** per verificare condizioni. Infine, si possono indicare i parametri formali, attuali, o entrambi. Per esempio:

$$getArea(shape = g)$$

Solitamente, le linee di vita possono mandare messaggi a sé stesse, per esempio con l'invocazione di operazioni private. Questa tecnica si chiama **attivazioni annidate**.



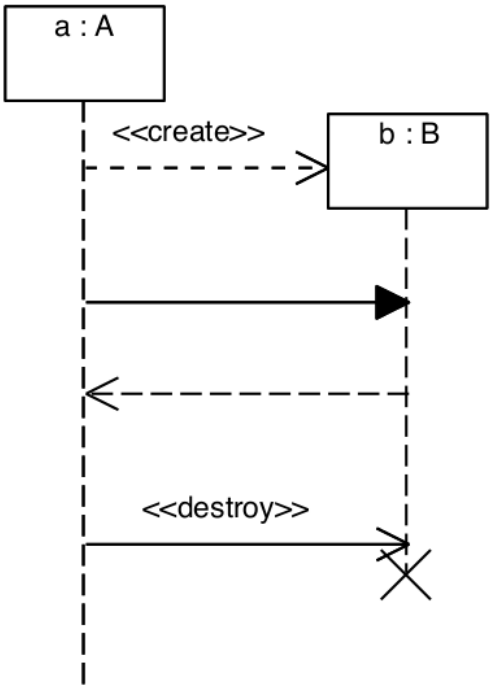
Le attivazioni annidate possono generare altre attivazioni sulla linea vita chiamante (esempio una *callback*).



Creazione e distruzione

Nel **messaggio di creazione** lo stereotipo può essere seguito dal nome di un'operazione e relativi parametri (costruttore). Questo diventa necessario nel caso si lavori con linguaggi di programmazione senza costruttori espliciti.

Molti linguaggi di programmazione Objected Oriented gestiscono la **distruzione** in modi diversi: esplicita o garbage collector. Durante la **fase di analisi** basta immaginare che dopo la distruzione l'oggetto non sia più accessibile.

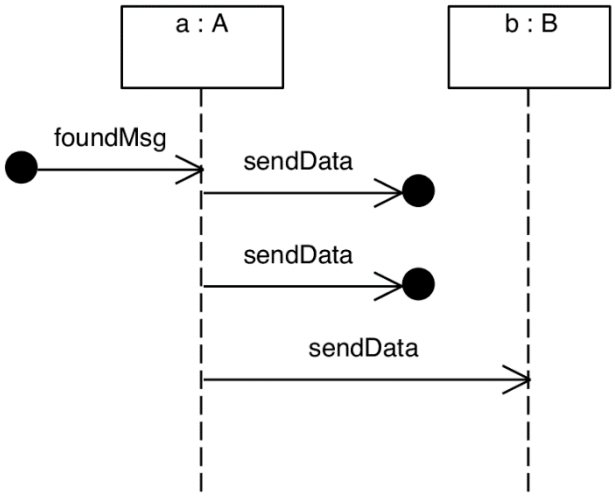


Messaggi trovati/persi

Questi tipi di messaggi vengono usati raramente **in fase di analisi**.

I **messaggi trovati** indicano situazioni in cui il mittente è sconosciuto al momento della ricezione, proviene dall'esterno del diagramma o i dettagli della provenienza non interessano.

I **messaggi persi** permettono di visualizzare il comportamento nel caso un messaggio non giunga a destinazione (situazione di errore).



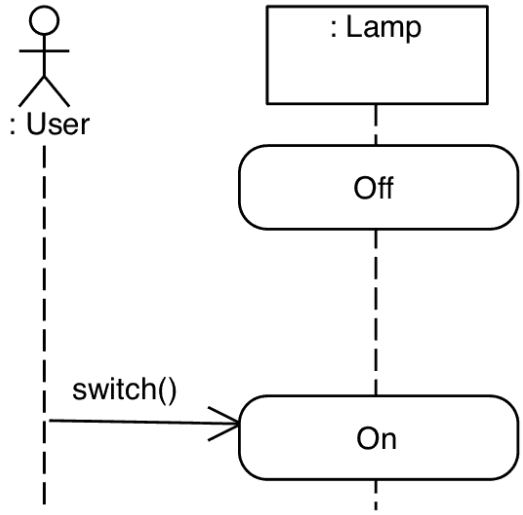
9.4 – Stati

Uno **stato** è definito come una condizione o situazione durante la vita di un oggetto in cui esso soddisfa una condizione, esegue un'attività o aspetta un evento.

Ogni oggetto in UML può avere una macchina a stati associata che ne descrive gli stati possibili.

I **diagrammi di stato** (in inglese "state machine diagram") sono i più adatti a rappresentare gli stati e loro transizioni, ma può essere conveniente mostrare alcuni cambiamenti di stato nei diagrammi di sequenza.

In generale, ci si limita agli stati principali all'oggetto mettendo in risalto i messaggi che provocano un cambiamento di stato.



In UML, gli stati si rappresentano con rettangoli arrotondati. I cambiamenti di stato sono generalmente la conseguenza di uno o più messaggi.

9.5 – Realizzare i casi d'uso

Le primitive introdotte nei paragrafi precedenti consentono di rappresentare sequenze di eventi senza **ramificazioni**. Tuttavia, per realizzare i casi d'uso è necessario poter descrivere sequenze più complesse.

Le sequenze degli eventi in un caso d'uso contengono parole chiave come if, then, else, for e while.

I diagrammi di sequenza permettono di tradurre questi costrutti ed inserirli nel diagramma.

Si ricorre dunque ai **frammenti combinati**.

Per **frammento combinato** si intende una sottoarea di un diagramma di sequenza che racchiude una parte dell'interazione e le modalità della sua esecuzione.

Gli elementi di un frammento combinato sono:

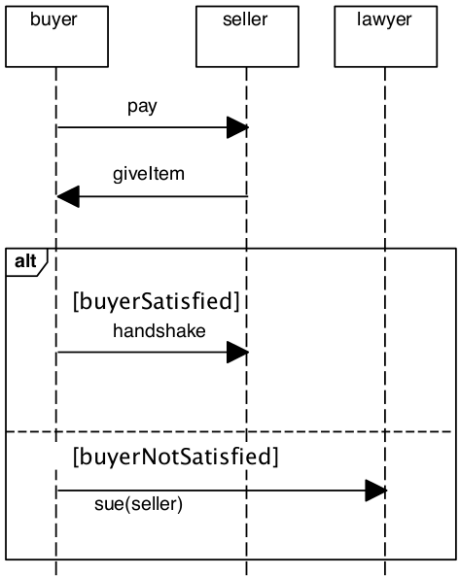
- Un **operatore**, che specifica come il frammento viene eseguito;
- Uno o più **operandi**, sottoaree del diagramma di sequenza che possono essere eseguite;
- Zero o più **condizioni di guardia**, espressioni che determinano quali operandi sono eseguiti.

La **sintassi** UML per un frammento combinato è la seguente:

- Un **rettangolo con il nome** dell'**operatore** in alto a sinistra racchiuso nel simbolo di diagramma (rettangolo con angolo tagliato);
- Gli **operandi** seguono come **strisce orizzontali** in sequenza, separati da linee tratteggiate;
- La **guardia**, se presente, compare dopo l'operatore oppure nella parte alta di un operando tra **parentesi quadre**.

Il rettangolo del frammento combinato si estende orizzontalmente per includere le linee di vita interessate dall'interazione.

La guardia può includere qualunque tipo di condizione booleana. Per esempio:



9.6 – Operatori

L'**operatore** determina la semantica dell'esecuzione del frammento.

UML specifica parecchi **tipi di operatori**, ma solo alcuni sono usati frequentemente, ovvero quelli che corrispondono alle primitive di controllo del flusso definite dai linguaggi di programmazione.

I più utilizzati sono:

- **opt** corrisponde a if/then;
- **alt** corrisponde a case/select;
- **loop** corrisponde a for/while;
- **break** corrisponde a break.

Nel dettaglio "**opt**" accetta solo un operando, che viene eseguito se e esolo se la guardia è valutata "true".

Invece, "**alt**" accetta un qualunque numero di operandi e ne esegue al massimo uno. Il suo funzionamento nello specifico:

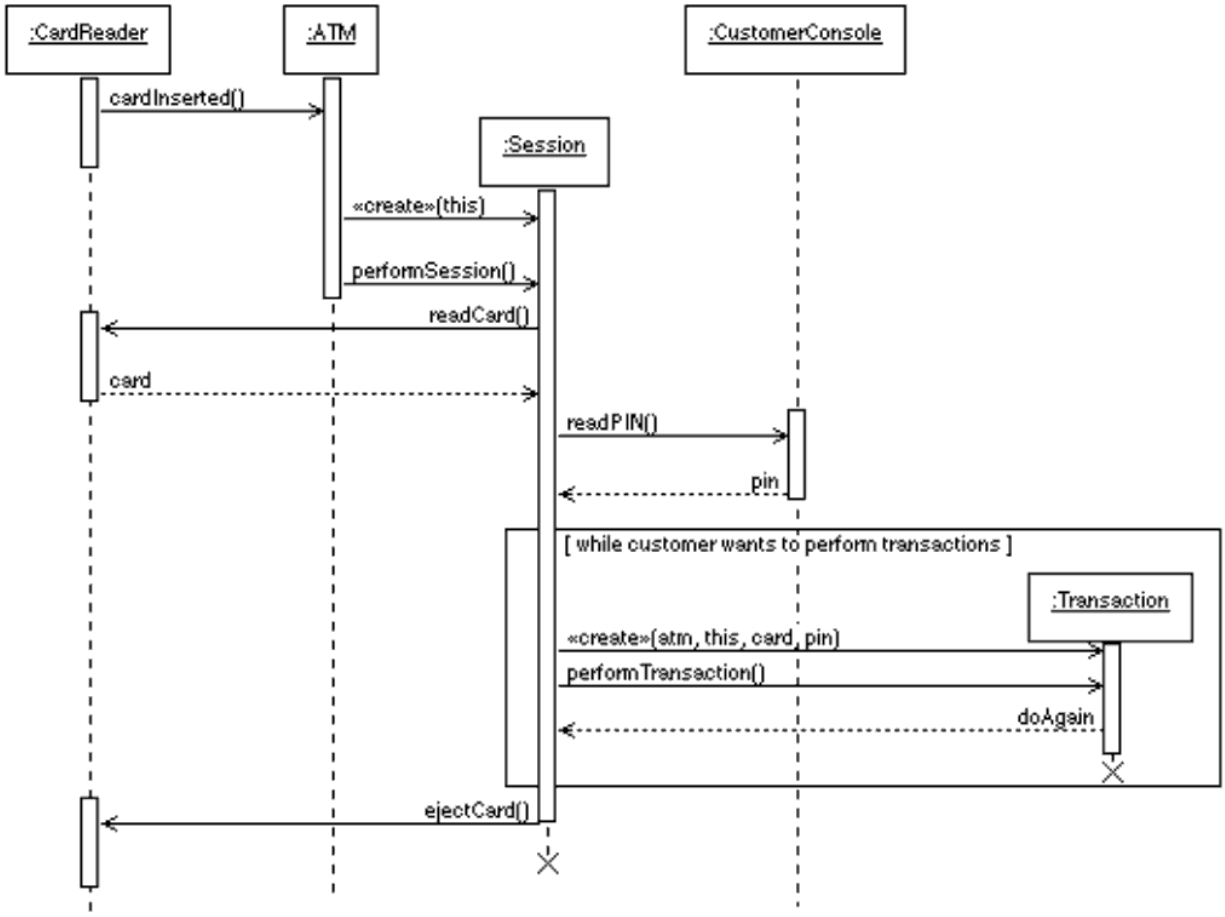
1. Esamina la lista a partire dal primo operando, valuta le guardi ed esegue solo il primo operando la cui guardia è vera;
2. L'operando opzionale [else] è eseguito se nessuna delle guardie è vera;
3. Le condizioni di guardia devono essere **mutualmente esclusive**; al massimo una può essere vera nello stesso istante, altrimenti il modello non è corretto.

L'operando "**loop**" può essere spiegato con un esempio:

$$\text{loop } min, max[\text{condizione}]$$

Viene eseguito l'operando min volte, e poi al massimo altre $(max - min)$ volte mentre la condizione è vera. È possibile omettere i valori min e max. La condizione è molto flessibile poiché può essere espressa in linguaggio naturale e consente di ciclare su un insieme di oggetti.

Per esempio, la condizione [forEach oggetto in lista].



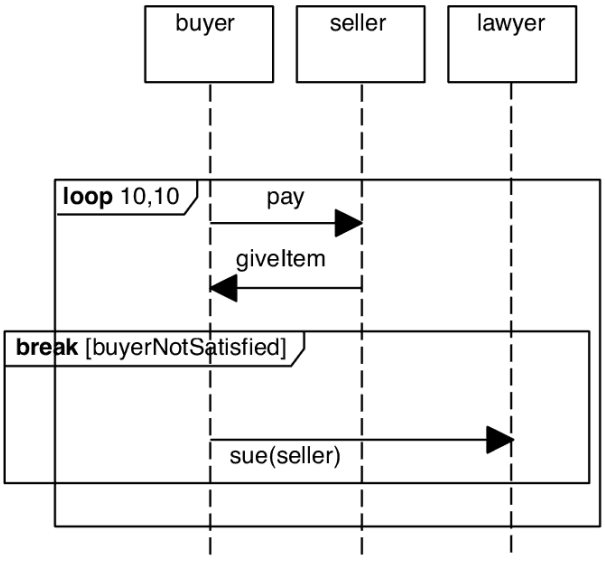
L'operatore **break** utilizzato singolarmente, viene adottato per terminare l'esecuzione del frammento di interazione corrente (spesso si tratta di un operatore **loop**).

Se **break** ha una condizione di guardia ed è:

- ✓ Vera, viene eseguito l'operando ma non il resto dell'interazione dopo il frammento **break**.
- × Falsa, non viene eseguito l'operando e si continua normalmente.

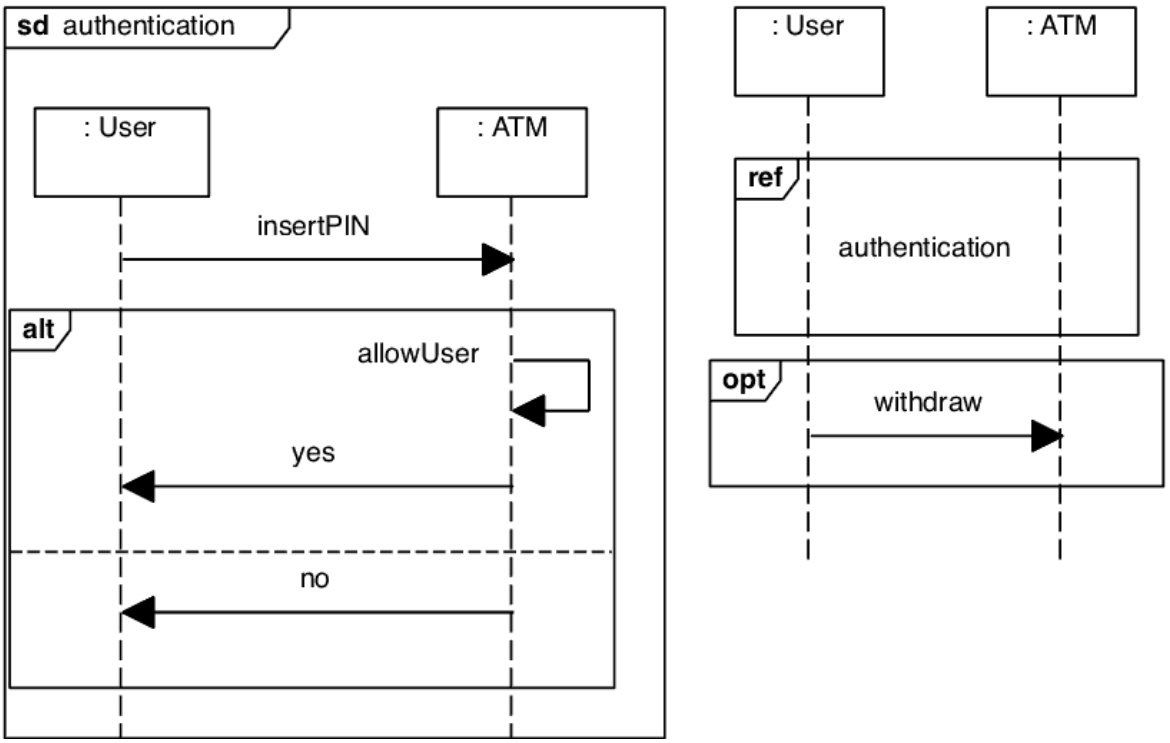
Se non c'è una guardia, la scelta tra continuare e saltare è non-deterministica.

Il frammento **break** deve essere globale rispetto a quello che interrompe: nella notazione UML, lo interseca ma si trova ad un livello di nesting superiore. Un esempio:



Esistono anche altri operatori: **ref** e **gate**.

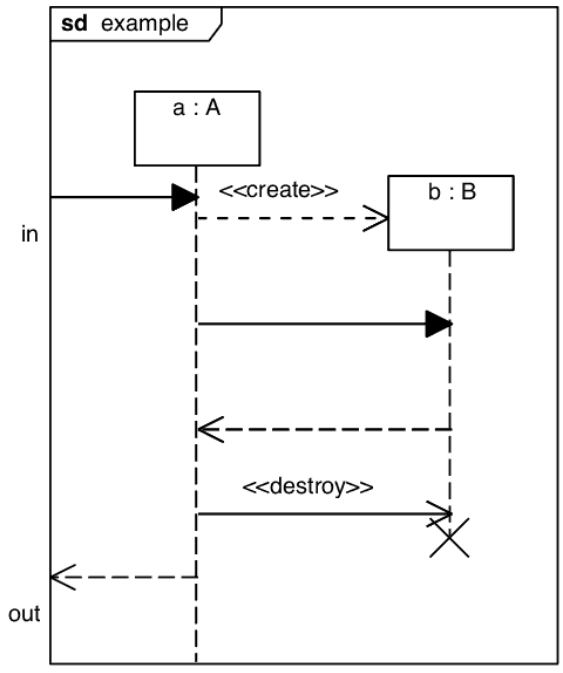
Per rendere i diagrammi di sequenza modulari su utilizza l'operatore **ref**. Esso contiene il nome di un'altra interazione che viene eseguita qui. Per esempio:



Invece, l'operatore **gate** è un qualunque punto sulla cornice esterna del diagramma, che può essere la fonte o la destinazione di una freccia. Essi vengono usati per modella la comunicazione delle linee di vita del diagramma con l'esterno.

I gate possono avere un nome e comparire qualunque numero di volte nello stesso diagramma o in diagrammi diversi.

Possono essere usati in combinazione con ref, forniscono l'interfaccia di collegamento tra l'interazione chiamante e quella chiamata (il frammento combinato può avere gli stessi gate a cui agganciare messaggi). Per esempio:



9.7 – Conclusioni

I **digrammi di interazione** vengono **utilizzati** per mostrare **come** varie **entità collaborino** nel fornire un comportamento desiderato, ad esempio un caso d'uso.

Possono essere **usati** sia in **fase di analisi** che in **fase di progettazione**.

Aiutano a **scoprire** e **correggere inconsistenze** nel comportamento desiderato (ad esempio una specifica scorretta del caso d'uso).

Spesso è **utile modellare** una situazione semplificata con un diagramma di comunicazione, per poi passare a uno di sequenza per i dettagli.