

**LINEE GUIDA - UML - MODELLI ARCHITETTURALI**

© Adriano Comai 2002-2006

Versione 2.0 del 27-2-2006

**INDICE**

<b>1. INTRODUZIONE .....</b>	<b>4</b>
1.1 Obiettivi .....	4
1.2 Ambito e limitazioni .....	4
1.3 Destinatari, e ruoli professionali .....	4
1.4 Riferimenti .....	4
<b>2. ARCHITETTURA.....</b>	<b>6</b>
2.1 Cosa comprende.....	6
2.2 Perché documentare l'architettura.....	6
2.3 Quando documentare l'architettura .....	7
2.4 Viste architeturali .....	7
2.5 Livelli di astrazione .....	7
<b>3. UML.....</b>	<b>9</b>
3.1 Tipologie di diagrammi UML .....	9
3.1.1. Modelli e diagrammi .....	10
3.2 Una complessità da risolvere .....	10
3.2.1. UML per pensare e per comunicare .....	11
3.2.2. UML con carta e matita, UML con strumenti.....	11
3.2.3. UML semplificato .....	11
<b>4. ELEMENTI BASE UML.....</b>	<b>13</b>
4.1 Classificatore .....	13
4.2 Componente.....	13
4.2.1. Rappresentazione di sistemi e sottosistemi .....	14
4.2.2. Articolazione interna di un componente .....	14
4.2.3. Dipendenze: relazioni tra componenti .....	15
4.2.4. Interfaccia.....	16
4.2.5. Artifact .....	17
4.3 Nodo .....	17
4.3.1. Device ed Execution environment .....	17
4.3.2. Connessioni.....	18
4.3.3. Allocazione dei componenti sui nodi.....	19
4.4 Classe.....	20
4.4.1. Oggetto (istanza di classe) .....	20
4.4.2. Associazione .....	21
4.4.3. Aggregazione e composizione .....	22
4.4.4. Generalizzazione - specializzazione .....	23
4.5 Interazione .....	23
4.5.1. Partecipante .....	24
4.5.2. Messaggio .....	24
<b>5. ORGANIZZAZIONE DEI MODELLI .....</b>	<b>27</b>
5.1 Gerarchie: versioni UML 1.x.....	27
5.2 Gerarchie: versioni UML 2.x.....	28
<b>6. I DIAGRAMMI .....</b>	<b>30</b>

6.1	Le tipologie di diagramma non sono a compartimenti stagni.....	30
6.2	Le macro-famiglie di diagrammi e l'organizzazione dei modelli .....	31
6.3	Rappresentazioni del sistema complessivo.....	32
6.3.1.	Diagramma di contesto .....	32
6.3.2.	Diagramma dei casi d'uso.....	33
6.3.3.	Diagramma di deployment.....	34
6.3.4.	Scomposizione del sistema complessivo .....	34
6.4	Rappresentazioni di dettaglio .....	35
6.4.1.	Modello di dominio e diagrammi derivati .....	35
6.4.2.	Schema dei dati .....	36
6.4.3.	Diagramma di navigazione delle interfacce utente .....	36
6.4.4.	Diagrammi di flusso.....	37
6.5	Una nota per concludere .....	37

## 1. Introduzione

### 1.1 Obiettivi

Le linee guida UML hanno un duplice obiettivo:

1. Introdurre il lettore che non conosca UML alle sue caratteristiche principali. Non presuppongono conoscenze preliminari.
2. Fornire, per chi abbia già una certa conoscenza del linguaggio, indicazioni per il suo utilizzo concreto.

### 1.2 Ambito e limitazioni

Le linee guida fanno principalmente riferimento alla documentazione ufficiale di UML (nella corrente edizione, alla versione 2.0), ed alla mia esperienza, maturata nell'utilizzo del linguaggio in ambiti eterogenei.

Non tengono in considerazione, invece, le caratteristiche e le peculiarità dei diversi strumenti di modellazione visuale UML. Di fatto, vi sono costrutti sintattici validi in UML, e riportati in questa linea guida, non supportati da alcuni strumenti; al contrario, alcuni strumenti permettono costrutti sintattici non validi in UML.

Questa linea guida sui modelli architetture si accompagna a quella sui casi d'uso (Linee Guida – UML – Casi d'uso – [9]), disponibile sul mio sito, [www.analisi-disegno.com](http://www.analisi-disegno.com) .

### 1.3 Destinatari, e ruoli professionali

Questa linea guida è indirizzata a chiunque voglia conoscere UML, sia per motivi di lavoro che di studio.

Per riferirmi all'utilizzatore di UML, uso il termine generico “sviluppatore”, nel senso di persona coinvolta, a qualsiasi titolo, in attività di sviluppo o evoluzione dei sistemi software.

Tipicamente, nelle aziende in cui si progetta software, questo ruolo generico di sviluppatore viene declinato in una pluralità di ruoli specifici (es. analista, progettista tecnico, programmatore, sistemista). Di conseguenza è frequente, all'interno delle organizzazioni, la domanda su quali modelli e diagrammi UML debbano essere creati dal ruolo X, e quali invece dal ruolo Y. Non è però mia intenzione, in questo documento, affrontare questi aspetti. Le risposte dipendono infatti solo dai confini che ai diversi ruoli sono stati assegnati in ogni particolare organizzazione.

### 1.4 Riferimenti

- [1] *UML 2.0 Superstructure Specification*, disponibile sul sito [www.omg.org](http://www.omg.org)
- [2] Jim Rumbaugh, Ivar Jacobson, Grady Booch: *The Unified Modeling Language Reference Manual* (2nd Edition) - Addison Wesley 2005
- [3] Bruce Powel Douglass : *Real Time UML* (3rd Edition) - Addison Wesley 2004
- [4] Martin Fowler : *UML Distilled* (3rd Edition) - Addison Wesley 2003

- [5] Robert Martin : *UML for Java Programmers* - Prentice Hall 2003
- [6] Craig Larman : *Applying UML and Patterns* (3rd Edition) - Prentice Hall 2005
- [7] Paul Clements ed altri: *Documenting Software Architectures. Views and Beyond* - Addison Wesley 2003
- [8] Eric Evans: *Domain-Driven Design* – Addison Wesley 2004
- [9] Adriano Comai: *Linea guida – UML – Casi d'uso* – 2004, disponibile sul sito [www.analisi-disegno.com](http://www.analisi-disegno.com)

Un sentito ringraziamento a Marco Abis, Andrea Baruzzo, John Favaro, Anna Pegna, Emilio C. Porcelli per le loro osservazioni ed i loro suggerimenti. La responsabilità di eventuali inesattezze e imprecisioni è ovviamente solo mia.

## 2. Architettura

### 2.1 Cosa comprende

La documentazione dell'architettura di un sistema software deve descrivere come è fatto il sistema. Più in particolare, può (deve) specificare:

1. Il contesto (ambiente esterno) in cui il sistema si colloca
2. L'organizzazione interna del sistema in "parti", e le modalità con cui tali "parti" interagiscono tra loro per fornire le funzionalità complessive di utilizzo (i casi d'uso) del sistema
3. Le tecnologie software utilizzate per l'implementazione e l'esecuzione delle "parti"
4. Le risorse hardware utilizzate per l'esecuzione.

L'architettura del sistema deve contribuire a soddisfare i requisiti progettuali, sia funzionali che non funzionali (livello di servizio, usabilità, sicurezza, ecc.). L'architettura, inoltre, contribuisce a determinare i costi del sistema, sia in termini di sviluppo che di gestione.

Nota: In alcuni approcci metodologici, ed in particolare nel Rational Unified Process (RUP) di IBM, la documentazione architeturale comprende una descrizione dei casi d'uso più significativi per la determinazione dell'architettura.

In questa Linea Guida, indipendente da ogni particolare processo di sviluppo, il modello dei casi d'uso non è trattato, e per le sue caratteristiche si rimanda, come già detto, alla Linea Guida specifica [9].

### 2.2 Perché documentare l'architettura

L'architettura descrive come è fatto internamente il sistema. Ma ... perché documentarla? Non è sufficiente guardare il codice applicativo, per capire com'è organizzato? In alcuni casi sì, per sistemi molto, molto semplici.

Ed esistono approcci metodologici, come Extreme Programming, che non considerano indispensabile documentazione architeturale ulteriore rispetto al codice.

Documentare l'architettura, però, ha dei vantaggi:

- permette di "entrare" più semplicemente in un'applicazione esistente, attraverso modelli sintetici (se aggiornati) piuttosto che guardando al codice; e ciò vale sia quando devo studiare un'applicazione realizzata da altri, sia quando devo rimettere le mani su una che ho realizzato io stesso in passato
- può agevolare la comunicazione tra chi partecipa ad un progetto, particolarmente quando le persone operano in luoghi diversi, o quando intervengono nel progetto in momenti successivi.

Comunque, documentare l'architettura può non essere troppo oneroso, se si segue un approccio minimalista. Se, cioè, si sceglie di limitare la documentazione alle scelte architeturali fondamentali, senza pretendere di documentare in dettaglio ogni singolo aspetto del sistema.

### 2.3 Quando documentare l'architettura

La documentazione architeturale può avvenire nel corso di un progetto, cioè mentre si sta sviluppando un nuovo sistema o si sta facendo evolvere un sistema esistente, oppure al termine del progetto.

Durante il progetto, la documentazione riveste un ruolo di indicazione, di guida per lo sviluppo, di comunicazione tra gli sviluppatori, e contribuisce a mantenere coerenza nell'architettura complessiva. La documentazione può essere più o meno dettagliata, e più o meno formale, a seconda del processo di sviluppo utilizzato e dell'organizzazione del lavoro nell'ambito del progetto.

Quando avviene al termine di un progetto, la documentazione viene effettuata per agevolare la comprensione da parte di chi dovrà intervenire per le evoluzioni future del sistema.

### 2.4 Viste architeturali

L'architettura di un sistema software è complessa, e non può essere rappresentata in modo soddisfacente adottando un unico punto di vista.

Esistono innanzitutto aspetti strutturali, relativi a come si collegano tra loro (e sono organizzate) le diverse parti del sistema.

Poi vi sono aspetti comportamentali, relativi da un lato all'interazione del sistema nei confronti del mondo esterno, e dall'altro al dinamismo dell'interazione tra le parti interne del sistema.

Inoltre c'è l'hardware, e le tecnologie software di base (sistema operativo, middleware, data base management systems) che costituiscono l'infrastruttura in cui il sistema si colloca. Le cose da rappresentare sono troppe, perché sia possibile rappresentarle tutte insieme.

La rappresentazione dell'architettura avviene quindi tramite la definizione di modelli articolati in diverse "viste architeturali":

"Una vista è una rappresentazione di un insieme di elementi del sistema, e delle relazioni che li associano" [7].

In sintesi, una vista architeturale:

- è una descrizione semplificata di un sistema
- descrive il sistema secondo uno specifico punto di vista (parziale)
- omette particolari irrilevanti per tale punto di vista

### 2.5 Livelli di astrazione

La documentazione dell'architettura di un sistema può essere effettuata a livelli di astrazione diversi. Tra i più tipici vi sono:

- livello di **analisi** - astratto, indipendente dalle scelte tecnologiche
- livello di **progettazione (sw design)** – tecnico, tiene conto delle specifiche tecnologie software utilizzate per l'implementazione delle "parti" del sistema, ma non evidenzia le risorse hardware
- livello di **deployment** - descrive le risorse hardware necessarie per l'esecuzione del sistema, e l'allocazione delle "parti" software su tali risorse

In alcuni ambiti, il livello di analisi e quello di progettazione (sw design) vengono documentati in modelli separati.

Ciò avviene, di solito, quando il modello di analisi viene creato da un ruolo organizzativo distinto rispetto a quello che progetta tecnicamente il sistema. In questa situazione, il modello di analisi costituisce normalmente uno "stadio preliminare" del modello di disegno, e non viene mantenuto aggiornato come modello separato nel corso dello sviluppo del sistema.

Analogamente, in altre situazioni progettuali, il modello di progettazione (sw design) viene creato da un ruolo organizzativo distinto rispetto a quello che implementa il sistema. Anche in questa situazione, il modello di design costituisce uno "stadio preliminare" dell'implementazione, ed è frequente il rischio che al termine dello sviluppo non risulti aggiornato rispetto all'implementazione effettiva.

Se ci si mette dal punto di vista di chi deve gestire l'evoluzione di un sistema esistente, o di chi lavora in un progetto di sviluppo che preveda una realizzazione incrementale del sistema, è comunque evidente che i modelli devono corrispondere allo stato effettivo del sistema, cioè alla sua implementazione. Un modello che non corrisponde alla realtà del sistema è, purtroppo, inutile. E perfino fuorviante.

Al contrario, un modello che renda espliciti e centrali i concetti del dominio applicativo, e le relazioni tra tali concetti, può costituire il fondamento dell'implementazione del sistema. Naturalmente, ciò presuppone un allineamento retroattivo del modello come conseguenza delle modifiche alla progettazione che avvengono inevitabilmente nel corso dell'implementazione.

In questo caso, il modello oltre a guidare efficacemente l'implementazione costituisce anche la base del linguaggio comune [8] che permette la comunicazione tra i partecipanti al progetto, ed in particolare la comprensione tra chi vuole il sistema e chi lo sviluppa.

### 3. UML

I modelli possono essere espressi in molti linguaggi. Testuali, e grafici. La storia dell'informatica ha visto succedersi una serie di forme eterogenee di rappresentazione dei sistemi software, tipicamente associate a metodi specifici di analisi e progettazione.

Dal 1997, finalmente, esiste un linguaggio unificato per la modellazione, cioè per la rappresentazione dei sistemi, **UML – Unified Modeling Language**. E' un linguaggio dotato di una semantica e di una sintassi sufficientemente ben definite [1, 2], condiviso da tutte le principali società di software internazionali, che si è rapidamente diffuso ed affermato come standard in tutto il mondo.

Questa è una linea guida pratica, e non è il caso di dilungarsi sull'importanza ed il significato di UML. E' però opportuno sottolinearne due caratteristiche fondamentali:

- può rappresentare qualunque tipo di sistema software, indipendentemente dalla tecnologia di implementazione
- è un linguaggio indipendente dai metodi e dai processi di sviluppo utilizzati per creare i sistemi

#### 3.1 Tipologie di diagrammi UML

UML è un linguaggio completo, con definizioni rigorose degli elementi che lo costituiscono e delle modalità con cui i diversi elementi possono essere combinati tra loro. A rigore, è possibile modellare un sistema in UML senza utilizzare notazioni grafiche, utilizzando unicamente, e in modo rigoroso, forme di espressione testuale.

Ciò nonostante, UML è soprattutto usato come notazione diagrammatica, per motivi di immediatezza.

Nella versione attuale dello standard UML, la **2.0**, il linguaggio prevede 13 tipologie di diagramma (i termini inglesi riportati tra parentesi sono quelli standard):

##### Diagrammi “strutturali”

- diagramma delle classi (class)
- diagramma degli oggetti (object)
- diagramma dei componenti (component)
- diagramma delle strutture composite (composite structure)
- diagramma di deployment (deployment)
- diagramma dei package (package)

##### Diagrammi “comportamentali”

- diagramma dei casi d'uso (use case)
- diagramma di stato (statechart)
- diagramma delle attività (activity)
- diagramma di sequenza (sequence)
- diagramma di comunicazione (communication)
- diagramma dei tempi (timing)
- diagramma di sintesi dell'interazione (interaction overview)

} diagrammi di interazione

Il numero dei diagrammi è elevato, ma non deve spaventare. Tra i 13 diagrammi esistono numerose affinità, e si possono identificare 5 “macro-famiglie” principali:

1. Diagrammi strutturali – i sei diagrammi strutturali rappresentano le parti del sistema, e le loro interrelazioni. Condividono molti elementi, e non sono rigidamente distinti tra loro.
2. Diagrammi di interazione – i quattro diagrammi di interazione rappresentano la dinamica delle collaborazioni che le parti del sistema attuano per implementare i servizi offerti dal sistema. Rappresentano in forme diverse i medesimi elementi. Si differenziano essenzialmente nelle modalità di rappresentazione della dimensione temporale.
3. Diagramma di stato – un diagramma che rappresenta il ciclo di vita di un oggetto, e che ne evidenzia le condizioni di esistenza (stati) e gli eventi che modificano tali condizioni, provocando la transizione da uno stato ad un altro.
4. Diagramma di attività – un diagramma di flusso (flow chart), utile per rappresentare la logica interna di un processo o di un'operazione.
5. Diagramma dei casi d'uso – un diagramma semplice, che rappresenta una mappa degli utilizzi del sistema, e dei soggetti esterni al sistema (attori) coinvolti in tali utilizzi.

Per mantenere questa Linea Guida entro dimensioni contenute evito di fornire spiegazioni ed esempi dei singoli diagrammi. Esempi e spiegazioni si possono trovare in [1, 2, 3, 4, 6], sul mio sito nel tutorial <http://www.analisi-disegno.com/uml/introuml.pdf>, e in molte altre fonti.

### 3.1.1. Modelli e diagrammi

A volte, i termini modello e diagramma vengono usati come se avessero il medesimo significato. Ma non è così.

Un modello relativo ad un sistema software contiene elementi di vario tipo (es. componenti; classi, casi d'uso). Può comprendere (non necessariamente) una serie di diagrammi, che rappresentano in modo visuale gli elementi e le relazioni tra gli elementi. Ma comprende anche altre forme di documentazione, tipicamente sotto forma di testo.

Occorre sottolineare che la scelta di quanti e quali diagrammi inserire in un modello non è vincolata in UML, dato che il linguaggio non fornisce alcuna indicazione in merito. Lo sviluppatore, a seconda della sua esperienza (e degli eventuali standard presenti nella sua organizzazione) può scegliere se e quali diagrammi utilizzare nel suo progetto.

## 3.2 Una complessità da risolvere

UML è complesso. La versione 2.0 del linguaggio, ufficializzata nel 2005, è documentata in una serie di testi (il più rilevante per gli utilizzatori, [1]) che comprendono migliaia di pagine.

La complessità deriva da più fattori:

- il linguaggio intende rappresentare qualunque tipo di sistema software, a diversi livelli di astrazione.
- il numero degli elementi è elevato, e in molti casi è possibile scegliere tra forme di rappresentazione diverse per il medesimo elemento.

- l'Object Management Group (OMG), che è l'organismo proprietario dello standard UML, ha spinto per una definizione sempre più precisa di tutti gli elementi del linguaggio, per permettere un'automazione completa della generazione di codice a partire dai modelli.

Fortunatamente, nel concreto dei progetti, non è necessario utilizzare “tutto UML”. Anzi. In alcune situazioni, un livello troppo spinto di formalizzazione può risultare controproducente, e rendere la comunicazione tra i partecipanti al progetto più difficile del necessario.

### 3.2.1. UML per pensare e per comunicare

UML può essere usato per **pensare**. Disegno un diagramma per chiarirmi le idee. Se lo uso così, per ragionare, posso permettermi di fare solo i diagrammi che mi servono al momento, al livello di dettaglio e di precisione che mi interessa, senza pormi problemi di comunicazione.

UML può essere usato per **comunicare** (e documentare). Tra soggetti diversi, distanti nello spazio (ad esempio da analisti separati fisicamente dagli sviluppatori) o nel tempo (posso trovarmi a riprendere in mano modelli che io stesso ho creato in passato). Tra aziende diverse, che devono collaborare su basi contrattuali.

L'utilizzo di UML per comunicare richiede naturalmente maggiore attenzione, maggior cura. Prima di tutto è necessario che chiarisca con chi voglio comunicare, che cosa esattamente voglio comunicare, e a quale livello di dettaglio e di precisione.

### 3.2.2. UML con carta e matita, UML con strumenti

I diagrammi UML possono essere prodotti con diverse tecnologie. Sono disponibili da anni numerosi strumenti di modellazione visuale, progettati per agevolare la diagrammazione UML. Questi strumenti si differenziano sia per caratteristiche (livello minimo: semplici disegnatori; livello massimo: ambiente di sviluppo con funzionalità di generazione di codice e reverse engineering) che per costo (strumenti gratuiti, strumenti a basso prezzo, strumenti molto costosi).

Comunque, in molte situazioni i diagrammi UML vengono disegnati con tecnologie più basilari. Ad esempio su lavagne a fogli mobili o cancellabili, durante una riunione di lavoro.

### 3.2.3. UML semplificato

E' possibile, e vantaggioso, un utilizzo semplificato di UML. E questa linea guida, che ha l'obiettivo di indicare come ci si può servire di UML nelle situazioni concrete, seguirà per l'appunto un approccio semplificato.

Ecco alcune indicazioni preliminari dettate dall'esperienza. Banalità, forse, puro buon senso. Ma secondo me utili:

- Il numero dei diagrammi deve essere contenuto. In termini generali, meglio pochi diagrammi che troppi. Sia per il “lettore”, che deve capire, che per il “produttore di diagrammi”, che li deve mantenere aggiornati (diagrammi non aggiornati sono controproducenti, perché ostacolano la

comprensione). Ogni nuovo diagramma deve aggiungere valore al modello, ossia fornire davvero un contributo ulteriore per la comprensione del sistema.

- Ogni elemento inserito in un diagramma deve fornire valore aggiunto per la comprensione del sistema. UML permette di rappresentare molti aspetti in estremo dettaglio. Così facendo, però, si rischia a volte un eccesso di informazione, e quindi di non riuscire ad evidenziare gli aspetti fondamentali che si vorrebbero comunicare.
- La precisione e l'aderenza a UML non devono andare a scapito della comprensibilità. Una immagine vale mille parole. Ma se servono mille parole a spiegare un diagramma a chi non conosce UML in modo approfondito, non funziona.
- Il modello deve riflettere la realtà del sistema. Oppure costituire un tentativo, una prova per la sua evoluzione futura. Altrimenti non serve.
- Meglio rappresentare un aspetto parziale, in modo comprensibile, che rappresentare in modo completo tutti gli elementi rendendo più difficile la comprensione.
- La documentazione non è fatta solo di diagrammi. Note e descrizioni testuali possono essere, a volte, maggiormente espressivi.
- La fedeltà a UML aiuta la comunicazione (è uno standard...). Ma è un mezzo, non il fine, non l'obiettivo. Se la fedeltà a UML rende più difficile la comunicazione, meglio essere meno fedeli. Non esiste una "polizia UML".
- UML serve a comunicare, a far capire le cose a chi guarda i diagrammi. Non a mostrare la competenza o la bravura di chi li fa.

In questa Linea Guida, i suggerimenti di utilizzo semplificato di UML verranno preceduti da questo simbolo **!!!** come nell'esempio seguente:

**!!!** Nel normale utilizzo pratico, non esistono differenze sostanziali tra artifact e componente. I componenti sono sufficienti, anche perché è ammesso in UML il deployment di componenti sui nodi (per retrocompatibilità con le prime versioni di UML).

## 4. Elementi base UML

### 4.1 Classificatore

Il classificatore (“classifier”) è l’elemento centrale nel meta-modello di UML.

Esistono diverse tipologie specifiche di classificatore, che condividono alcune caratteristiche di fondo, tra cui la possibilità di:

- avere attributi ed operazioni
- avere istanze proprie
- avere associazioni con altri classificatori
- essere specializzato da altri classificatori

Tra le molte tipologie di classificatore definite in UML, le più importanti sono:

- Componente (“component”)
- Classe (“class”)
- Interfaccia (“interface”)
- Attore (“actor”)
- Caso d’uso (“use case”)
- Nodo (“node”)

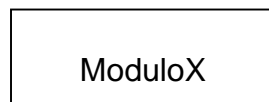
L’icona di base per la rappresentazione dei classificatori è un rettangolo. Pertanto, tutti i classificatori possono essere rappresentati sotto forma di rettangolo, anche se UML prevede anche icone particolari per ogni tipologia.

### 4.2 Componente

Rappresenta una parte del sistema, modulare e sostituibile.

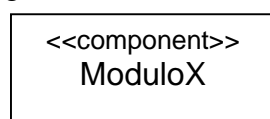
Ogni parte di un sistema software può essere rappresentata come componente UML: moduli realizzati in qualunque tecnologia (object oriented o meno); pagine web; archivi e tabelle, ecc..

La rappresentazione grafica dei componenti è quella di base per tutti i classificatori, ossia il rettangolo.

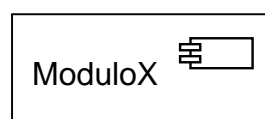


rappresentazione base di un componente

Per specificare che si tratta di un componente, e non di un classificatore generico, può essere usato uno stereotipo, visuale o testuale (gli stereotipi sono indicazioni usate in UML per specificare una tipologia particolare di un elemento generico).



rappresentazione con stereotipo testuale



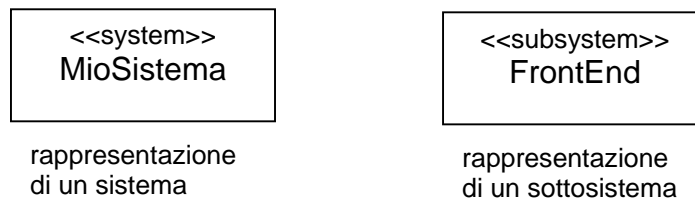
rappresentazione con stereotipo visuale

!!! Utilizzare gli stereotipi solo quando servono a chiarire. Se ai lettori di un diagramma risulta comunque chiaro che il rettangolo rappresenta un componente, l'esplicitazione dello stereotipo <<component>> o dell'icona corrispondente è superflua.

!!! Gli stereotipi testuali possono utilizzare vocaboli italiani, quando in questo modo si favorisce la comunicazione (dipende, com'è ovvio, dai destinatari del diagramma).

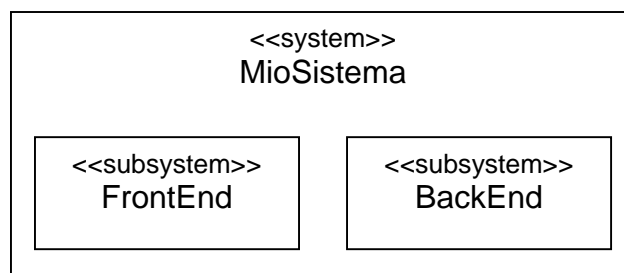
#### 4.2.1. Rappresentazione di sistemi e sottosistemi

In UML 2.0 anche il sistema complessivo, e le macro-parti in cui è articolato (i sottosistemi) vengono rappresentati come componenti.



#### 4.2.2. Articolazione interna di un componente

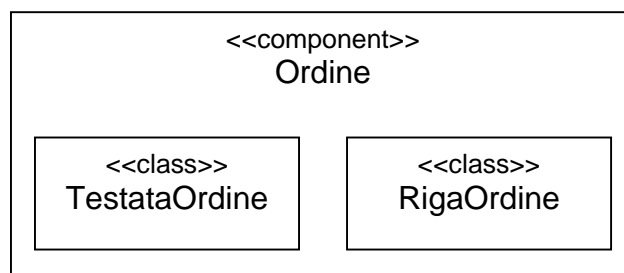
L'articolazione interna di un componente (cioè la sua scomposizione) può essere rappresentata includendo le parti all'interno del componente di livello superiore:



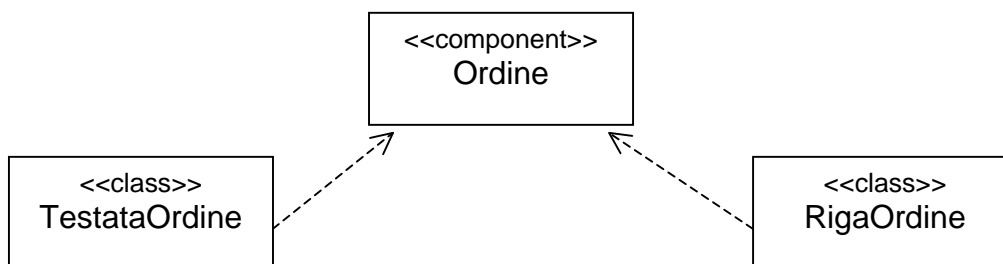
rappresentazione dell'articolazione interna di un componente  
(in questo caso, un sistema)

Ogni componente può essere articolato al suo interno in componenti di livello inferiore.

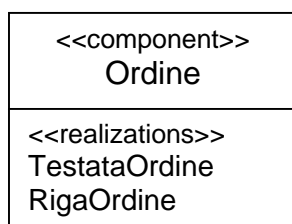
Per i componenti realizzati con linguaggi object oriented, le caratteristiche ed il comportamento di un componente sono specificate dalle classi che "risiedono" nel componente, cioè che lo implementano. La rappresentazione grafica avviene anche in questo caso come inclusione:



In alternativa, il legame tra il componente object oriented e le classi che lo implementano può essere rappresentato con una dipendenza (dependency):



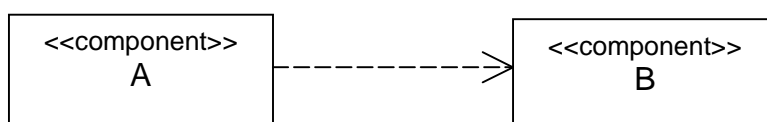
Oppure anche elencando le classi all'interno del componente:



### 4.2.3. Dipendenze: relazioni tra componenti

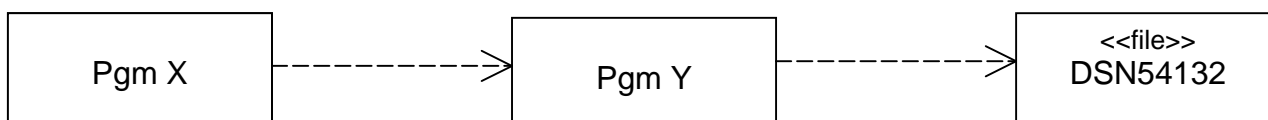
Le relazioni tra componenti sono rappresentate in UML tramite dipendenze.

La dipendenza (“dependency”) indica che il funzionamento di un elemento (ad esempio il componente A) richiede la presenza di uno o più altri elementi (ad esempio il componente B). La dipendenza è espressa con una “freccia” tratteggiata, che parte dall’elemento “client” (dipendente) e arriva all’elemento “supplier” (quello necessario per il funzionamento del client).



La dipendenza implica anche che, se un elemento viene modificato, potrebbe essere necessario modificare anche ogni elemento che da esso dipende (nell’esempio, se viene modificato il componente B è possibile che debba essere modificato anche il componente A).

In alcuni ambiti tecnologici, i componenti possono essere collegati tra loro con dipendenze dirette.



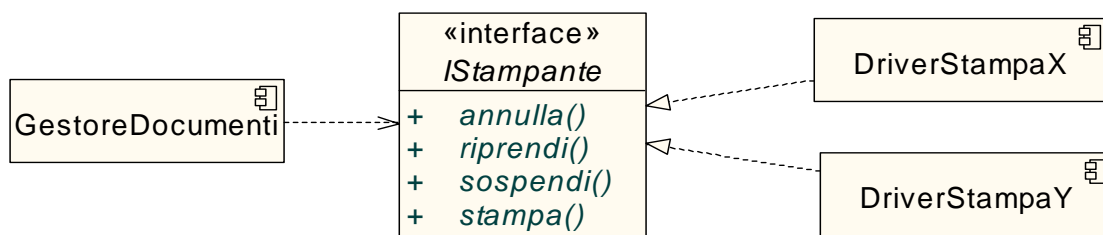
In altri ambiti tecnologici, per ridurre il “coupling” (accoppiamento) tra i componenti, cioè le relazioni dirette che possono rendere onerose le evoluzioni future del sistema, si richiede che ogni relazione tra componenti venga intermediata da interfacce.

#### 4.2.4. Interfaccia

E' un classificatore che dichiara un insieme di operazioni, ma non le implementa.

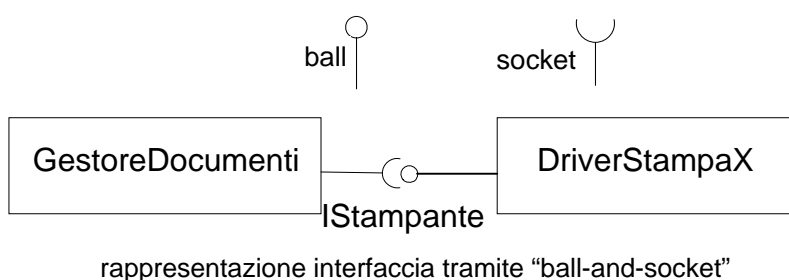
Costituisce un meccanismo fondamentale per il disaccoppiamento (“decoupling”) tra le diverse parti del sistema, consentendo a chi richiede una operazione di effettuare la richiesta in modo generico, senza preoccuparsi delle particolarità implementative di chi fornirà il relativo servizio.

Ad esempio, un componente “elaboratore di testi” GestoreDocumenti può richiedere servizi di stampa a stampanti diverse, ma formulando le richieste (stampa, annulla stampa, sospendi, riprendi) sempre con le stesse modalità, dichiarate in una interfaccia comune IStampante. I driver di stampa specifici (nell'esempio, DriverStampaX, DriverStampaY, che guidano il funzionamento di diverse stampanti) implementano ciascuno a modo proprio le operazioni dichiarate nell'interfaccia IStampante.



La relazione tra l'interfaccia e l'elemento che richiama le operazioni definite nell'interfaccia (nell'esempio, GestoreDocumenti) è rappresentata con una dipendenza. La relazione tra l'interfaccia e l'elemento che implementa le operazioni (nel nostro caso, i Driver) è invece rappresentata come “realizzazione” (linea tratteggiata, punta della freccia chiusa e bianca).

Esiste però anche una rappresentazione grafica alternativa, e più sintetica. L'interfaccia viene rappresentata come un pallino, ed è connessa all'elemento che implementa da una linea continua. L'elemento che richiede i servizi può essere agganciato al pallino tramite un “socket” .

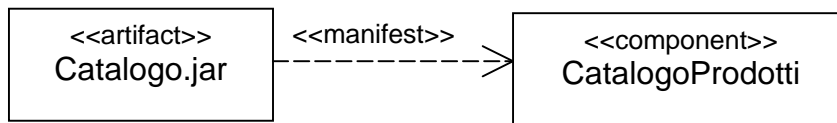


Possono implementare interfacce i seguenti classificatori:

- attori
- classi
- componenti

### 4.2.5. Artifact

L'“artifact” è un prodotto fisico del processo di sviluppo software, che può costituire l'implementazione di un componente. Serve a rappresentare gli elaborati “fisici” che verranno installati sui nodi hardware.



relazione esplicita tra componente e artifact

**!!!** Nel normale utilizzo pratico, non esistono differenze sostanziali tra artifact e componente. I componenti sono sufficienti, anche perché è ammesso in UML il deployment di componenti sui nodi (per retrocompatibilità con le prime versioni di UML).

## 4.3 Nodo

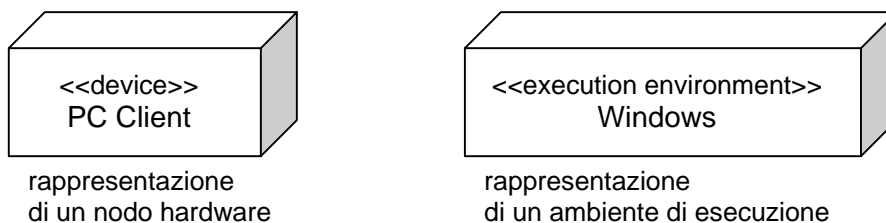
Il nodo è un elemento presente nell'ambiente di esecuzione del sistema, nel quale possono essere ubicati dei componenti (o, per essere più precisi, degli artifact che costituiscono la “manifestazione” di componenti).

### 4.3.1. Device ed Execution environment

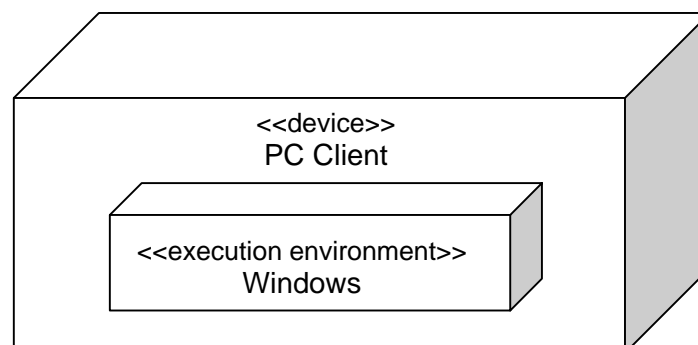
UML 2.0 prevede due tipologie di nodo:

- Device – rappresenta una risorsa hardware
- Execution environment – rappresenta un “contenitore” software, che permette l'esecuzione del software applicativo (es. sistema operativo, data base management system)

L'icona per rappresentare un nodo è il parallelepipedo.

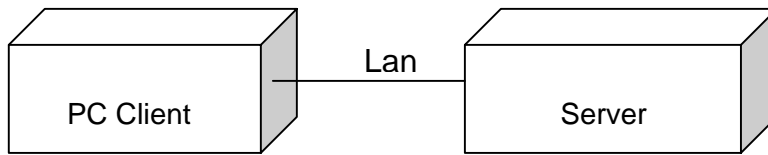


Un ambiente di esecuzione può essere rappresentato all'interno di un nodo hardware:

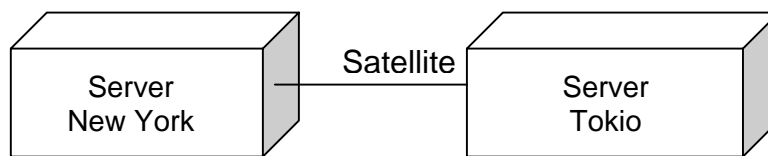


### 4.3.2. Connessioni

I nodi sono collegati da connessioni, che indicano un percorso di comunicazione.



Dato che le connessioni sono tipicamente assicurate da apparati fisici, è possibile rappresentare anche questi elementi come nodi, nel caso se ne intendano specificare le caratteristiche.

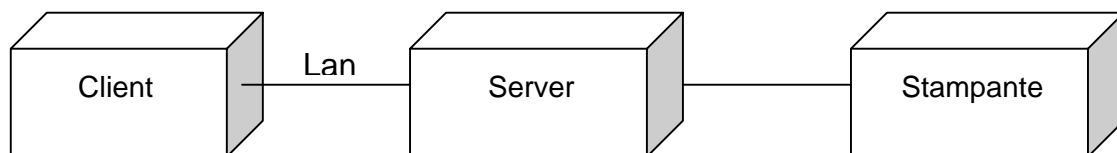


rappresentazione di un satellite come connessione



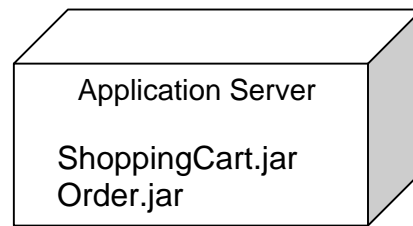
rappresentazione di un satellite come nodo

**!!!** Nei diagrammi di deployment, se non si vuole rappresentare il software, può essere meglio usare icone stereotipo che rappresentino in modo più esplicito la tipologia dei nodi hardware, anziché i parallelepipedi.



rappresentazione con icone generiche





rappresentazione dei componenti (artifact) allocati su un nodo come elenco testuale

#### 4.4 Classe

La classe costituisce l'elemento base della progettazione per i sistemi basati su tecnologie object oriented.

E' un classificatore che rappresenta una parte del sistema autonoma, dotata di caratteristiche (attributi) e comportamenti (operazioni).

Più propriamente, la classe è un descrittore, che definisce gli attributi, le operazioni e le associazioni che caratterizzano e si applicano ad un insieme di oggetti, che costituiscono le istanze della classe. Ogni oggetto della classe (o di sottoclassi della classe) può avere valori specifici per ogni attributo definito nella classe (variabili di istanza); ad esso si applicano tutte le operazioni dichiarate nella classe. Fanno eccezione attributi ed operazioni il cui "scope" (ambito) è la classe stessa: i primi costituiscono variabili di istanza, e non vengono quindi valorizzati sui singoli oggetti; le seconde non vengono indirizzate ad oggetti specifici, bensì alla classe nella sua globalità.

La rappresentazione sintetica di base della classe, come per gli altri classificatori, è un rettangolo. La rappresentazione estesa prevede tre comparti: nome, elenco degli attributi, elenco delle operazioni.

La classe può essere associata ad altre classi, e implementare (realizzare) un insieme di interfacce.

##### 4.4.1. Oggetto (istanza di classe)

Una classe è un tipo particolare di insieme, che non ammette elementi duplicati. Un oggetto della classe è un elemento dell'insieme.

La classe definisce i dati (attributi) e le funzioni (operazioni) previsti per tutti gli oggetti che ne fanno parte.

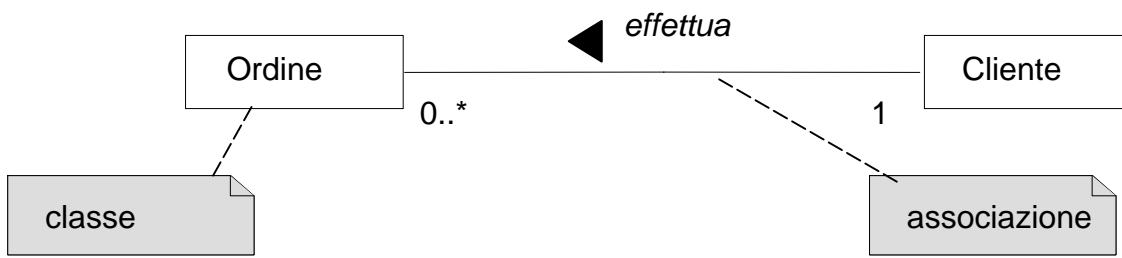
Ad esempio, tutti gli oggetti della classe OrdineAcquisto:

- ricevono un valore per gli attributi numOrdineAcquisto e dataOrdine
- vengono creati con l'operazione creaOrdineAcquisto()
- aggiornati con aggiornaDataOrdine()
- cancellati con cancellaOrdine()

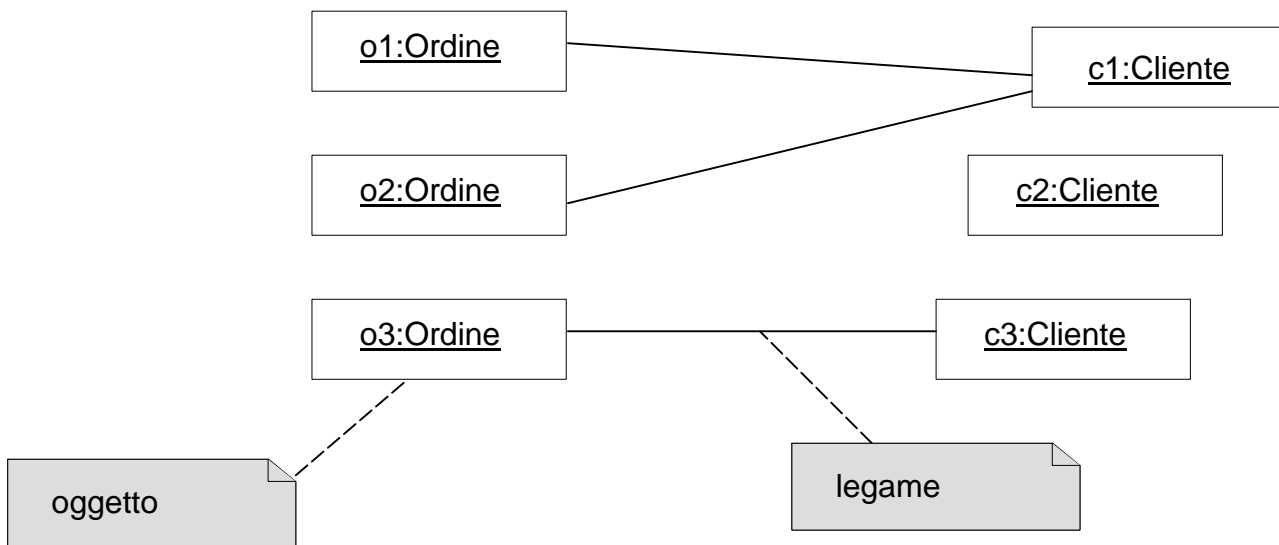
<b>OrdineAcquisto</b>
numOrdineAcquisto dataOrdine
creaOrdineAcquisto() aggiornaDataOrdine() cancellaOrdine()

La rappresentazione di oggetti in un diagramma strutturale (l'Object Diagram, appunto) viene effettuata quando si vuole esemplificare in concreto una relazione tra classi.

Ad esempio, questa associazione tra le classi Ordine e Cliente:



può essere esemplificata dal successivo diagramma degli oggetti:



#### 4.4.2. Associazione

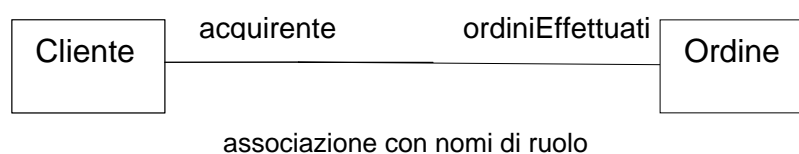
L’associazione (“association”) definisce una relazione semantica tra classificatori, e rappresenta un insieme di legami (“link”) tra le istanze di tali classificatori.

Ogni associazione ha almeno due estremi (“association ends”), per ognuno dei quali possono essere definiti:

- un nome (nome di ruolo)
- una molteplicità
- un indicatore di navigabilità

##### Nome di ruolo

Il nome degli estremi delle associazioni che partono da un classificatore deve essere univoco, anche rispetto agli eventuali attributi del classificatore.



### Molteplicità

La molteplicità indica il numero di istanze del classificatore associato ad un estremo che possono essere collegate a ciascuna delle istanze del classificatore collocato all'altro estremo dell'associazione, ed è espressa sotto forma di range di interi non negativi.



associazione con molteplicità espressa

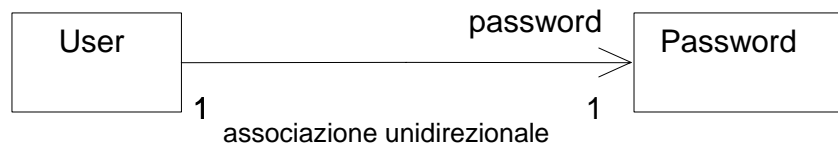
La molteplicità è espressa da due valori, minimo e massimo, separati da due punti. L'asterisco indica un intero non negativo (zero compreso). Quando le molteplicità minima e massima coincidono, si rappresenta un valore unico.

### Navigabilità

L'indicatore di navigabilità indica se è possibile raggiungere le istanze collocate ad un estremo a partire da una istanza collocata all'altro estremo dell'associazione, cioè se è possibile percorrere il legame e arrivare, a partire da un oggetto in una classe, agli oggetti ad esso collegati nell'altra classe

Le associazioni navigabili in entrambe le direzioni sono dette, appunto, bidirezionali. La loro rappresentazione è una linea priva di verso.

Nelle associazioni unidirezionali, invece, l'associazione assume una forma di freccia che indica la direzione percorribile. Nell'esempio riportato sotto, è possibile passare da un oggetto User al corrispondente oggetto Password, ma non viceversa.



associazione unidirezionale

### 4.4.3. Aggregazione e composizione

Le associazioni possono essere di tre tipi:

- normali
- aggregazioni
- composizioni

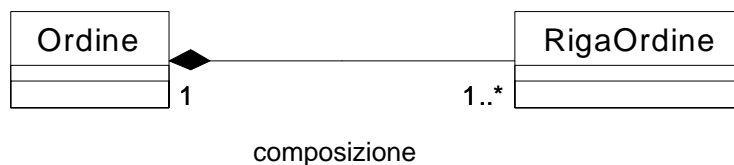
Le **aggregazioni** semplici (“aggregation”) esprimono una relazione tra un insieme e delle parti di tale insieme (“tutto / parte”), ma non implicano vincoli particolari per l'esistenza della parte o dell'insieme. Sono espresse con una losanga bianca, collocata all'estremo dell'associazione vicino al classificatore che rappresenta l'insieme.



aggregazione

Le aggregazioni di composizione (o più semplicemente, **composizioni** – “composition”) implicano vincoli precisi: ogni istanza del classificatore “parte” può essere riferita ad una sola istanza del “tutto”; inoltre, il “tutto” è responsabile per la creazione e la distruzione delle sue “parti”, che non possono avere una esistenza autonoma.

Sono espresse con una losanga nera, collocata all’estremo dell’associazione vicino al classificatore che rappresenta l’insieme.



#### 4.4.4. Generalizzazione - specializzazione

La generalizzazione - specializzazione (“generalization - specialization”) è una relazione che collega un elemento più generico ad un elemento più specifico.

L’elemento specializzato (sottoclasse) deve essere pienamente consistente con quello generalizzato (superclasse), cioè ne eredita tutte le caratteristiche (una sottoclasse eredita tutti gli attributi, le operazioni e le associazioni delle sue superclassi), ma può definire ulteriori attributi, operazioni e relazioni.

La generalizzazione - specializzazione è una relazione che definisce dei sottotipi: un’istanza dell’elemento generico può essere sempre sostituita da un’istanza dell’elemento più specifico.

La generalizzazione – specializzazione è espressa con una linea continua, conclusa con una punta di freccia chiusa all’estremo corrispondente all’elemento generico.



### 4.5 Interazione

L’interazione è una collaborazione tra più elementi, finalizzata all’implementazione di una funzionalità. Specifica il modo in cui un caso d’uso viene realizzato, grazie alla collaborazione tra un insieme di classificatori o di istanze. Per ogni caso d’uso possono essere creati uno o più diagrammi di interazione (idealmente, uno per ogni scenario).

L’interazione può essere rappresentata in quattro diagrammi UML 2.0, simili per contenuto anche se diversi nella forma:

- il diagramma di comunicazione (che nelle versioni 1.x di UML si chiamava diagramma di collaborazione), che evidenzia le relazioni tra i partecipanti all’interazione
- il diagramma di sequenza, che evidenzia invece gli aspetti temporali, cioè l’ordine secondo il quale i partecipanti si scambiano messaggi
- il diagramma dei tempi (timing), una variante del diagramma di sequenza che permette una maggior precisione nella specifica delle tempistiche dei messaggi, grazie alla definizione di unità di misura temporali e all’utilizzo di scale basate su tali unità di misura

- il diagramma di sintesi dell'interazione, che serve per sintetizzare interazioni molto complesse mediante una notazione analoga ai diagrammi di attività

**!!!** Non è indispensabile creare diagrammi di interazione per ogni caso d'uso, in particolare se si tratta di casi d'uso molto semplici o molto simili tra loro. E in generale vanno descritte le interazioni solo per gli scenari più importanti, ossia quelli che forniscono valore aggiunto nella comprensione del sistema.

Gli elementi principali di un'interazione sono i partecipanti e i messaggi che i partecipanti si scambiano.

#### 4.5.1. Partecipante

Il termine ufficiale UML 2 per indicare i partecipanti in una interazione è Lifeline (letteralmente, linea della vita).

Le lifeline possono corrispondere a qualunque classificatore (es. componenti, classi, attori, nodi), sia a livello di tipi che di istanze. E' quindi possibile creare diagrammi di interazione che specifichino gli aspetti dinamici di collaborazioni tra ruoli organizzativi (rappresentati come attori o come "workers", secondo il profilo UML per il Business Modeling); tra componenti software; tra nodi hardware. E naturalmente tra oggetti.

#### 4.5.2. Messaggio

I partecipanti ad una interazione collaborano scambiandosi messaggi, cioè richieste di servizio.

UML prevede due tipologie di messaggio: sincrono e asincrono.

- messaggio sincrono (call di un'operazione): il mittente si ferma ad aspettare la risposta dal destinatario
- messaggio asincrono (invio di un segnale): il mittente non si ferma, e prosegue la sua attività senza aspettare risposta



Ogni messaggio può avere argomenti associati (parametri), e specificare valori di ritorno.

L'invio di un messaggio nell'ambito di una interazione presuppone l'esistenza di una qualche forma di legame tra il mittente ed il destinatario, ossia di un canale di comunicazione tra i partecipanti coinvolti.

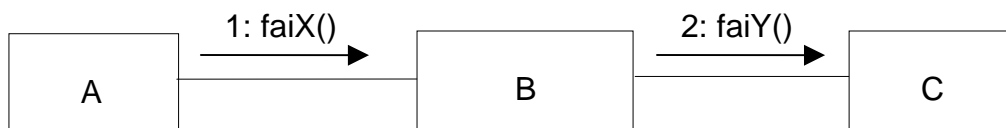


diagramma di comunicazione: il messaggio "viaggia" attraverso un legame

Il legame tra i partecipanti, necessario per permettere lo scambio di messaggi, ha una natura strutturale. Se esiste uno scambio di messaggi tra due partecipanti in un diagramma di interazione, deve esistere un corrispondente legame strutturale (dipendenza o associazione) tra i classificatori coinvolti.

Quindi, seguendo l'esempio sopra riportato, a livello strutturale deve esistere una dipendenza o un'associazione tra i partecipanti A e B, ed un'altra dipendenza o associazione tra B e C.

Il legame viene reso esplicito nel diagramma di comunicazione, mentre rimane implicito in un diagramma di sequenza.

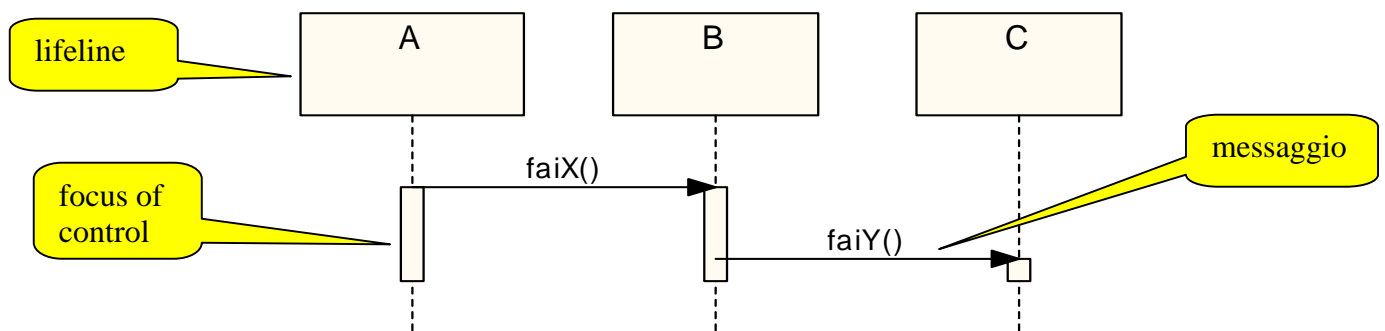


diagramma di sequenza: i legami non vengono rappresentati in modo esplicito

In ogni caso, il messaggio è una richiesta di servizio che un mittente indirizza ad un destinatario. Nell'esempio, il messaggio `faiX()` corrisponde ad un'operazione definita nella classe del partecipante B o, in termini più generali, ad uno dei servizi offerti da B.

Ricevuto il messaggio, B si attiva. L'attivazione è rappresentata da un rettangolo sottile, il "focus of control", che indica l'esecuzione dell'operazione `faiX()`.

Nell'ambito dell'esecuzione di `faiX()`, B richiede a C il servizio `faiY()`, corrispondente ad un'operazione definita nella classe del partecipante C, o, in termini più generali, ad uno dei servizi offerti da C.

C si attiva, ed al termine dell'esecuzione di `faiY()` restituisce il controllo a B. Notare che l'attivazione di C è evidenziata dall'inizio di un focus of control, mentre il termine dell'attivazione di C, con la restituzione del controllo a B, è rappresentato dalla conclusione del focus of control. Notare anche che durante l'attivazione di C, il focus of control di B rimane attivo, in quanto evidenzia l'attesa della restituzione di controllo.

Dopo aver riottenuto il controllo da C, B terminerà l'esecuzione di `faiX()` e restituirà a sua volta il controllo ad A.

**!!!** Il focus of control è opzionale. Può essere utile se si intende rappresentare quali oggetti sono attivi contemporaneamente.

La rappresentazione della restituzione di controllo al termine dell'esecuzione di un'operazione è, per messaggi sincroni, opzionale. Volendo, può essere rappresentata con una freccia tratteggiata, in direzione opposta al messaggio sincrono che ha provocato l'attivazione.

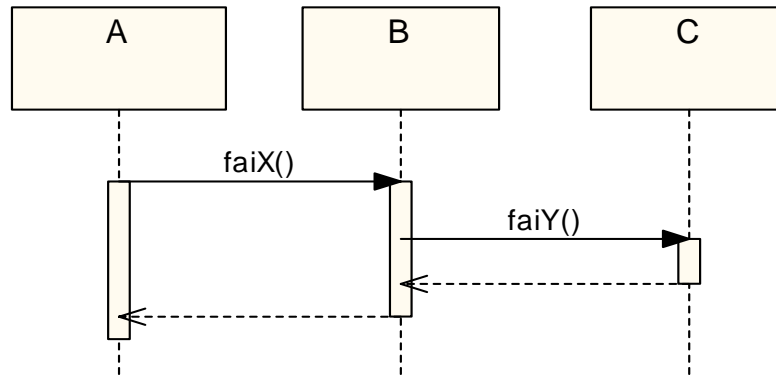


diagramma di sequenza con risposte evidenziate anche per messaggi sincroni

**!!!** In diagrammi di sequenza con pochi partecipanti e pochi messaggi, rappresentare in modo esplicito la restituzione di controllo, cioè le risposte a messaggi sincroni, aiuta a chiarire la dinamica dell'interazione. Quando invece il numero dei messaggi è elevato, evidenziare le risposte complica il diagramma (il numero di frecce si duplica).

Nel caso di messaggi asincroni, invece, la rappresentazione delle eventuali risposte è obbligatoria (non la si indica però con una freccia tratteggiata, utilizzabile solo per messaggi sincroni, ma con un altro messaggio).

## 5. Organizzazione dei modelli

UML può essere utilizzato per creare diagrammi non correlati tra loro.

Ad esempio così: ogni volta in cui serve creare un diagramma, lo si fa. Ogni diagramma viene creato in modo totalmente indipendente dagli altri. I diagrammi prodotti possono essere archiviati oppure no, ma non esiste l'esigenza di integrarli. Per chi intende utilizzare UML in questo modo (che è assolutamente lecito, e in molte situazioni funziona), molte delle considerazioni che riporto in questo capitolo sono superflue.

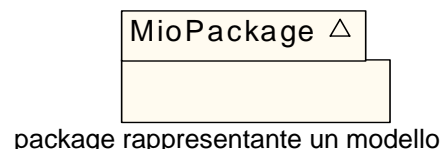
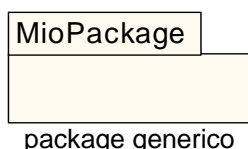
Nella maggioranza delle situazioni, comunque, è necessario che i modelli creati con UML vengano organizzati, strutturati per ridurre la complessità del lavoro dei progettisti. E' opportuno farlo, in particolare, quando si utilizzano strumenti di modellazione visuale basati su UML (ce ne sono decine, un elenco parziale si può trovare all'indirizzo <http://www.analisi-disegno.com/uml/StrumentiUML.htm> ).

Il meccanismo più efficace per organizzare i modelli è la gerarchia. L'organizzazione ottimale del modello è quella che riflette la strutturazione interna del sistema (se invece la strutturazione del modello diverge da quella del sistema, la confusione è assicurata).

### 5.1 Gerarchie: versioni UML 1.x

Nelle versioni 1.x di UML (quelle anteriori al 2005), il costrutto gerarchico per eccellenza è il package.

Il Package è un contenitore generico di altri elementi (di qualunque tipo, ad esempio classi, componenti, casi d'uso, nodi). E' rappresentato con l'icona di una cartella. Esattamente come per una cartella in un file system, si possono definire gerarchie di package, ed ogni package definisce un ambito di denominazione univoco (un "namespace"). UML definisce anche una tipologia specifica di package, con stereotipo testuale <<model>> per rappresentare modelli (o la corrispondente icona stereotipo a forma di triangolo, come nella figura seguente).



In UML 1.x, un'altra tipologia particolare di package era il <<subsystem>>. In UML 2.x invece, come si è detto in precedenza, il subsystem non è più un tipo particolare di package, bensì un tipo di componente. E questo fa sì che l'organizzazione dei modelli in UML 2.x sia diversa rispetto a quella tipica nelle versioni 1.x .

La strutturazione tipica dei modelli in UML 1.x si basa su una gerarchia di package. Se l'organizzazione del modello riflette quella del sistema, può assumere la seguente forma:

- Un package di livello "0" (il più aggregato) che rappresenta il sistema;
- N package di livello "1", contenuti nel package "0" che rappresentano i sottosistemi
- Per ogni package-sottosistema non-elementare (cioè scomposto in package di livello inferiore), package in esso contenuti
- Per ogni package elementare, classi o componenti in esso contenuti.

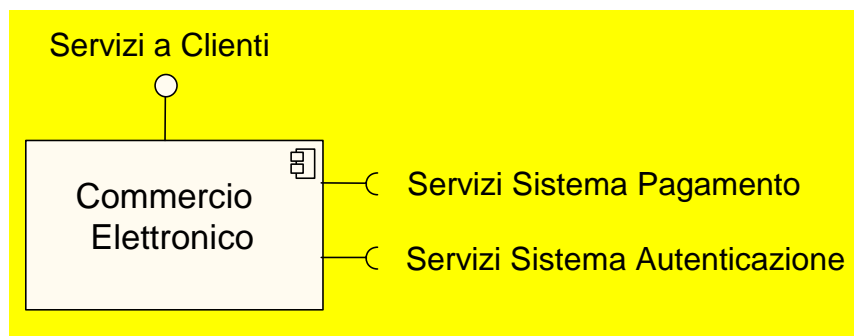
Ogni nodo della gerarchia (ogni package) costituisce un livello al quale possono essere riferiti più diagrammi.

## 5.2 Gerarchie: versioni UML 2.x

In UML 2.0, il package esiste ancora. Ma l'organizzazione dei modelli può essere effettuata in modo più efficace utilizzando il costrutto dei componenti, visto che, come si è già ricordato, la rappresentazione degli elementi a “grana grossa” (sistema, sottosistemi) in UML 2 avviene utilizzando il componente, non più il package. Opportunamente, in UML 2 anche il componente definisce un ambito di denominazione univoco (un “namespace”).

Organizzare il modello del sistema come gerarchia di componenti presenta ulteriori vantaggi, legati al fatto che la scomposizione in parti di un componente (ed in particolare dei componenti aggregati come sistemi e sottosistemi) può venire rappresentata in modo preciso con il diagramma delle strutture composite (“composite structure diagram”). In questo modo, l'alberatura gerarchica del modello può essere anche rappresentata con precisione in una serie di diagrammi di scomposizione.

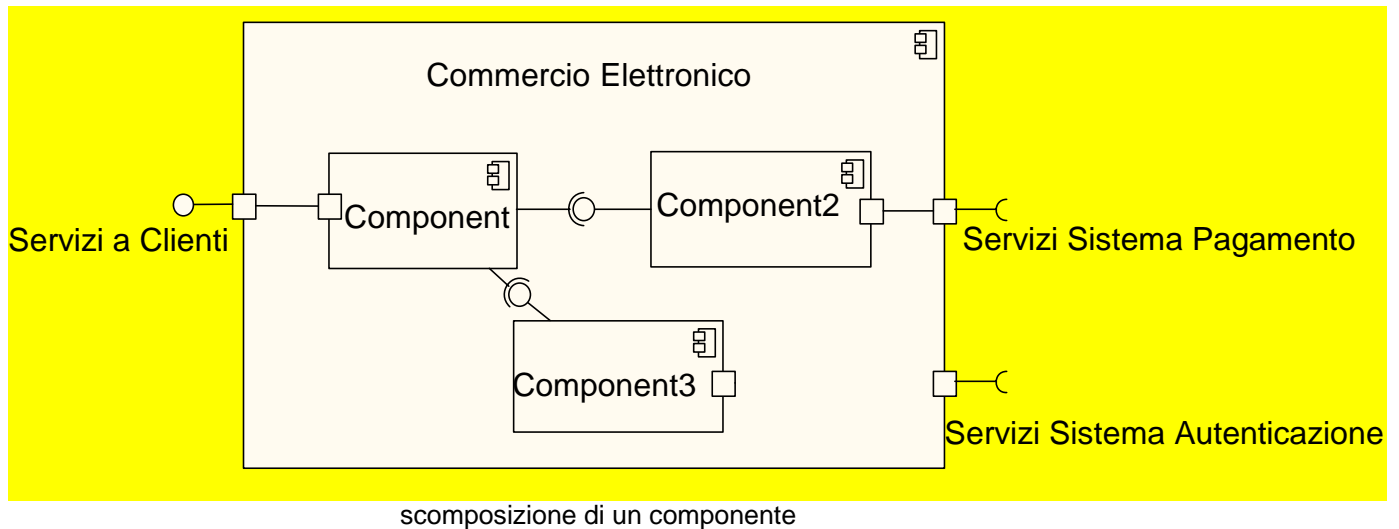
Il diagramma delle strutture composite permette di rappresentare la scomposizione di un componente o di una classe nelle sue parti costitutive, in modo maggiormente preciso rispetto a quanto era possibile nelle precedenti versioni di UML.



rappresentazione di un componente da scomporre

Nella figura è rappresentato un (macro) componente “Commercio Elettronico”, che realizza dei servizi (non rappresentati in dettaglio) definiti nell’interfaccia “Servizi a Clienti”. Per svolgere il proprio lavoro, il componente richiede a sua volta dei servizi ad un sistema di autenticazione e ad un sistema di pagamento (i due sistemi non sono rappresentati, ma sono presenti le rispettive interfacce “required”).

Una possibile scomposizione del componente “Commercio Elettronico” è rappresentata sotto:



Sapevamo (dal diagramma di livello superiore) che il componente “Commercio Elettronico” realizza i servizi definiti nell’interfaccia “Servizi a Clienti”. Il diagramma ci dice che la Parte interna che prende in carico le richieste di servizio è il “Component1”. I quadratini posti sul bordo dei due componenti (quello di livello superiore, “Commercio Elettronico”, e quello di livello inferiore, la Parte “Component1”) sono Porti (inglese Port), cioè punti di interazione tra il componente e l’ambiente in cui opera. La linea che connette i due Porti è chiamata in UML Connettore di delega (Delegation Connector). Il meccanismo permette di precisare, per le interfacce “provided”, quale parte interna di un componente prende in carico i servizi offerti dal componente stesso; per le interfacce “required”, quale parte interna del componente richiede i servizi specificati nelle relative interfacce e realizzati da componenti esterni. Così, il diagramma afferma che i servizi definiti nell’interfaccia “Servizi Sistema Pagamento” vengono richiesti dalla Parte “Component2”, mentre i servizi definiti in “Servizi Sistema Autenticazione” vengono richiesti dalla Parte “Component3”.

Il connettore che lega Component1 e Component2 è chiamato in UML Connettore di assemblaggio (Assembly Connector): Si tratta di una notazione sintetica, che serve per rappresentare connessione tra componenti del medesimo livello, cioè parti del medesimo componente composito. Nel caso specifico, la notazione “ball-and-socket” afferma che il Component1 richiede servizi a Component2.

Il meccanismo di scomposizione può essere ripetuto fino al livello di dettaglio più elementare che si ritiene opportuno rappresentare.

**!!!** Nell’utilizzo semplificato di UML i porti (Port) possono essere tralasciati, se non significativi. E tutti i connettori possono essere rappresentati come semplici linee.

## 6. I diagrammi

Quali sono i diagrammi UML più utili? E' una domanda frequente, alla quale la risposta più frequente è "dipende".

E' vero, dipende. Innanzitutto dalla tipologia di sistema che dobbiamo rappresentare. Ad esempio, se il nostro è un sistema real-time (automazione, robotica) il diagramma di stato assume un'importanza enorme, e diventa il diagramma più importante. Se è un sistema "gestionale" (es. commercio elettronico, gestione pratiche, ...) il diagramma di stato svolge di solito un ruolo meno centrale.

Dipende. Dipende anche dall'organizzazione del progetto, dal fatto di utilizzare (o meno) i modelli ed i diagrammi UML come specifiche di rilevanza contrattuale in un rapporto cliente-fornitore. Dalla cultura dell'azienda. Dal livello di rischio del progetto.

Purtroppo, la risposta alla domanda non è davvero univoca. Solo analizzando le situazioni specifiche si possono fornire risposte sensate. Quindi il discorso potrebbe terminare qui.

Eppure, c'è qualcosa da aggiungere, per fornire alcuni suggerimenti derivati dalla mia esperienza.

### 6.1 Le tipologie di diagramma non sono a compartimenti stagni

Riprendiamo l'elenco dei diagrammi riportato nel capitolo 3.1 :

Diagrammi "strutturali"

- diagramma delle classi (class)
- diagramma degli oggetti (object)
- diagramma dei componenti (component)
- diagramma delle strutture composite (composite structure)
- diagramma di deployment (deployment)
- diagramma dei package (package)

Diagrammi "comportamentali"

- diagramma dei casi d'uso (use case)
- diagramma di stato (statechart)
- diagramma delle attività (activity)
- diagramma di sequenza (sequence)
- diagramma di comunicazione (communication)
- diagramma dei tempi (timing)
- diagramma di sintesi dell'interazione (interaction overview)

} diagrammi di interazione

Si può, volendo, attribuire a ciascun diagramma che compare nell'elenco un grado di importanza. E' un esercizio che è stato fatto da alcuni (non da chi ha maggiormente contribuito a definire il linguaggio, va detto), e può avere una sua utilità. Relativa, però. Perché il nocciolo della questione non è lì.

L'elenco definisce delle tipologie di diagramma. Ma in UML 2 le tipologie di diagramma non sono a compartimenti stagni. Tendono a sfumare, per così dire, l'una nell'altra. E i diagrammi reali, quelli che servono nei progetti concreti, rientrano a fatica nelle tipologie "ufficiali".

Ad esempio tra i diversi diagrammi strutturali, ed in particolare tra il diagramma delle classi e quello dei componenti, non esistono differenze rilevanti. Se un diagramma strutturale contiene più classi, si chiama diagramma delle classi. Se contiene più componenti, si chiama diagramma dei componenti.

Naturalmente è possibile modellare un sistema rappresentando come classe ciò che lo è, ma rappresentando anche (nello stesso diagramma, se lo si ritiene opportuno) come componente ciò che non è implementato con tecnologie Object Oriented (script, moduli C, Cobol, ...). Il diagramma risultante è delle classi o dei componenti?

Analogamente, qual è il confine tra un diagramma dei package e un diagramma delle classi? Se in un diagramma dei componenti rappresento un nodo, diventa automaticamente un diagramma di deployment?

Perfino il confine tra diagrammi strutturali e diagrammi comportamentali è sottile. Il diagramma di comunicazione (uno dei diagrammi di interazione) corrisponde precisamente, per gli aspetti strutturali, ad un diagramma degli oggetti (se rappresenta istanze) o delle classi. Diventa un diagramma di interazione (comportamentale) nel momento stesso in cui vi si aggiunge un singolo messaggio.

In sintesi, le tipologie di diagramma UML non sono a compartimenti stagni. Ma soprattutto, non è detto che sia in quell'elenco che bisogna cercare i diagrammi più utili.

## 6.2 Le macro-famiglie di diagrammi e l'organizzazione dei modelli

Esistono, come si è già accennato in precedenza (3.1), delle macro-famiglie in cui i tredici diagrammi UML 2 possono essere ricompresi:

- Diagrammi strutturali – componenti, classi, strutture composite, package, oggetti, deployment. Servono a rappresentare le parti del sistema e le relazioni che le legano tra loro.
- Diagrammi di interazione – sequenza, comunicazione, timing, sintesi dell'interazione. Servono a rappresentare il modo in cui le parti del sistema interagiscono tra loro (tipicamente, per implementare una specifica funzionalità).
- Diagramma di stato. Servono a rappresentare gli stati in cui una delle parti del sistema si può venire a trovare nel corso della sua esistenza, i possibili passaggi da uno stato ad un altro (transizioni), le operazioni che dipendono dallo stato dell'oggetto.
- Diagramma di attività. Un flow chart, utilizzabile per rappresentare la logica interna di una funzionalità (a qualsiasi livello, dai processi di business alle operazioni software di dettaglio).
- Diagramma dei casi d'uso. Ad essere rigorosi, il diagramma dei casi d'uso presenta molte affinità con i diagrammi strutturali, ma conviene mantenerlo distinto in quanto non rappresenta parti e relazioni tra parti, bensì le modalità di utilizzo del sistema.

Nel capitolo precedente (5) si è affrontato il tema dell'organizzazione gerarchica dei modelli, e si è visto che un sistema può essere visto come una gerarchia di componenti:

- ad un livello zero il sistema complessivo
- ad un primo livello di scomposizione, i sottosistemi che ne costituiscono le parti principali
- ogni sottosistema può essere a sua volta scomposto in parti, e così via

Ogni componente presente nella gerarchia è associabile ad una serie di possibili diagrammi:

- uno o più diagrammi strutturali (componenti, classi), che ne evidenziano le caratteristiche e le relazioni con altri componenti
- un diagramma delle strutture composite che ne rappresenta la eventuale scomposizione in parti interne (se si tratta di un componente o classe scomponibile)
- una serie di diagrammi di interazione, che mostrano come il componente (o classe) interagisce con altre parti del sistema (tipicamente per implementare una specifica funzionalità)
- un diagramma di stato, che rappresenta il ciclo di vita del componente (o classe) e gli eventi che lo riguardano

In altri termini, ciascuno dei componenti che costituiscono il sistema, a qualsiasi livello (dal sistema complessivo ad una delle sue parti elementari) può essere rappresentato in una delle tre viste fondamentali previste in UML: quella strutturale, e le due comportamentali (interazione e stato).

## 6.3 Rappresentazioni del sistema complessivo

### 6.3.1. Diagramma di contesto

Il diagramma di contesto è il diagramma che rappresenta il sistema nella sua globalità, e nelle sue relazioni con sistemi esterni (esseri umani, altri sistemi informatici, oggetti fisici). In altri termini, è il diagramma che definisce i confini del sistema, chiarendo cosa è interno e cosa è esterno.

“Utilizzare il diagramma di contesto come la prima informazione architeturale da presentare al lettore. Il diagramma di contesto serve come punto di partenza per ogni esplorazione di dettaglio all’interno del sistema” [7].

In UML 2.0, un diagramma di contesto può essere rappresentato con un diagramma dei componenti, come nella figura seguente:

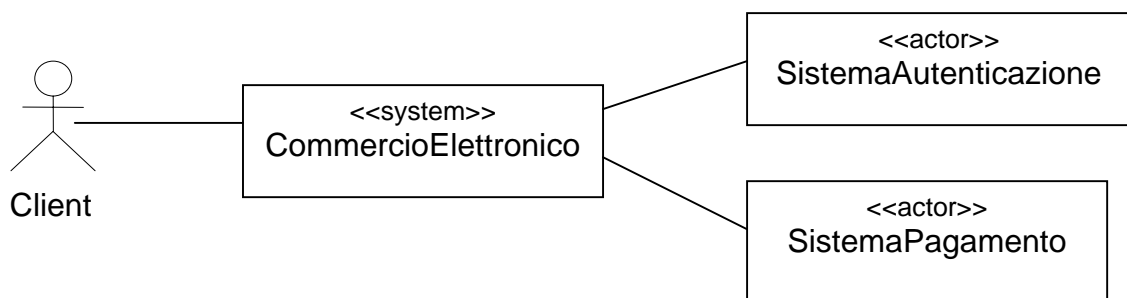


diagramma di contesto

Il diagramma rappresenta il sistema “Commercio Elettronico” come un unico componente, e riporta le associazioni che il sistema ha con il mondo esterno, cioè verso gli attori.

Quando opportuno, la relazione tra sistema e attori può essere rappresentata in modo più dettagliato, esplicitando le interfacce che dichiarano i servizi richiamabili. Nell’esempio seguente, la comunicazione

tra il sistema “Commercio Elettronico” e i due sistemi informatici esterni avviene tramite le interfacce “Servizi Autenticazione” e “Servizi Pagamento”:

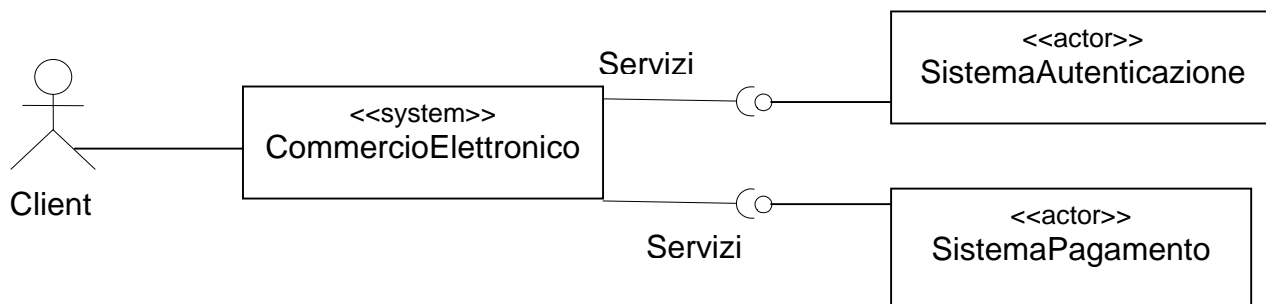


diagramma di contesto con interfacce

### Interazioni del sistema complessivo

Le relazioni del sistema con i suoi attori, espresse a livello strutturale nel diagramma di contesto, possono essere rappresentate dal punto di vista dinamico con i diagrammi di interazione.

Larman [7] suggerisce ad esempio di creare per ogni caso d’uso del sistema complessivo un System Sequence Diagram, cioè un diagramma di sequenza che illustra la dinamica dei messaggi che attori e sistema si scambiano per la realizzazione del caso d’uso in questione. Nei System Sequence Diagram non vengono evidenziate le parti interne del sistema, che è trattato come una “black box”, una scatola nera vista in modo unitario.

**!!!** I diagrammi di interazione a livello di sistema complessivo sono utili solo nei casi in cui l’interazione tra attori e sistema segue un protocollo di comunicazione complesso, con un elevato numero di messaggi. In tutti gli altri casi, i diagrammi di interazione sono più utili a livello di dettaglio, quando rappresentano le parti del sistema coinvolte nell’implementazione di un caso d’uso.

### Ciclo di vita del sistema complessivo

In ambito real-time (automazione, robotica) [3] può essere utile descrivere il ciclo di vita del sistema complessivo (ad esempio un ascensore, una lavatrice) tramite un diagramma di stato.

**!!!** Nel mondo del software gestionale, il ruolo dei diagrammi di stato è più limitato, e viene circoscritto alla descrizione del ciclo di vita di particolari classi del sistema..

#### 6.3.2. Diagramma dei casi d’uso

Il diagramma di contesto rappresenta il sistema nella sua globalità, e le sue relazioni con gli attori. I medesimi elementi possono essere illustrati nel diagramma dei casi d’uso, che costituisce però anche e soprattutto una mappa visuale degli utilizzi del sistema.

Nel diagramma dei casi d’uso, il sistema è rappresentato come un rettangolo (subject), all’interno del quale sono collocate le ellissi che rappresentano i casi d’uso. E’ necessario, per assicurare coerenza al modello complessivo, che gli attori coincidano con quelli evidenziati nel diagramma di contesto.

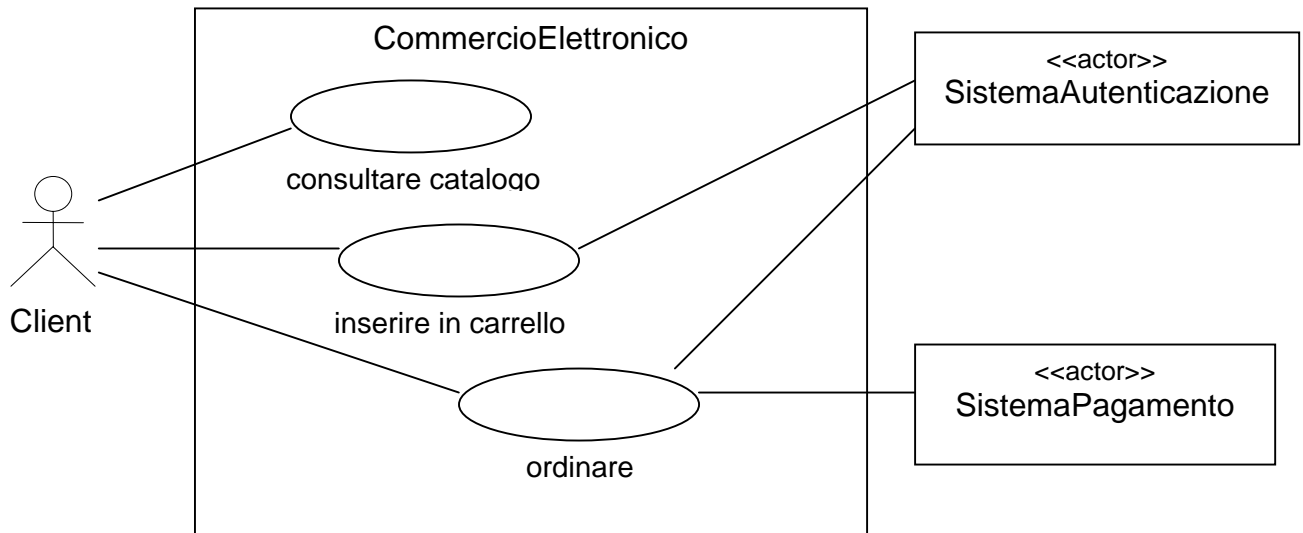


diagramma dei casi d'uso

Il diagramma dei casi d'uso costituisce solo una parte minimale e introduttiva del modello dei casi d'uso, che è un modello fondamentalmente testuale. Per indicazioni di dettaglio sul diagramma dei casi d'uso, si veda la relativa Linea Guida [9].

### 6.3.3. Diagramma di deployment

Rappresenta la configurazione dei nodi hardware su cui viene eseguito il sistema. Può rappresentare il software allocato su tali nodi.

Il diagramma può rappresentare la configurazione hardware del sistema in termini generali (a livello di tipi), oppure fotografare (a livello di istanze) una specifica situazione, in cui vengono descritte specifiche istanze di nodi, ambienti di esecuzione, componenti (o artifact), oggetti.

### 6.3.4. Scomposizione del sistema complessivo

Il sistema unitario rappresentato nel contesto viene scomposto in un diagramma di "primo livello", che evidenzia le parti principali in cui è articolato (i sottosistemi), e le loro relazioni. Il diagramma (si può utilizzare a questo scopo il diagramma delle strutture composite) fornisce una rappresentazione complessiva del sistema a livello macro.

Ogni sottosistema può essere a sua volta scomposto in un nuovo diagramma, che ne rappresenta le parti. Le quali, a loro volta, sono scomponibili in ulteriori diagrammi, sempre più dettagliati. Il procedimento si interrompe al raggiungimento dei componenti elementari (ossia di sufficiente dettaglio, per i quali non vale la pena operare una scomposizione ulteriore).

**!!!** Il diagramma strutturale che rappresenta la scomposizione del sistema complessivo nelle sue parti principali è in assoluto uno dei diagrammi più utili dal punto di vista architettuale.

Per quanto riguarda gli aspetti dinamici, anche per i sottosistemi è possibile, quando opportuno, rappresentare il ciclo di vita tramite un diagramma di stato. Analogamente, quando lo si ritiene utile, possono essere definiti diagrammi di interazione che mostrano lo scambio di messaggi a livello di sottosistemi (ogni diagramma di interazione va comunque riferito ad una sola funzionalità – cioè ad un solo caso d’uso).

Naturalmente, diagrammi di interazione e di stato a livello di subsystem forniscono semplicemente una visione aggregata (sintetica) del comportamento dinamico del sistema. Se si desidera descrivere la dinamica ad un livello di maggiore dettaglio, è necessario utilizzare componenti o classi.

## 6.4 Rappresentazioni di dettaglio

### 6.4.1. Modello di dominio e diagrammi derivati

I diagrammi utili per la rappresentazione del sistema a livello di dettaglio hanno un origine comune nel cosiddetto modello di dominio.

Il modello di dominio nasce tipicamente sotto forma di diagramma delle classi, e in un primo momento il suo ruolo essenziale è quello di fissare il significato dei concetti fondamentali del sistema, e individuare le relazioni tra tali concetti.

Il modello del dominio è il fondamento del linguaggio comune tra tutti gli stakeholder interessati al progetto (informatici e non), e costituisce al tempo stesso il punto di partenza per la progettazione e l’implementazione del sistema.

Nella progettazione ed implementazione del sistema, il modello di dominio si arricchisce grazie a due fattori paralleli:

- la definizione progressiva dei casi d’uso, e la conseguente precisazione dei requisiti
- il consolidamento ed il progressivo perfezionamento dell’architettura complessiva del sistema

La progettazione progressiva dei casi d’uso precisa le responsabilità delle classi, in termini di operazioni, di attributi e di relazioni con le altre parti del sistema.

In questo processo di approfondimenti e integrazioni progressive, il modello di dominio origina come si è detto una serie di diagrammi che rientrano, ancora una volta, nelle macro-famiglie principali di UML:

- diagrammi strutturali (classi, componenti) per rappresentare gli aspetti statici
- diagrammi di interazione per rappresentare gli aspetti dinamici
- diagrammi di stato per rappresentare stati, transizioni ed eventi delle classi che presentano un ciclo di vita significativo

**!!!** I diagrammi di dettaglio possono essere utilizzati, durante un progetto, come specifica per l’implementazione. In questo caso si tende normalmente ad enfatizzare la completezza (pretendendo, ad esempio, almeno un diagramma di interazione per ogni caso d’uso) e la precisione, soprattutto quando la specifica ha una valenza contrattuale.

In altre situazioni, i diagrammi servono come guida per l’implementazione, ma non hanno un ruolo di specifica ufficiale. In questi casi si tendono a produrre meno diagrammi, per comunicare solo gli aspetti più rilevanti del sistema. Normalmente, sono proprio questi (pochi) diagrammi a risultare significativi per esprimere l’architettura del sistema, e a risultare utili per chi lo debba conoscere in futuro.

### 6.4.2. Schema dei dati

Rappresenta le strutture dati “persistenti” del sistema, gestite in un file system o in un DBMS (Data Base Management System - spesso, relazionale).

Lo schema dei dati può essere rappresentato, in termini di notazione UML, con la notazione del diagramma delle classi (va ben distinto, però, dai diagrammi che rappresentano le classi vere e proprie). E’ stato proposto a tal fine uno specifico profilo UML (i profili costituiscono estensioni di UML, che consentono di rappresentare in modo preciso esigenze particolari). Nel profilo di data modeling, come si vede nella figura seguente, gli identificatori vengono indicati con stereotipi.



uno schema dati che utilizza la notazione delle classi

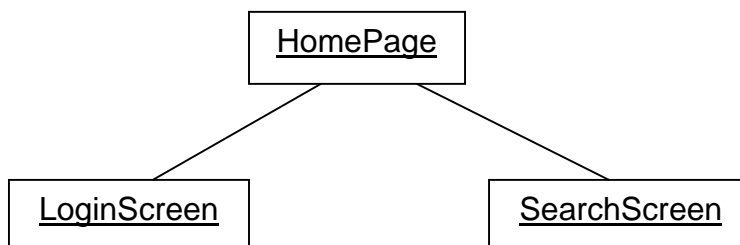
In alternativa, possono essere utilizzate notazioni Entity-Relationship tradizionali.

Lo schema dei dati più utile ai fine della documentazione architeturale è certamente quello logico, corrispondente alle strutture implementate nel DBMS o file system.

### 6.4.3. Diagramma di navigazione delle interfacce utente

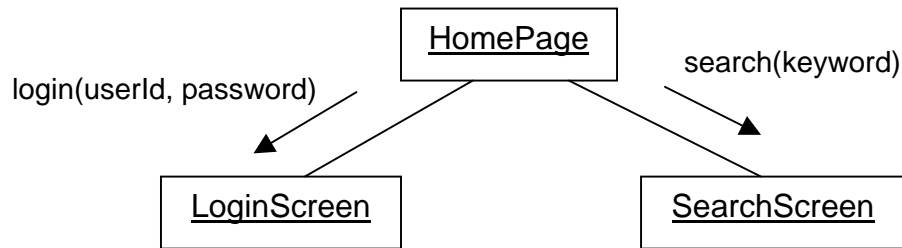
Rappresenta gli elementi (pagine, schermate) principali attraverso cui gli utilizzatori umani interagiscono con il sistema, ed i percorsi “navigabili” che collegano tali elementi.

Il diagramma più efficace per rappresentarlo è il diagramma di comunicazione. Gli elementi di interfaccia utente (pagine) vengono rappresentati come partecipanti all’interazione, ed i percorsi navigabili vengono rappresentati come legami (volendo direzionati).



un diagramma di navigazione

Ad un livello di maggior dettaglio, è possibile specificare per ogni legame navigabile i comandi che consentono di percorrerlo (es. link, redirect), esprimendoli come messaggi.



un diagramma di navigazione con evidenza dei messaggi

#### 6.4.4. Diagrammi di flusso

In molte situazioni, un diagramma di flusso (flow chart) può tornare utile. Ad esempio quando la descrizione di uno scenario (di un caso d'uso) è particolarmente complessa, e si vuole rappresentarne la logica in modo visuale. O quando si vuole rappresentare il workflow di una pratica. O quando si progetta la logica interna di un modulo, di un'operazione.

Il diagramma di flusso, in UML, è chiamato diagramma di attività. Tra tutti i diagrammi previsti da UML è probabilmente il più versatile (utilizzabile in situazioni molto diverse) e il più intuitivo (facilmente comprensibile anche per chi non conosce UML).

### 6.5 Una nota per concludere

Le indicazioni riportate in questa linea guida intendono aiutare ad utilizzare UML in contesti di lavoro reali. Progetti di sviluppo, attività di evoluzione in cui UML può fornire un contributo effettivo in termini di comunicazione e documentazione.

Ho quindi adottato un punto di vista fortemente pragmatico, privilegiando la semplificazione e la concretezza rispetto alla precisione nei dettagli ed alla completezza dal punto di vista teorico.

Per chi debba utilizzare UML in un ambito di maggiore formalizzazione, o approfondirne le caratteristiche in ambito accademico, le indicazioni proposte qui sono insufficienti, e vanno integrate con una conoscenza diretta della documentazione ufficiale del linguaggio [1].