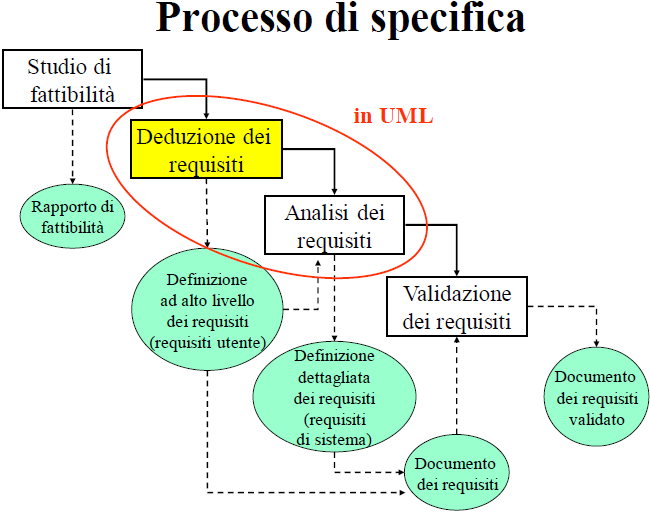
**DEDUZIONE DEI REQUISITI**

**UML:** È un linguaggio di progettazione usato per documentare e progettare software orientati agli oggetti. **(Unified Modeling Language)**



Nella deduzione dei requisiti si compie un’indagine per “dedurre” i requisiti. Si stabiliscono i requisiti raccogliendo informazioni da:

**1)Studio del dominio applicativo del sistema.**

**2)Dialogo con stakeholder** (soggetto coinvolto direttamente o indirettamente in un’attività).

**3)Studio di sistemi già realizzati e con lo stesso dominio applicativo o con un dominio simile.**

**4)Studio dei sistemi con cui dovrà interagire quello da sviluppare.**

**Dominio applicativo**: è un insieme di **oggetti** reali a cui il sistema software viene applicato. Un dominio può essere visto come un insieme di classi.

Il dominio definisce il contesto in cui opera il sistema:

1)Ogni oggetto del dominio è caratterizzato da **attributi**.

2)Alcuni oggetti hanno gli stessi attributi e quindi possono essere raggruppati in sottoinsiemi del dominio detti **classi**.

**Modellazione del dominio con il Diagramma delle classi**

Le classi del dominio corrispondono alle classi del diagramma. In tale diagramma si possono rappresentare gli attributi delle classi, le relazioni tra classi e si possono generalizzare classi con attributi in comune.

**Nodo classe**: che rappresenta una classe è composto da tre parti

1)Nome della classe

2)Attributi

3)Operazioni

**Archi**: rappresentano le relazioni tra classi.

**Tipi di relazione**

**1)Associazione:** quando tra due classi esiste una relazione logica. Per esempio, il CLIENTE effettua ORDINE oppure UTENTE possiede FOTO.

**Notazione grafica:** è un arco semplice con etichetta che descrive l’associazione.

**2)Generalizzazione:** una classe può specificare un caso specifico di un’altra. In tal caso una classe eredita attributi e metodi da un’altra. Per esempio, il LIBRO che è una sottoclasse eredita da BENE detta superclasse tutti gli attributi e i metodi.

**Notazione grafica:** è un arco orientato dalla sottoclasse verso la superclasse e tale arco termina con un triangolo bianco chiuso che punta la classe madre.

**3)Composizione**: un oggetto di una classe contiene oggetti di altre classi. Ovvero la composizione si usa quando una classe non può esistere senza un’altra classe. Per esempio, la classe CATALOGO contiene la classe BENE, perciò, la classe CATALOGO non può esistere senza la classe BENE. Tale relazione è detta anche composizione forte.

**Notazione grafica:** è un arco che collega la classe contenitore (CATALOGO) alla classe contenuta (BENE). Un rombo nero punta la classe contenitore.

**4)Aggregazione:** tale relazione dice che una classe può esistere anche senza l’altra. In pratica il contrario di composizione. Tale relazione è detta anche composizione debole.

**Dichiarazione della visibilità e del tipo degli attributi**

**Tipi di visibilità** di un attributo e di un metodo:

**1)Public (+):** visibile da tutte le classi.

**2)Private (-):** visibile solo dalla classe in cui è definito.

**3)Protected(#):** in tal caso metodi e attributi sono visibili a tutte le sottoclassi, all’interno o meno dello stesso package.

Esiste anche il default praticamente mai utilizzato.

**Tipi di attributo:**

**1)I tipi di attributo possono essere primitivi**: integer, float, string, Boolean, …,

**2)Oppure possono essere classi che si stanno definendo**. In questo caso si utilizza un **arco di dipendenza**(orientato e tratteggiato).

**Notazione:** La classe dove il tipo è usato -----> classe dove il tipo è definito.

Esempio: Persona ---> Data significa che Persona dipende da Data perché Persona usa Data come tipo di attributo (la data di nascita della persona).

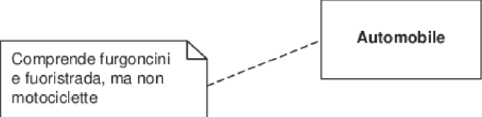
**Molteplicità di un attributo**

–Serve per dichiarare che un attributo è un array.

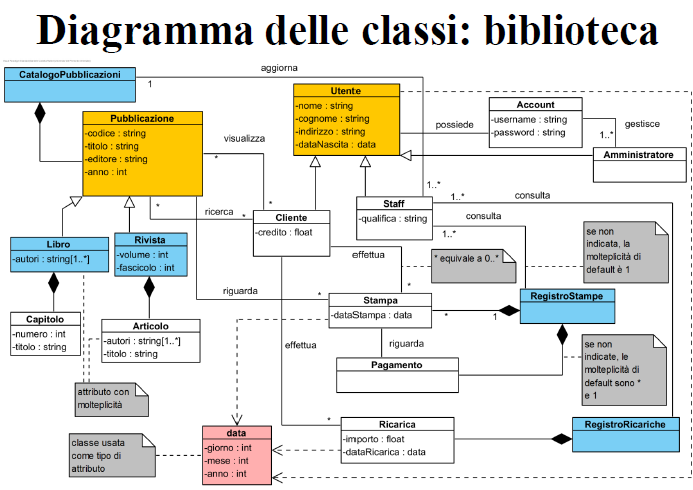
–Si esprime come la molteplicità delle relazioni tra classi.

**Note (commenti)**

In ogni tipo di diagramma UML è possibile scrivere delle note (commenti) e collegarle ai nodi o agli archi a cui si riferiscono. Un arco tratteggiato collega la nota al nodo/arco. La nota può essere scritta in linguaggio naturale o in Object Constraint Language (OCL).



**Esempio:** Supponiamo di dover realizzare una biblioteca digitale che contiene un archivio di pubblicazioni in formato digitale e che consente agli utenti registrati di visualizzare (gratuitamente) e stampare (a pagamento) le pubblicazioni.



**Deduzione dei requisiti dal dominio**

Le classi del dominio e le relazioni tra di esse possono fornire informazioni per dedurre i requisiti. In base al dominio “Biblioteca digitale”, possiamo dedurre i seguenti requisiti funzionali della biblioteca digitale:

**1)L’utente deve essere registrato**. Quindi:

•Creazione account (tramite un amministratore?)

•Accesso al sistema tramite login e password

**2)L’utente può visualizzare gratuitamente una pubblicazione**. Quindi:

•Ricercare una pubblicazione in base a titolo o autore

•Visualizzare una certa parte della pubblicazione (se questa è composta da molte pagine)

**3)L’utente può stampare a pagamento la parte visualizzata**. Quindi:

•Effettuare pagamento

•Gestire un credito pre-pagato

**4)Lo staff può gestire le pubblicazioni**. Quindi:

•Accedere al sistema

•Aggiungere una pubblicazione all’archivio

•Rimuovere una pubblicazione dell’archivio

**5)Qualcuno dovrebbe gestire gli utenti**. Quindi:

•Un amministratore potrebbe creare gli account.

**6)Il sistema potrebbe registrare tutte le stampe effettuate.**

•Lo staff può vedere l’elenco delle stampe.

**7)Il sistema potrebbe archiviare i pagamenti/ricariche effettuate.**

•Lo staff può vedere l’elenco dei pagamenti/ricariche

**Modellazione dell’insieme dei requisiti dedotti**:

L’insieme dei requisiti funzionali dedotti dal dominio si può rappresentare tramite un Diagramma dei casi d’uso.

**Il caso d’uso è un requisito funzionale ovvero un servizio del sistema.**

–Si può associare il caso d’uso con il tipo di utente coinvolto quindi fuori dal nostro sistema ma con cui il sistema dovrà interagire (attore). Definito con un omino.

–Si può associare il caso d’uso con un sistema esterno coinvolto.

–Si possono stabilire dipendenze tra i casi d’uso.

–Si possono generalizzare attori con casi d’uso in comune.

**Diagramma dei casi d’uso**

-Tipi di nodo:

**1)Caso d’uso**: indica un requisito funzionale (servizio) del sistema. Viene indicato con un’elisse con al suo interno il nome del servizio.

**2)Attore**: entità esterna che interagisce con il sistema (utente o un altro sistema). Viene indicato con l’omino.

**Associazione tra attore e caso d’uso**: indica che l’attore interagisce con il sistema durante l’esecuzione del caso d’uso.

**Notazione grafica:** arco continuo che collega l’attore al caso d’uso.

Ad esempio, segretario può fare inserisci appello.

**Dipendenze (----->) tra casi d’uso**:

**1)Include**: in questo caso l’esecuzione di un caso d’uso include necessariamente un altro caso d’uso. Per esempio, la stampa di un articolo <<include>> Pagamento

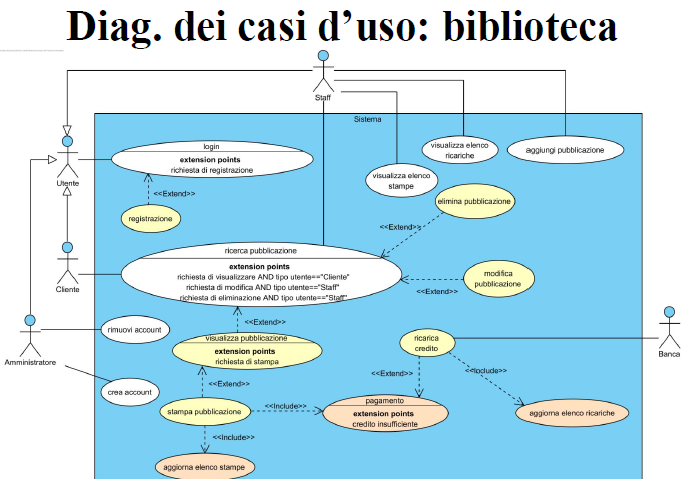
–Ogni volta che si stampa si deve per forza pagare. (vedi diagramma casi d’uso sotto). L’arco va dal caso d’uso primario verso il secondario perché diciamo che il caso d’uso primario include il secondario.

**2)Extend**: un caso d’uso può estendere un altro caso d’uso se vale una certa condizione o si verifica un certo evento (*extension point*).

Per esempio, la Carica del credito <<extend>> Pagamento, infatti quando si paga, potrebbe essere necessario ricaricare il credito se questo è insufficiente. L’arco va dal caso d’uso secondario verso il primario perché diciamo che il caso d’uso secondario estende il primario.

**3)Relazioni tra attori**: generalizzazione ovvero un attore può ereditare i casi d’uso di un altro attore.

**4)Boundary**: riquadro che separa i casi d’uso del sistema dagli attori ovvero separa ciò che fa parte del sistema da ciò che non ne fa parte. Nello schema sotto viene rappresentato dal riquadro azzurro in cui dentro ci sono i servizi e fuori ci sono gli attori.



Ogni singolo caso d’uso (requisito funzionale) si può modellare in termini di stati e transizioni tra stati che si verificano nel sistema durante l’esecuzione del caso d’uso.

**Che cos’è uno stato?**

Uno stato è una configurazione che dura per un certo intervallo di tempo durante il quale il sistema:

1)si trova in una particolare condizione;

2)sta svolgendo una o più attività;

3)è in attesa di un evento.

La durata dello stato è delimitata da due eventi: l’ingresso nello stato e l’uscita dallo stato e il passaggio da uno stato ad un altro è detto **transizione di stato**.

Lo **stato** nel diagramma è rappresentato come un nodo con due componenti:

1)Nome

2)Attività interne allo stato

Gli **archi di Transizione** rappresentano le transizioni di stato (passaggio da uno stato ad un altro). Attraverso un’etichetta sull’arco si indica la causa della transizione. Le cause possono essere varie:

1)Condizione

2)Periodo di tempo

3)Evento

**NOTA: È presente uno stato iniziale e ci possono essere n>=0 stati finali**

–**Stato iniziale:** non ha archi entranti provenienti da altri stati.

–**Stato finale**: non ha archi uscenti verso altri stati.

**Le attività all’interno di uno stato**

**1)Attività di entrata**: attività svolta nel momento in cui si entra nello stato.

Notazione: *entry / descrizione attività*

**Molto importante** non è l’evento o la condizione di ingresso nello stato.

**2)Attività di uscita**: attività svolta nel momento in cui si esce dallo stato.

Notazione: *exit / descrizione attività*

**Molto importante** non è l’evento o la condizione di uscita dallo stato.

**3)Do-activity**: attività svolta per tutta la durata dello stato

**Macrostato**

Un macrostato è uno stato che contiene al suo interno degli stati e delle transizioni. Gli stati e le transizioni interne possono essere rappresentati in un diagramma degli stati separato.

Il macrostato può essere usato come riferimento ad un altro diagramma degli stati. **Quindi viene usato come metodo per indicare che passiamo dal diagramma degli stati al diagramma dei casi d’uso.**

Si applica quando un caso d’uso invoca (include/extend) un altro caso d’uso. Graficamente appare come uno stato con un simbolo particolare all’interno.

**Corrispondenze tra Diagramma dei casi d’uso e Diagramma degli stati**

Se nel diagramma dei casi d’uso, A<<*include*>>B oppure B<<*extend*>>A, allora il diagramma degli stati di A deve contenere:

1)Un macrostato B.

2)Un arco di transizione da uno stato di A verso B.

3)Se necessario, un arco da B verso uno stato di A.

Se A<<*include*>>B allora la transizione da uno stato di A al macrostato B deve avvenire prima o poi.

Se A<<*extend*>>B la transizione da uno stato di A al macrostato B avviene solo in situazioni particolari.

**MODELLI DI PROGETTAZIONE Strutturazione, controllo e deployment**



**Diagramma dei componenti**

Rappresenta i sottosistemi e i moduli interni che compongono un’architettura. **Elementi:**

**1)Componente**: nodo che rappresenta un sottosistema o un modulo.

Graficamente appare come un rettangolo

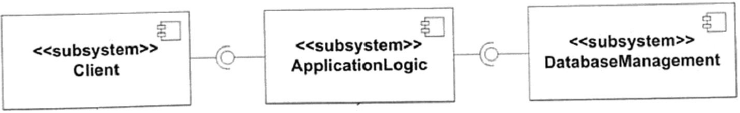
**2)Interfaccia**: rappresenta il fatto che un componente richiede l’esecuzione di operazioni da parte di un altro componente.

Graficamente appare come la coppia di archi **“ball & socket”**

**Ball:** indica che un componente mette a disposizione dei servizi ed esegue delle operazioni.

**Socket:** indica che un altro componente richiede (invoca) tali operazioni.

**Esempio diagramma dei componenti**



Da questo esempio deduco che il client richiede delle operazioni dall’application logic mentre quest’ultimo mette a disposizione dei servizi e li esegue per il client. Stessa logica e da applicare per gli altri due componenti application logic e database management connessi dall’interfaccia.

**Diagramma dei componenti: porte**

Una coppia di componenti potrebbe avere più di una interfaccia.

–Le interfacce potrebbero avere lo stesso verso o versi opposti.

–Le interfacce possono essere distinte

•Assegnando un’etichetta ad ogni interfaccia;

•Indicando i dati o le operazioni che le riguardano.

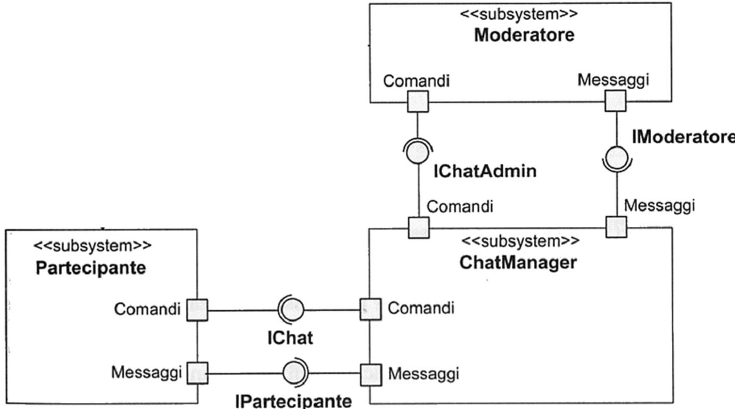
Uso di **porte**

🡪 Simbolo di forma quadrata collocato sul componente e toccato da un arco di tipo ‘‘ball’’ o ‘‘socket’’.

🡪 Il nome della porta indica il dato o l’operazione associata all’interfaccia.

🡪 Le porte relative alla stessa interfaccia hanno lo stesso nome.

**ESEMPIO DI DIAGRAMMA DEI COMPONENTI**



**Diagramma di deployment per la distribuzione fisica del sistema**

**1)Nodo**: elemento in grado di eseguire software

–2 tipi di nodo:

* 1. **Dispositivo**: dispositivo hardware in grado di eseguire software.

–Computer, cellulare, palmare, ...

* 1. **Ambiente di esecuzione**: software in grado di eseguire altro software.

–Browser, macchina virtuale, ...

I nodi contengono

•**Componenti**

•**Elaborati (artifact)**:

–File eseguibili: .exe, .dll, .jar, .class, script, ...

–File di dati, file di configurazione, pagine html, ...

Le **etichette** indicano caratteristiche del nodo.

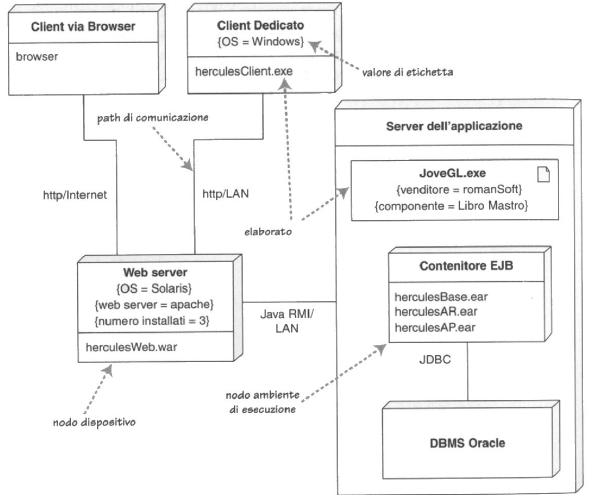
I nodi possono essere annidati.

–Un dispositivo può contenere un ambiente di esecuzione.

•**Path di comunicazione**: arco tra due nodi che indica la comunicazione tra due nodi;

Esso viene indicato tramite il protocollo, la tecnica o il mezzo di comunicazione. Degli esempi sono: •http, TCP/IP, RMI, socket, ethernet, LAN, internet, ...

**ESEMPIO DIAGRAMMA DI DEPLOYMENT**



**DAL DIAGRAMMA DEI COMPONENTI AL DIAGRAMMA DELLE CLASSI**

Il diagramma dei componenti è poco descrittivo: si può ‘‘tradurre’’ in un diagramma delle classi per definire più dettagli:

–Ogni modulo diventa una classe.

–Ogni interfaccia diventa ad un arco di dipendenza.

A -----------> B

🡪A è il componente con ‘‘socket’’

🡪B è il componente con ‘‘ball’’

–Si aggiungono le classi per rappresentare le strutture dati di un componente (archi di composizione).

–Si aggiungono le operazioni di ogni classe (dopo aver fatto i diagrammi di sequenza).

**DIAGRAMMA DELLE CLASSI**

Definisce le classi e le loro relazioni.

–Lo stesso tipo di diagramma usato nella Specifica per rappresentare il dominio.

–Usato ora per rappresentare la strutturazione del sistema.

•Nodo classe: rappresenta una classe.

–composto da tre compartimenti

•Nome della classe

•Attributi

•**Operazioni** (Metodi)

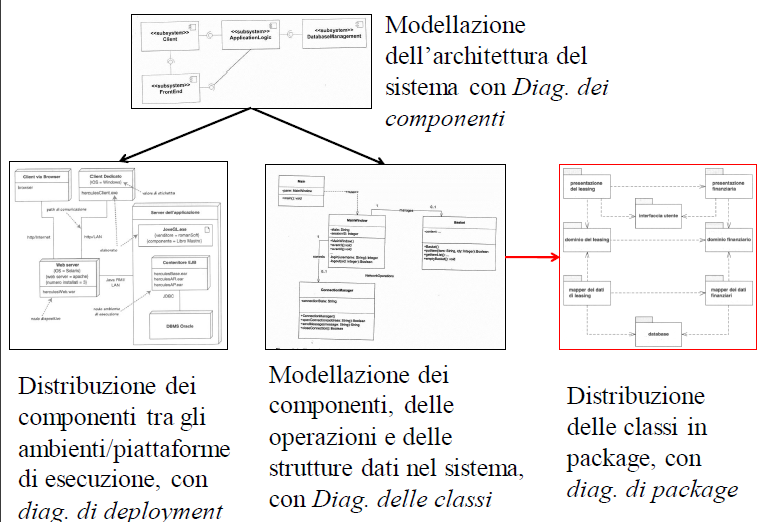
**Tipi di relazione tra classi**

•**Composizione:** un componente (classe) contiene una struttura dati (classe).

•**Dipendenza**: una classe invoca operazioni di un’altra classe.

–Notazione grafica: arco tratteggiato e orientato

A --------------> B significa che A dipende da B, cioè A invoca le operazioni di B.



**Diagramma di package**

Serve per organizzare le classi in package in vista dell’implementazione.

–Esempio: un package per ogni sottosistema.

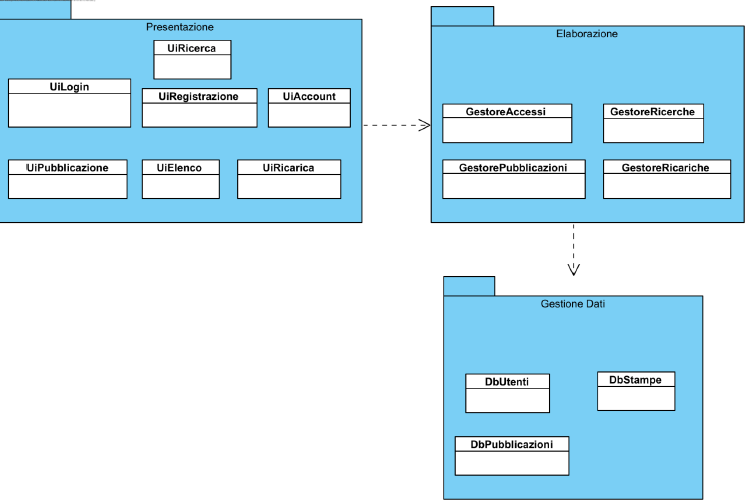
•**Package**: contenitore di classi.

–I package possono essere annidati

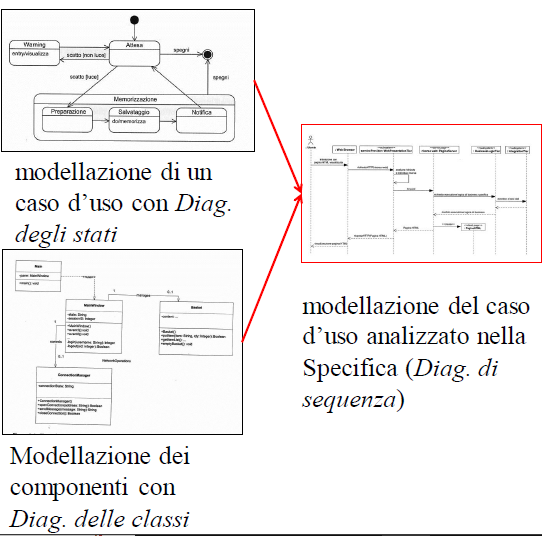
•**Archi di dipendenza**: stesso ruolo nel diagramma delle classi.

–Le dipendenze tra package derivano da dipendenze tra classi presenti in package diversi. Ad esempio: package A ----> package B

•A contiene una classe che dipende da una classe contenuta in B.



**MODELLI DI PROGETTAZIONE Comportamento**



**Diagramma di sequenza**

Mostra una sequenza temporale di interazioni tra oggetti (e attori) allo scopo di svolgere un determinato caso d’uso. L’interazione avviene attraverso lo scambio di messaggi.

I partecipanti sono:

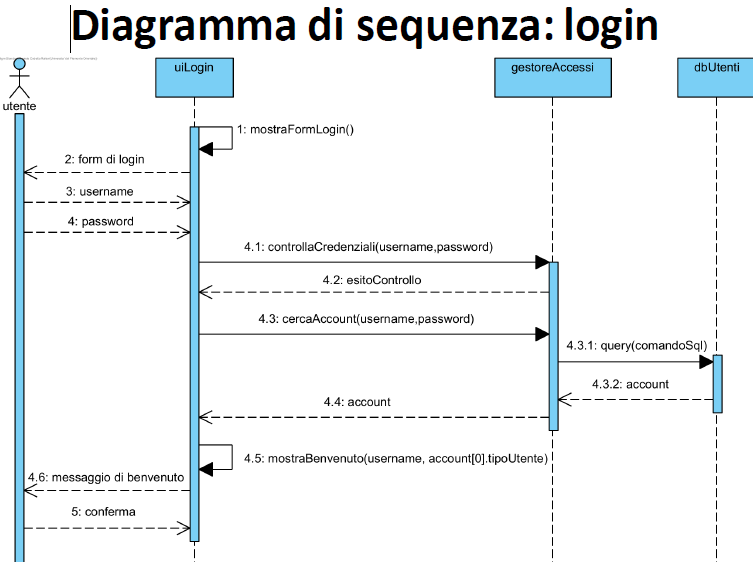
**1)Oggetti**: rappresentano componenti e sono istanze delle classi.

**2)Attori**: rappresentano tipi di utente o sistemi esterni.

Ci sono **due dimensioni**:

**1)Verticale:** rappresenta il tempo.

**2)Orizzontale:** contiene gli oggetti e gli attori.



**Diagramma di sequenza: messaggi**

**1)Invocazione di operazione**: un oggetto richiede ad un altro oggetto di eseguire una determinata operazione.

🡪Notazione grafica: arco orientato e continuo, tracciato dalla linea di vita dell’oggetto invocante verso la linea di vita dell’oggetto invocato.

Un’etichetta sull’arco indica l’operazione invocata e gli eventuali parametri.

**2)Ritorno di valore**: valore prodotto dall’esecuzione dell’operazione.

🡪Notazione grafica: arco orientato e tratteggiato, tracciato dalla linea di vita dell’oggetto invocato verso la linea di vita dell’oggetto invocante.

Un’etichetta sull’arco indica il valore ritornato.

Il valore di ritorno ha un tipo:

1)Tipo primitivo

2)Classe

Le operazioni di tipo **void** non hanno un valore di ritorno.

**3)Autoinvocazione**: un oggetto invoca un’operazione su sé stesso.

🡪Notazione grafica: arco orientato e continuo, tracciato dalla linea di vita dell’oggetto verso la stessa linea.

–Si può rappresentare l’eventuale valore di ritorno in modo analogo (arco tratteggiato).

**4)Input/Output con l’attore**: archi orientati e tratteggiati

–Input: da attore (utente) verso oggetto (componente).

–Output: da oggetto verso attore.

**5)Messaggio con ritardo**: se trascorre del tempo tra l’invio del messaggio e la sua ricezione, allora l’arco corrispondente (continuo o tratteggiato) può essere tracciato in modo obliquo (se non c’è ritardo l’arco è orizzontale).

**Vita e attivazione degli oggetti**

**1)Linea di vita**: indica il periodo di esistenza dell’oggetto.

🡪Notazione grafica: linea tratteggiata verticale che parte dall’oggetto e prosegue verso il basso.

**2)Creazione / distruzione di oggetti:** Gli oggetti già esistenti inizialmente sono indicati nella parte alta del diagramma.

🡪Un oggetto può essere creato tramite un’operazione in un momento successivo.

–Viene indicato più in basso, a livello del tempo di creazione.

🡪Un oggetto può essere distrutto tramite un’operazione in un momento successivo.

–La linea di vita si interrompe a livello del tempo di distruzione.

**3)Periodo di attivazione**: periodo di tempo in cui l’oggetto è attivo.

–Esegue un’operazione.

–Attende un valore di ritorno.

🡪Notazione grafica: rettangolo di altezza pari al tempo di attivazione, posto sopra la linea di vita.

**Diagramma di sequenza: frame**

🡪Un insieme di messaggi può essere racchiuso in un frame. I tipi principali di frame:

**1)Loop**: i messaggi all’interno del frame sono ripetuti finché vale una certa condizione (o guardia).

**2)Opt**: se la condizione è vera vengono eseguiti i messaggi nella sezione. Può avere una seconda sezione che viene eseguita se la condizione è falsa.

**3)Alt**: il frame è diviso in varie sezioni; ogni sezione è eseguita se vale la condizione dichiarata nella sezione stessa.

**4)Ref**: il frame è un riferimento ad un altro diagramma di sequenza.

🡪I frame possono essere annidati.

**Diagramma a stati 🡪** **Diagramma di sequenza**

🡪**In fase di** **specifica**, ogni caso d’uso è stato modellato da un diagramma degli stati.

🡪**In fase di** **progettazione**, ogni caso d’uso deve essere rappresentato da un diagramma di sequenza.

–Mostra le interazioni tra i componenti (oggetti) del sistema durante l’esecuzione del caso d’uso.

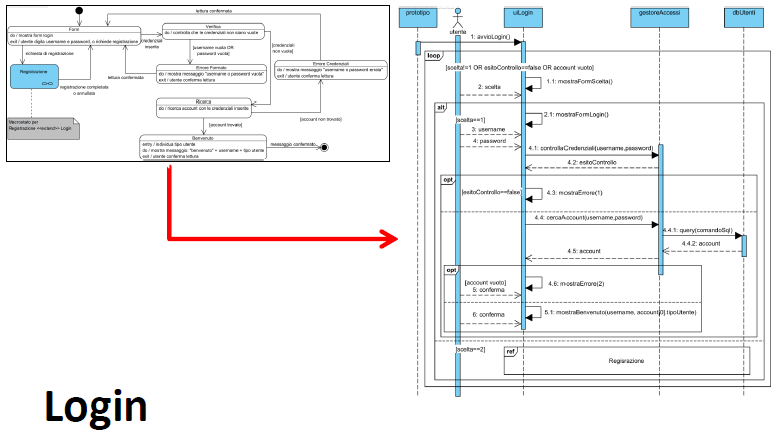
–I componenti devono essere quelli previsti nel diagramma dei componenti e delle classi.

🡪Il diagramma di sequenza deve essere coerente con il diagramma degli stati corrispondente.

–Le transizioni di stato e le attività rappresentate nel diagramma degli stati devono corrispondere alle interazioni rappresentate nel diagramma di sequenza.

–**Bisogna “tradurre” il diagramma degli stati in un diagramma di sequenza.**

**DAL DIAGRAMMA A STATI AL DIAGRAMMA DI SEQUENZA**



**Corrispondenze tra macrostati e Ref**

Se il diagramma degli stati di A contiene il macrostato B, allora il diagramma di sequenza di A deve contenere un riferimento (frame **Ref**) a B.

🡪All’interno del diagramma degli stati di A, viene invocato il diagramma degli stati di B.

🡪All’interno del diagramma di sequenza di A, viene invocato il diagramma di sequenza di B.

**Metodo di avvio**

🡪Ogni servizio (caso d’uso) avrà il proprio metodo di avvio.

🡪Il metodo di avvio può avere dei parametri in ingresso, se necessari. **Esempio**:

–Visualizza Pubblicazione *«extend»* Ricerca Pubblicazione

*public void avvio\_visualizza\_pubblicazione(Pubblicazione p)*

–Il metodo è invocato da *UI\_Ricerca* su *UI\_Pubblicazione* indicando la pubblicazione da visualizzare.

🡪Il metodo di avvio può avere un valore di ritorno, se necessario. **Esempio**:

–Stampa Pubblicazione «*include*» Pagamento

*public boolean avvio\_pagamento(int importo)*

–Il metodo è invocato da UI\_Pubblicazione su UI\_Pagamento e ritorna *true* se il pagamento è riuscito, *false* altrimenti.

•**Indicare il metodo di avvio nel diagramma di sequenza.**

**Corrispondenze tra il diagramma di sequenza (SD) e il diagramma delle classi (CD)**

🡪Ad ogni oggetto nel SD corrisponde una classe nel CD.

🡪Ogni operazione invocata su un oggetto nel SD, deve essere dichiarata nella classe corrispondente nel CD.

🡪L’invocazione di un’operazione nel SD deve rispettare la sua dichiarazione nel CD: numero di parametri, tipi dei parametri, tipo del valore di ritorno.

🡪Se l’oggetto di classe A invoca operazioni sull’oggetto di classe B nel SD, allora nel CD un arco di dipendenza è tracciato dalla classe A verso la classe B.

**Dichiarazione di operazioni nel diagramma delle classi**

🡪Dichiarazione di un’operazione:

visibilità nome\_operazione (param1: tipo[molteplicità],

param2: tipo[molteplicità], …): tipo\_valore\_ritorno[molteplicità]

🡪Tipi di visibilità di un’operazione:

–*Public* (+): visibile da tutte le classi.

–*Private* (-): visibile solo dalla classe in cui è definito.

🡪Tipi di parametro o valore di ritorno:

–Tipi primitivi: integer, float, string, Boolean, …

–Classi

**Diagramma di attività**

🡪Versione evoluta dei diagrammi di flusso:

–Barre di sincronizzazione, segnali, partizioni, passaggio di parametri, ...

🡪Applicazione: definizione dell’algoritmo relativo ad un’operazione presente nel diagramma di sequenza.

–Mostra i passi necessari per generare il valore di ritorno a partire dai parametri in ingresso.

**Esempio**: algoritmo eseguito internamente da *gestoreAccessi* quando riceve l’invocazione di +*controllaCredenziali(username: string, password: string): boolean*

**Diagramma di attività: elementi**

**1)Attività**: insieme strutturato di azioni (algoritmo)

Immagine che contiene testo

Descrizione generata automaticamente

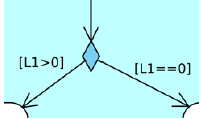
**2)Azioni**: passi elementari dell’algoritmo

–Azione iniziale

–Azione finale



**3)Decisione**: se vale una certa condizione si esegue una certa azione, altrimenti se ne esegue un’altra.



**4)Control Flow (CF)**: archi che collegano azioni, decisioni, segnali, barre di sincronizzazione, per indicare l’ordine di esecuzione.

**5)Segnali**:

–Inviato: produzione di output verso un attore.

–Ricevuto: produzione di input da parte di un attore.



**6)Nodi oggetto**: rappresentano parametri in ingresso, valori di ritorno, variabili/oggetti locali.

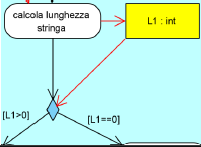


**7)Object Flow (OF)**: archi che collegano i nodi oggetto alle azioni/decisioni/segnali che ne impostano, utilizzano o ritornano il valore.

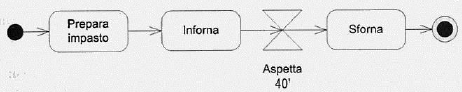
•azione 🡪 oggetto = assegnamento o return del valore

•oggetto 🡪 azione = utilizzo del valore

**8)CF** e **OF** hanno la stessa notazione grafica



**9)Attesa**: sospensione dell’attività per un certo tempo.



**10)Barre di sincronizzazione**

•**Fork**: il completamento di un’azione avvia l’esecuzione di più azioni in parallelo.

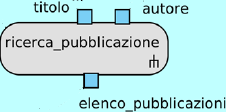
•**Join**: il completamento di più azioni avvia l’esecuzione di un’azione.

**11)Partizioni (swimlanes)**: indicano il soggetto (componente, attore, ...) che esegue l’azione.

**Diagramma di attività: attività esterna**

Prima di ritornare un valore, il componente potrebbe invocare un’ulteriore operazione su un altro componente.

–Ciò si può modellare come un’attività esterna:



–L’attività esterna viene richiamata tramite il simbolo detto ‘‘rastrello’’ che fa riferimento ad un altro diagramma di attività.

–L’uso di pin (quadrati) consente di rappresentare i parametri passati all’attività esterna, o restituiti da essa.