



# Ingegneria del Software

Bumma Giuseppe

## Contents

1 Diagrammi UML .....	6
1.1 Premessa .....	6
1.2 Unified Modeling Language .....	7
1.3 Diagrammi di UML 2.5.1 .....	7
1.4 UML 2.5.1 e Visio .....	8
1.5 Diagramma dei Package .....	8
1.5.1.1 Package .....	8
1.5.1.2 Diagramma dei Package .....	9
1.5.1.3 Dipendenze .....	9
1.5.1.4 Diagramma dei Package .....	10
1.5.1.5 Esempio .....	10
1.6 Interfaccia .....	11
1.6.1.1 Notazione .....	11
1.7 Diagramma delle Classi .....	11
1.7.1.1 Esempio .....	12
1.7.1.2 Classe .....	12
1.7.2 Attributi .....	13
1.7.2.1 Molteplicità .....	13
1.7.2.2 Visibilità .....	13
1.7.2.3 Attributi: Molteplicità .....	13
1.7.2.4 Operazioni .....	14
1.7.2.5 Operazioni e Attributi Statici .....	14
1.7.2.6 Associazioni .....	14
1.7.2.7 Associazioni Bidirezionali .....	15
1.7.2.8 Associazioni Ternarie .....	15
1.7.2.9 Associazioni: molteplicità .....	16
1.7.3 Classi di Associazione .....	16
1.7.4 Aggregazione e Composizione .....	16
1.7.5 Generalizzazione .....	17
1.7.6 Generalizzazione Multipla .....	18
1.7.6.1 Relazione tra classi: sintassi .....	19
1.7.7 Classi Astratte .....	19
1.7.8 Enumerazioni .....	20
1.8 Diagramma di sequenza .....	20
1.8.1 Esempio .....	21
1.8.2 Lifeline .....	21
1.8.2.1 Vincoli Temporali .....	21
1.8.3 Riferimento ad altri Diagrammi .....	22
1.8.3.1 Messaggio .....	23
1.8.3.1.1 Tipi di Messaggio .....	23
1.8.4 Combined Fragment .....	25
1.8.4.1 Tipi .....	25
1.9 Diagrammi di Stato .....	27

1.9.1.1 Esempio .....	28
1.9.1.2 Concetti .....	28
1.9.1.3 Esempio .....	29
1.9.1.4 Stato .....	29
1.9.1.5 Esempio .....	30
1.9.1.6 Esempio .....	30
1.9.1.7 Esempio .....	31
1.9.1.8 Pseudostato .....	31
1.9.1.9 Tipi di Pseudostati .....	31
1.9.1.10 Transizione .....	33
1.9.1.11 Tipi di Eventi .....	34
1.9.1.12 Azione .....	34
1.9.1.13 Diagramma di Stato .....	35
1.10 Diagramma delle attività .....	36
1.10.1.1 Notazione .....	37
1.10.2 Object Flow .....	38
1.10.3 Segnali ed Eventi .....	38
1.10.4 Exception Handler .....	38
1.10.5 InterruptibleActivityRegion .....	40
2 Progetto .....	40
2.1 Analisi dei requisiti .....	40
2.1.1 Requisiti .....	40
2.1.1.1 Requisiti di Sistema .....	41
2.1.1.2 Requisiti Funzionali .....	41
2.1.1.3 Requisiti Non Funzionali .....	41
2.1.1.4 Requisiti di Dominio .....	42
2.1.2 Analisi .....	42
2.1.2.1 Raccolta dei Requisiti .....	42
2.1.2.2 Validazione dei Requisiti .....	43
2.1.2.3 Documento dei Requisiti .....	43
2.1.2.4 Cambiamento dei Requisiti .....	44
2.1.2.5 Analisi del Dominio .....	44
2.1.2.6 Analisi dei Requisiti .....	45
2.1.2.7 Vocabolario .....	45
2.1.2.8 Analisi e gestione dei rischi .....	45
2.1.3 Sicurezza e Privacy .....	46
2.1.3.1 Nuova normativa .....	46
2.1.3.1.1 General Data Protection Regulation .....	46
2.1.3.1.2 Principi .....	47
2.1.3.1.3 Articolo 25 .....	47
2.1.3.1.4 Articolo 32: .....	47
2.1.3.1.5 Trasferimento Dati a Paesi Terzi .....	48
2.1.3.2 Concetti di Base .....	48
2.1.3.2.1 Sicurezza Informatica .....	48
2.1.3.2.2 Violazioni .....	49
2.1.3.2.3 Fattori Influenti .....	49
La Catena degli Anelli .....	49
2.1.3.2.4 Metodi di Protezione .....	49

2.1.3.2.5 Protezione Fisica .....	50
2.1.3.2.6 Autenticazione Forte .....	50
2.1.3.3 Crittografia .....	51
2.1.3.3.1 Crittografia Simmetrica .....	51
2.1.3.3.2 Crittografia Asimmetrica .....	51
2.1.3.3.3 Confronto tra i due tipo di crittografia .....	52
2.1.3.3.4 Biometria .....	53
2.1.3.3.5 Sicurezza delle Password .....	53
2.1.3.4 Analizzare e Progettare la Sicurezza .....	53
2.1.3.4.1 Sistema Sicuro .....	53
2.1.3.4.2 Sistemi Critici .....	54
2.1.3.5 Introduzione alla Security Engineering .....	55
2.1.3.5.1 Security Engineering .....	55
2.1.3.5.2 Applicazione e Infrastruttura .....	56
2.1.3.5.3 Gestione della Sicurezza .....	56
2.1.3.6 Glossario e Minacce .....	56
2.1.3.6.1 Glossario della Sicurezza .....	56
2.1.3.6.2 Tipi di Minacce .....	57
2.1.3.6.3 Tipi di Controllo .....	57
Esempio .....	57
2.1.3.7 Analisi del Rischio .....	58
2.1.3.8 Identificazione del bene .....	59
2.1.3.8.1 Analisi del Sistema Informatico .....	59
2.1.3.8.2 Analisi delle Risorse Fisiche .....	59
2.1.3.8.3 Analisi delle Risorse Logiche .....	60
2.1.3.8.4 Analisi delle Dipendenze tra Risorse .....	60
2.1.3.9 Identificazione delle minacce .....	60
2.1.3.9.1 Attacchi Intenzionali .....	60
2.1.3.9.2 Attacchi a Livello Logico .....	60
2.1.3.9.3 Eventi Accidentali .....	61
2.1.3.9.4 Valutazione dell'Esposizione .....	61
2.1.3.9.5 Valutazione delle Probabilità: Attacchi Intenzionali .....	61
2.1.3.9.6 Individuazione del Controllo .....	61
2.1.3.9.7 Valutazione del Rapporto Costo/Efficacia .....	61
2.1.3.9.8 Costi Nascosti .....	62
2.1.3.9.9 Controlli di Carattere Organizzativo .....	62
2.1.3.9.10 Controlli di Carattere Tecnico .....	62
2.2 Analisi del problema .....	72
2.2.1 Introduzione .....	72
2.2.2 Passi dell'Analisi del Problema .....	73
2.2.2.1 Architettura Logica .....	74
2.2.2.2 Piano di Lavoro .....	74
2.2.2.3 Piano del Collaudo .....	74
2.2.3 Analisi Documento dei Requisiti .....	76
2.2.3.1 Analisi delle Funzionalità .....	76
2.2.3.2 Analisi delle Funzionalità .....	77
2.2.3.3 Esempio .....	77
2.2.3.4 Analisi dei Vincoli .....	78

2.2.3.5 Esempio .....	79
2.2.3.6 Analisi delle Interazioni .....	79
2.2.3.7 Esempio .....	80
2.2.3.8 Esempio .....	80
2.2.4 Analisi Ruoli e Responsabilità .....	81
2.2.4.1 Analisi dei Ruoli e Responsabilità .....	81
2.2.4.2 Esempio .....	81
2.2.4.3 Esempio .....	82
2.2.5 Scomposizione del Problema .....	82
2.2.6 Creazione Modello del Dominio .....	83
2.2.6.1 Individuazione delle Classi .....	84
2.2.6.2 Individuazione delle Classi: Villaggio Turistico .....	85
2.2.6.3 Individuazione delle Relazioni .....	86
2.2.6.4 Individuazione dell'Ereditarietà .....	87
2.2.6.5 Ereditarietà: Villaggio Turistico .....	87
2.2.6.6 Individuazione delle Associazioni .....	87
2.2.7 Individuazione delle Associazioni .....	88
2.2.7.1 1 ° Esempio di Associazione .....	88
2.2.7.2 2 ° Esempio di Associazione .....	89
2.2.7.3 Associazioni: Villaggio Turistico .....	89
2.2.7.4 Individuazione Collaborazioni .....	90
2.2.7.5 Individuazione degli Attributi .....	91
2.2.7.6 Individuazione degli Attributi: Villaggio Turistico .....	93
2.2.7.7 Individuazione delle Operazioni .....	94
2.2.7.8 Individuazione delle Operazioni .....	95
2.2.7.9 Esempio .....	95
2.2.8 Architettura Logica: Struttura .....	97
2.2.8.1 BCE .....	97
2.2.8.2 Layer .....	98
2.2.8.3 Struttura: Package .....	98
2.2.8.4 Esempio .....	100
2.2.8.5 Struttura: Classi .....	100
2.2.8.6 Esempio .....	100
2.2.9 Architettura Logica: Interazione .....	101
2.2.9.1 Esempio .....	101
2.2.10 Architettura Logica: Comportamento .....	102
2.2.10.1 Esempio .....	102
2.2.11 Definizione del Piano di Lavoro .....	103
2.2.11.1 Esempio .....	103
2.2.12 Definizione del Piano del Collaudo .....	104
2.2.12.1 Definizione Piano del Collaudo .....	104
2.2.12.2 JUnit .....	105
2.2.12.3 NUnit .....	105
3 Framework .NET .....	105
3.1 Introduzione .....	105
3.1.1 Tecnologia COM - Component Object Model .....	105
3.1.2 Cos'è il Framework .NET .....	106
3.1.3 Standard ECMA-335 .....	107

3.1.4 Codice interpretato .....	108
3.1.5 Codice nativo .....	108
3.1.6 Codice IL .....	108
3.1.7 Assembly .....	109
3.1.8 Metadati .....	110
3.1.9 Chi usa i metadati? .....	110
3.1.10 Esempio Assembly .....	111
3.1.11 Dove trovare gli Assembly .....	111
3.1.12 Deployment semplificato .....	111
3.1.13 Common Language Runtime .....	111
3.1.14 Garbage Collector .....	112
3.1.15 Gestione delle eccezioni .....	113
3.1.16 Gestione delle eccezioni .....	113
3.1.17 Common Type System .....	113
3.1.18 Common Language Specification .....	114
3.1.18.1 Tipi nativi .....	114
3.1.19 Common Type System .....	114
3.1.19.1 Tipi valore .....	115
3.1.19.2 Tipi valore vs tipi riferimento .....	115
3.1.19.3 Common Type System .....	115
3.1.19.4 Tipi valore e tipi riferimento .....	117
3.1.20 Boxing / Unboxing .....	118
3.1.21 Bibliografia .....	118
3.2 Garbage Collection .....	119
3.2.1 Utilizzo di un oggetto .....	119
3.2.2 Ciclo di vita di un oggetto .....	119
3.2.3 Allocazione della memoria .....	119
3.2.4 Inizializzazione della memoria .....	120
3.2.5 Definite Assignment .....	120
3.2.6 Clean up dello stato .....	121
3.2.7 Liberazione della memoria .....	121
3.2.8 Garbage Collection .....	121
3.2.9 GC: Reference counting .....	122
3.2.10 GC: Tracing .....	122
3.2.11 Allocazione della memoria .....	123
3.2.12 Garbage Collector .....	124
3.2.12.1 Fase 1: Mark .....	124
3.2.12.2 Fase 2: Compact .....	125
3.2.13 Finalization .....	125
3.2.14 Rilascio deterministico senza gestione delle eccezioni .....	127
3.2.15 Rilascio deterministico con gestione delle eccezioni .....	127
3.2.16 Il pattern Dispose .....	127
3.2.17 Rilascio deterministico con using .....	127
3.2.18 Il pattern Dispose (altro esempio di utilizzo) .....	128

# 1 Diagrammi UML

## 1.1 Premessa

- In questo blocco vedremo i principali Diagrammi UML per la fasi di Analisi del Problema e del Progetto
  - per ora non vedremo i diagrammi di deployment e dei componenti
- In particolare:
  - Diagramma dei Package e Diagramma delle Classi → parte “**statica**” dell’Architettura Logica e dell’Architettura del Sistema
    - definiscono le entità di sistema senza analizzare come possano interagire tra loro
  - Diagramma di Sequenza → parte “**interazione**” dell’Architettura Logica e dell’Architettura del Sistema
    - definiscono come le entità, definite nella parte precedente, interagiscono tra loro
  - Diagramma di Stato e Diagramma delle Attività → parte “**comportamentale**” dell’Architettura Logica e dell’Architettura del Sistema
    - definiscono il comportamento delle entità del sistema
- Per ogni Diagramma vedremo solo i concetti fondamentali
- In caso di dubbi fare riferimento alla specifica UML 2.5.1

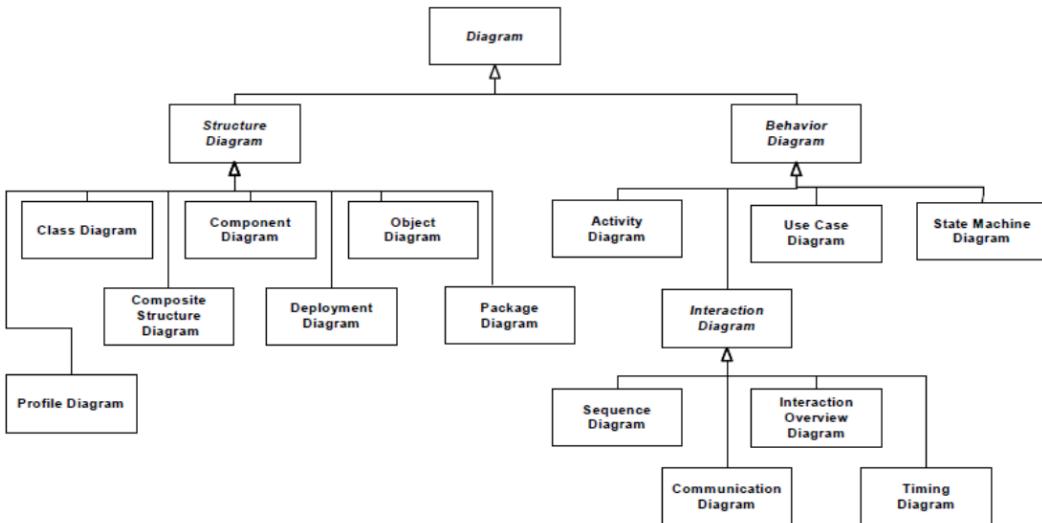
## 1.2 Unified Modeling Language

- È un **linguaggio** che serve per visualizzare, specificare, costruire, documentare un sistema e gli elaborati prodotti durante il suo sviluppo
- Ha una semantica e una notazione standard, basate su un metamodello integrato, che definisce i costrutti forniti dal linguaggio
- La notazione (e la semantica) è estensibile e personalizzabile
- È utilizzabile per la modellazione durante tutto il ciclo di vita del software (dai requisiti al testing) e per piattaforme e domini diversi
- Combina approcci di:
  - modellazione dati (Entity/Relationship)
  - business Modeling (workflow)
  - modellazione a oggetti
  - modellazione di componenti
- Prevede una serie di diagrammi standard, che mostrano differenti viste architettoniche del modello di un sistema
- UML è un linguaggio e non un processo di sviluppo
- UML propone un ricco insieme di elementi a livello utente; tuttavia è alquanto informale sul modo di utilizzare al meglio i vari elementi
  - ciò implica che per comprendere un diagramma un lettore deve conoscere il contesto in cui esso è collocato

## 1.3 Diagrammi di UML 2.5.1

- Diagrammi di struttura:
  - diagramma delle classi (class)
  - diagramma delle strutture composite (composite structure)
  - diagramma dei componenti (component)
  - diagramma di deployment (deployment)

- diagramma dei package (package)
- diagramma dei profili (profile)
- Diagrammi di comportamento:
  - diagramma dei casi d'uso (use case)
  - diagramma di stato (state machine)
  - diagramma delle attività (activity)
  - diagrammi di interazione:
    - diagramma di comunicazione (communication)
    - diagramma dei tempi (timing)
    - diagramma di sintesi delle interazioni (interaction overview)
    - diagramma di sequenza (sequence)



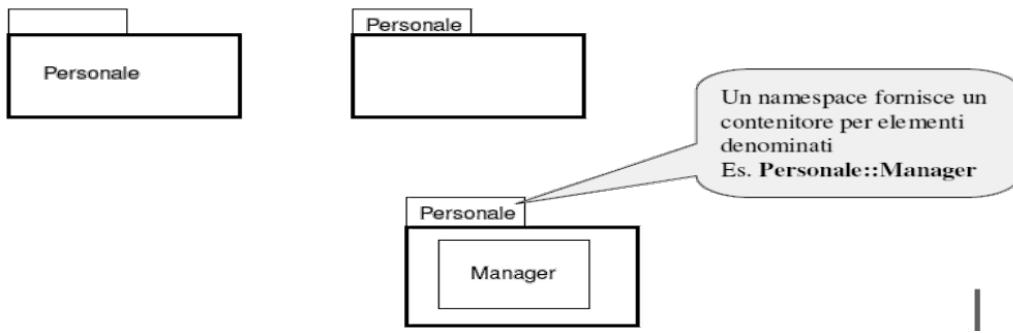
## 1.4 UML 2.5.1 e Visio

- Visio 2016 gestisce nativamente solo alcuni dei Diagrammi di UML 2.5.1
  - classi
  - attività
  - sequenza
  - stato
  - casi d'uso
- È possibile però scaricare uno stencil per poter gestire anche gli altri diagrammi:  
<http://softwarestencils.com/uml/index.html>

## 1.5 Diagramma dei Package

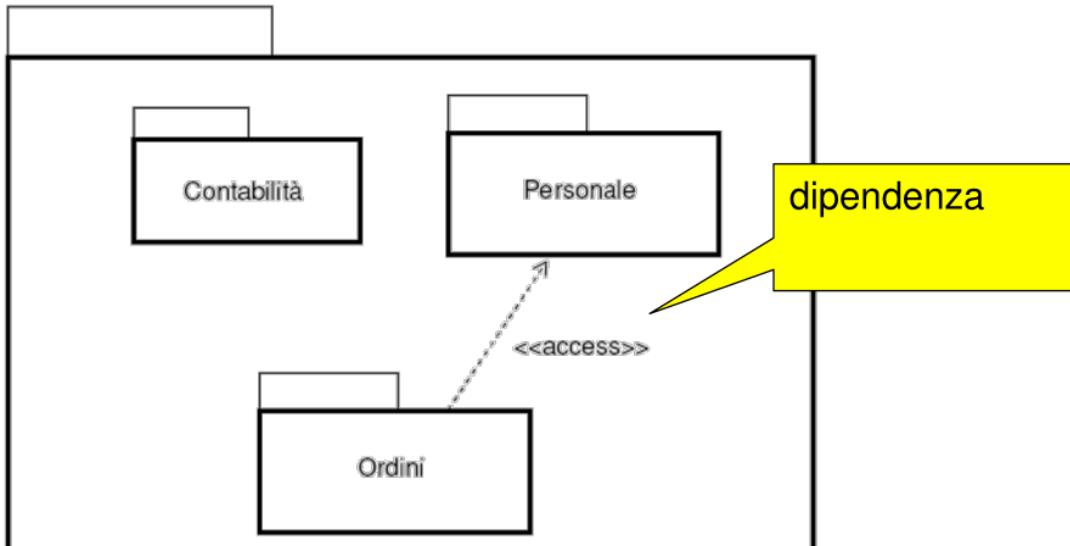
### 1.5.1.1 Package

- Un package, in programmazione, è un contenitore di classi
- Un package, in generale, è utilizzato per raggruppare elementi e fornire loro un namespace
- Un package può essere innestato in altri package
- NAMESPACE: è una porzione del modello nella quale possono essere definiti e usati dei nomi
- In un namespace ogni nome ha un significato univoco



### 1.5.1.2 Diagramma dei Package

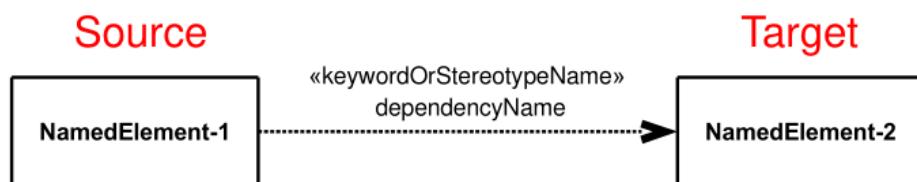
- Un diagramma dei package è un diagramma che illustra come gli elementi di modellazione sono organizzati in package e le relazioni (dipendenze) tra i package



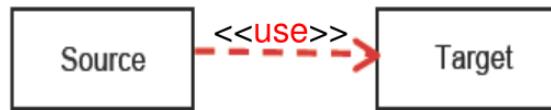
- Un diagramma può rappresentare package con diversi livelli di astrazione

### 1.5.1.3 Dipendenze

- UML permette di rappresentare relazioni che NON sussistono fra istanze nel dominio rappresentato, ma sussistono fra gli elementi del modello UML stesso o fra le astrazioni che tali elementi rappresentano
- Dipendenza: è rappresentata da una linea tratteggiata orientata che va da un elemento dipendente (Source) ad uno indipendente (Target)



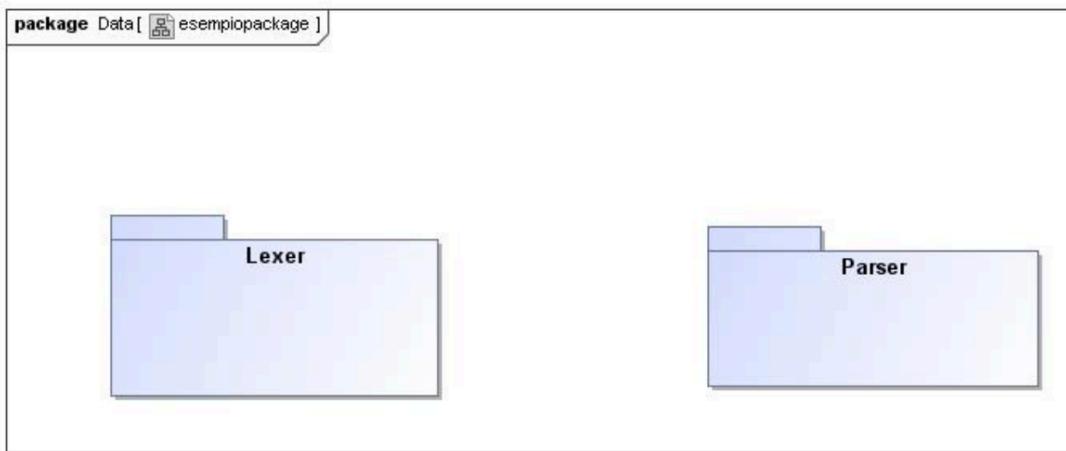
- Una dipendenza indica che cambiamenti dell'elemento **indipendente** influenzano l'elemento **dipendente**
  - modificano il significato dell'elemento dipendente
  - causano la necessità di modificare anche l'elemento dipendente perché i significati sono dipendenti
- UML mette a disposizione nove diversi tipi di dipendenza, ma per i nostri fini consideriamo quasi sempre <>



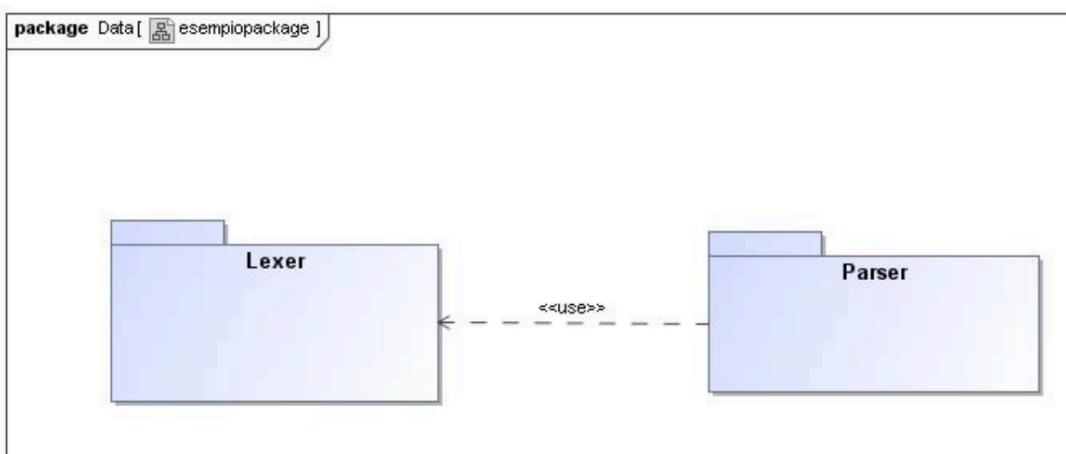
#### 1.5.1.4 Diagramma dei Package

- Tutti i diagrammi che utilizzeremo nella fase di analisi saranno trasformati per venir utilizzati nella fase di progettazione (in analisi definisco il “cosa” e in progettazione definisco il “come”)
- Quando si usa il diagramma dei package per definire la parte strutturale dell’architettura logica ricordare sempre che si stanno esprimendo **dipendenze logiche** che sussistono tra le entità del problema
- Non è detto che tali dipendenze rimangano tali anche nella fase di progettazione
- Esempio: in sistema per l’interpretazione di una grammatica si hanno due parti fondamentali
  - Lexer → strumento che legge e spezza una sequenza di caratteri in sotto-sequenze dette “token”
  - Parser prende in ingresso i token generati, li analizza e li elabora in base ai costrutti specificati nella grammatica stessa

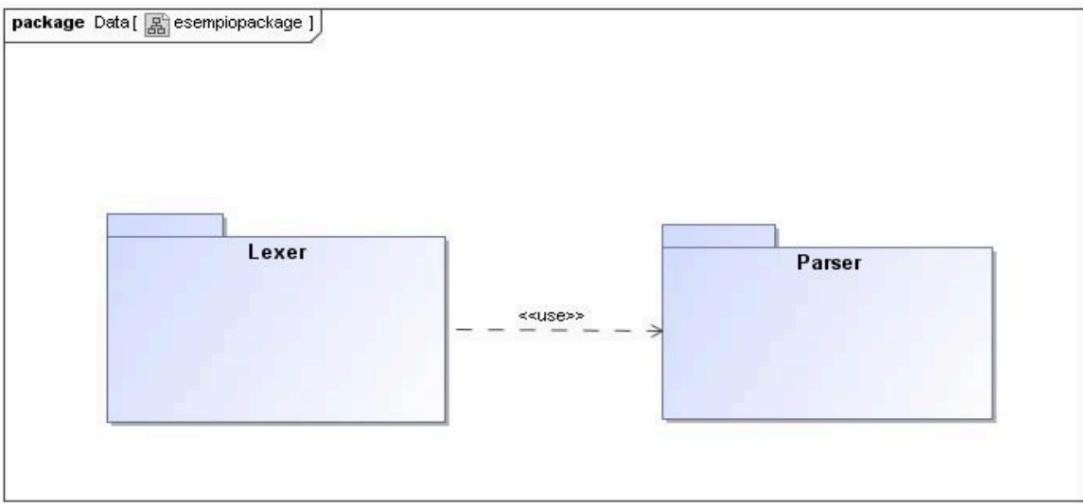
#### 1.5.1.5 Esempio



È ovvio che è il Parser a usare il Lexer, quindi il diagramma prodotto nella fase di analisi sarà così



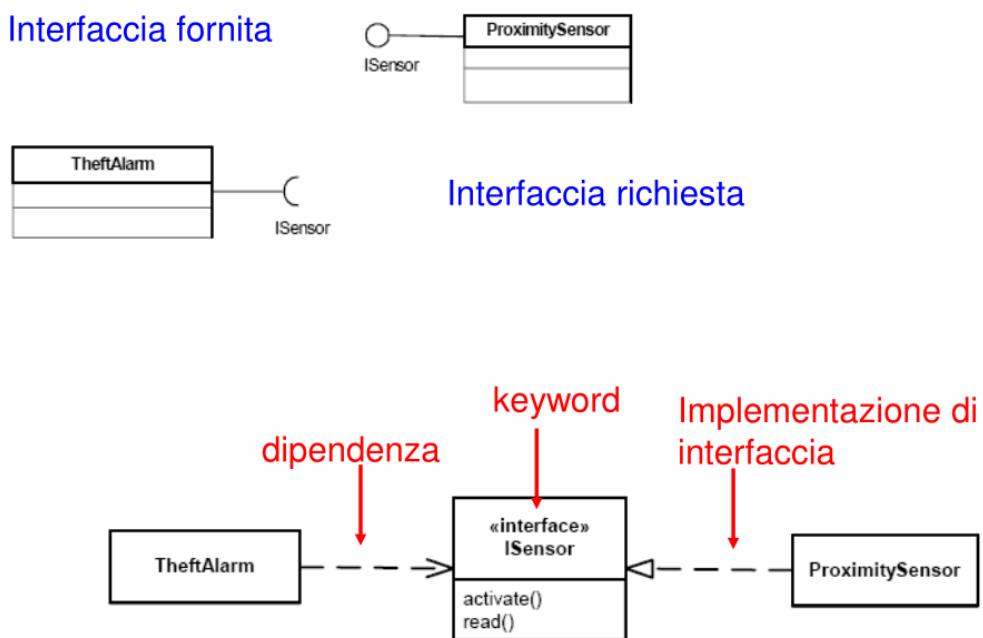
Ma nel diagramma del progetto è il Lexer a fare il Parser



## 1.6 Interfaccia

- Le interfacce forniscono un modo per partizionare e caratterizzare gruppi di proprietà
- Un'interfaccia non deve specificare come possa essere implementata, ma semplicemente quello che è necessario per poterla realizzare
- Le entità che realizzano l'interfaccia dovranno fornire una “**vista pubblica**”(attributi, operazioni, comportamento osservabile all'esterno) conforme all'interfaccia stessa
- Se un'interfaccia dichiara un attributo, non significa necessariamente che l'elemento che realizza l'interfaccia debba avere quell'attributo nella sua implementazione, ma solamente che esso apparirà così a un osservatore esterno

### 1.6.1.1 Notazione



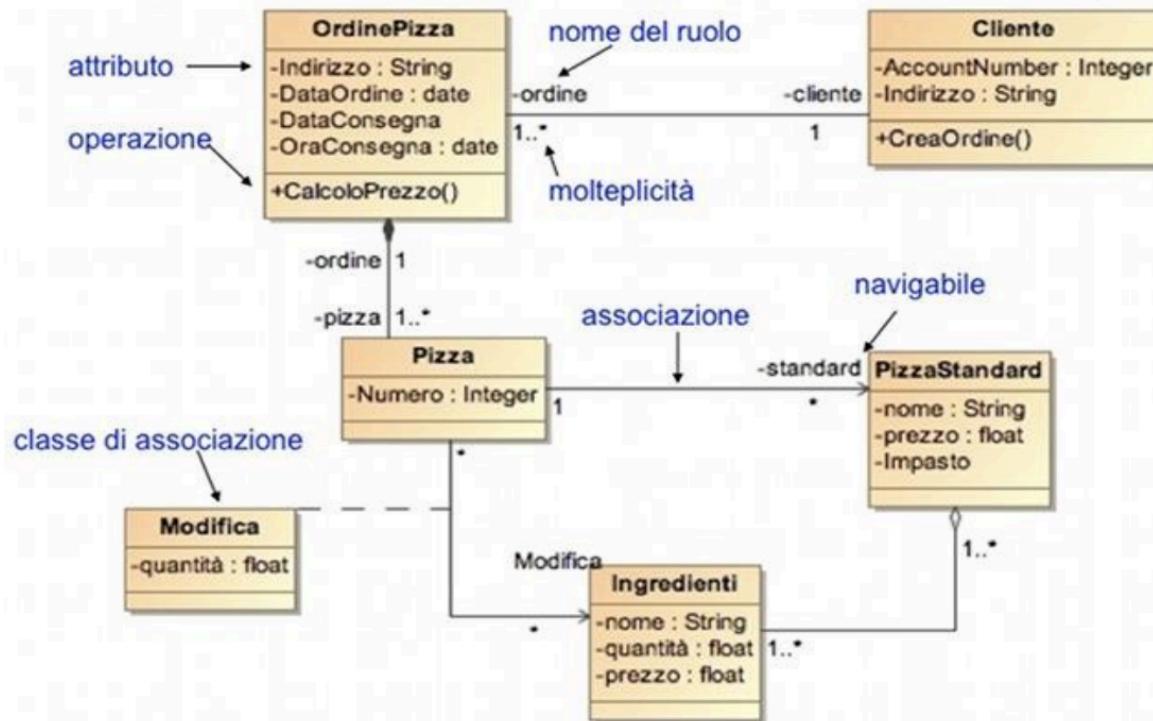
Il diagramma sopra e il diagramma sotto hanno lo stesso significato

## 1.7 Diagramma delle Classi

- Un diagramma delle classi descrive il tipo degli oggetti facenti parte di un sistema e le varie tipologie di relazioni statiche tra di essi
- I diagrammi delle classi mostrano anche le proprietà e le operazioni di una classe e i vincoli che si applicano alla classe e alle relazioni tra classi

- Le proprietà rappresentano le caratteristiche strutturali di una classe:
  - sono un unico concetto, rappresentato però con due notazioni molto diverse: attributi e associazioni
  - benché il loro aspetto grafico sia molto differente, concettualmente sono la stessa cosa

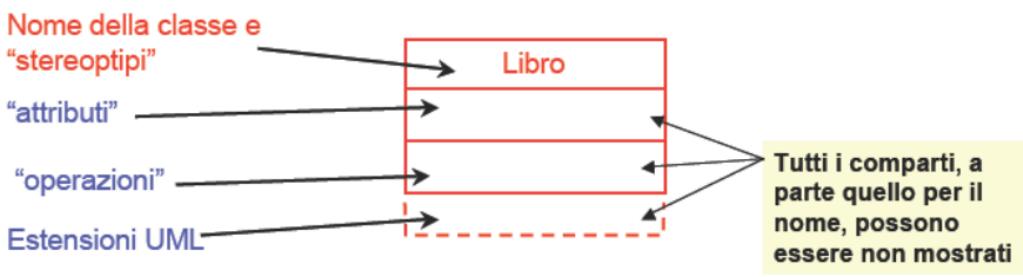
### 1.7.1.1 Esempio



- La molteplicità da **OrdinePizza** a **Cliente** è **uno a molti**: un cliente può avere da 1 a  $n$  ordini
- La molteplicità da **Cliente** a **OrdinePizza** è **univoca**: un ordine è relativo a uno e un solo cliente
- Associazione di **composizione** tra **Pizza** e **OrdinePizza**
- Associazione **navigabile** da **Pizza** a **PizzaStandard**: da **Pizza** ci interessa capire qual è la **PizzaStandard**, ma da **PizzaStandard** non ci interessa andare a **Pizza**, cioè ci interessa sapere ogni **Pizza** da quale **PizzaStandard** deriva
- Associazione di **aggregazione** tra **PizzaStandard** e **Ingredienti**
- Ogni **Pizza** può avere delle modifiche di ingredienti
- Classe di associazione** **Modifica**, che aggiunge attributi e/o comportamenti a quell'associazione
  - Per ogni **Pizza**, ogni **Ingrediente** è presente in una certa quantità, che può essere modificata

### 1.7.1.2 Classe

- Una **classe** modella un insieme di entità (le istanze della classe) aventi tutti lo stesso tipo di caratteristiche (attributi, associazioni, operazioni...)
- Ogni classe è descritta da:
  - un nome
  - un insieme di caratteristiche (feature): attributi, operazioni, ...



### 1.7.2 Attributi

- La notazione degli attributi descrive una proprietà con una riga di testo all'interno del box della classe
- La forma completa è:
  - visibilità nome:tipo molteplicità==default {stringa di proprietà}
- Un esempio di attributo è:
  - stringa: String[10] == "Pippo"{readOnly}
- L'unico elemento necessario è il nome
  - Visibilità: attributo pubblico (+), privato (-) o protected( == )
  - Nome: corrisponde al nome dell'attributo
  - Tipo: vincolo sugli oggetti che possono rappresentare l'attributo
  - Default: valore dell'attributo in un oggetto appena creato
  - Stringa di proprietà: caratteristiche aggiuntive (readOnly)
  - Molteplicità: ...

#### 1.7.2.1 Molteplicità

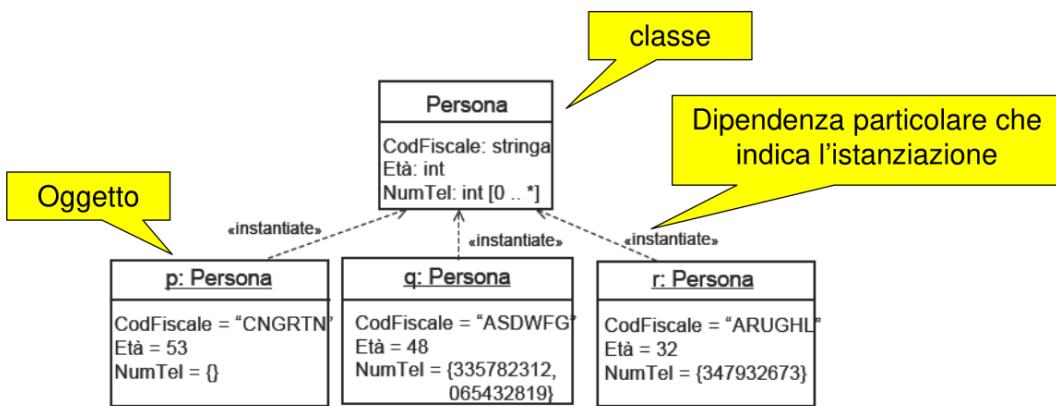
- È l'indicazione di quanti oggetti possono entrare a far parte di una proprietà
- Le molteplicità più comuni sono:
  - 1, 0..1, \*
- In modo più generale, le molteplicità si possono definire indicando gli estremi inferiore e superiore di un intervallo (per esempio 2..4).
- Molti termini si riferiscono alla molteplicità degli attributi:
  - Opzionale: indica un limite inferiore di 0
  - Obbligatorio: implica un limite inferiore di 1 o più
  - A un solo valore: implica un limite superiore di 1
  - A più valori: implica che il limite superiore sia maggiore di 1, solitamente

#### 1.7.2.2 Visibilità

- È possibile etichettare ogni operazione o attributo con un identificatore di visibilità
- UML fornisce comunque quattro abbreviazioni per indicare la visibilità:
  - + (public)
  - - (private)
  - ~ (package)
  - # (protected)

#### 1.7.2.3 Attributi: Molteplicità

- Nelle istanze, il valore di un attributo multi-valore si indica mediante un insieme

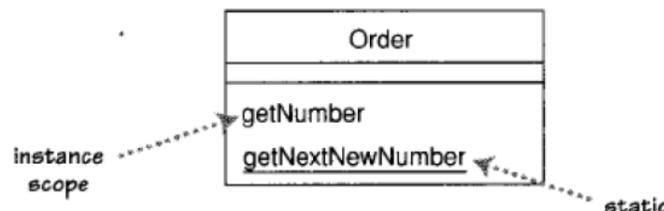


#### 1.7.2.4 Operazioni

- Le operazioni sono le azioni che la classe sa eseguire, e in genere si fanno corrispondere direttamente ai metodi della corrispondente classe a livello implementativo
- Le operazioni che manipolano le proprietà della classe di solito si possono dedurre, per cui non sono incluse nel diagramma
- La sintassi UML completa delle operazioni è **visibilità nome (lista parametri) : tipo ritorno {stringa di propri}**
  - Visibilità: operazione pubblica (+) o privata (-)
  - Nome: stringa
  - Lista parametri: lista parametri dell'operazione
  - Tipo di ritorno: tipo di valore restituito dall'operazione, se esiste
  - Stringa di proprietà: caratteristiche aggiuntive che si applicano all'operazione

#### 1.7.2.5 Operazioni e Attributi Statici

- UML chiama **static** un'operazione o un attributo che si applicano a una classe anziché alle sue istanze
- Questa definizione equivale a quella dei membri statici nei linguaggi come per esempio java e C
- Le caratteristiche statiche vanno sottolineate sul diagramma



#### 1.7.2.6 Associazioni

- Le associazioni sono un altro modo di rappresentare le proprietà
- Gran parte dell'informazione che può essere associata a un attributo si applica anche alle associazioni
- Un'associazione è una linea continua che collega due classi, orientata dalla classe sorgente a quella destinazione
- Il nome e la molteplicità vanno indicati vicino all'estremità finale dell'associazione:
- la classe destinazione corrisponde al tipo della proprietà

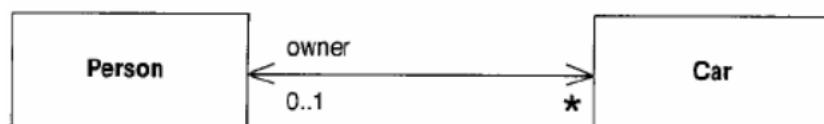


- È possibile assegnare un nome all’associazione e anche assegnare dei nomi ai “ruoli” svolti da ciascun elemento di un associazione
- Gran parte delle cose ce valgono per gli attributi valgono anche per le associazioni
- Anche nei casi in cui non è strettamente necessario, il ruolo può essere utile per aumentare la leggibilità del diagramma
- Ovviamente quando abbiamo un associazione il tipo della proprietà corrisponde alle classi che sono dall’altra parte della associazione, cioè se guardiamo a la proprietà nella classe A sarà di tipo classe B, e viceversa se guardiamo a classe B la proprietà sarà di tipo classe A



### 1.7.2.7 Associazioni Bidirezionali

- Una tipologia di associazione è quella **bidirezionale (o binaria)**, costituita da una coppia di proprietà collegate, delle quali una è l’inversa dell’altra
- Il collegamento inverso implica che, se seguite il valore di una proprietà e poi il valore della proprietà collegata, dovreste ritornare all’interno di un insieme che contiene il vostro punto di partenza

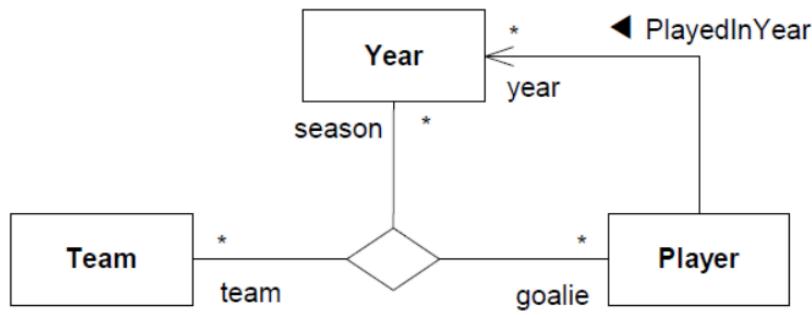


la natura bidirezionale dell’associazione è palesata dalle **frecce di navigabilità** aggiunte a entrambi i capi della linea

- Con questa specifica stiamo dicendo all’implementatore che nel progetto deve essere garantita la navigabilità in entrambi i sensi; nel caso non ci sia una specifica di navigabilità la realizzazione è deliberata all’implementatore

### 1.7.2.8 Associazioni Ternarie

Quando si hanno associazioni ternarie (o che coinvolgono più classi) si introduce il simbolo “**diamante**”



### 1.7.2.9 Associazioni: molteplicità

- La specifica UML (vista fino a ora) dichiara che la molteplicità di un'associazione è  
*the multiplicity of instances of that entity is the range of number of objects that participate in the association from the perspective of the other end (the other class)*  
 in italiano

*la molteplicità delle istanze di quell'entità è l'intervallo del numero di oggetti che partecipano all'associazione dal punto di vista dell'altro estremo (l'altra classe)*

- Tale definizione (derivata dalla specifica E/R originale) non può però applicarsi alle associazioni multiple
- Pertanto, come già visto nel corso SIT, la notazione usata in questo corso (e in altri) prevede che

*la molteplicità di un'associazione rappresenti il numero (minimo e massimo) di istanze dell'associazione a cui un'istanza dell'entità può partecipare*

### 1.7.3 Classi di Associazione

- In UML non si possono aggiungere attributi come in ER, ma si creano delle classi di associazione, che permettono di aggiungere attributi, operazioni e altre caratteristiche a una classe, che sono proprie dell'associazione
- In UML si indicano le classi di associazione con una linea tratteggiata, o utilizzando un rombo, come in figura

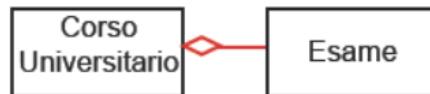


- In generale le classi di associazione sono delle classi che sono venute fuori più avanti nel progetto, perché, ad esempio, l'associazione in figura si può realizzare anche con una classe normale
- La classe di associazione aggiunge implicitamente un vincolo extra: ci può essere solo un'istanza della classe di associazione tra ogni coppia di oggetti associati

### 1.7.4 Aggregazione e Composizione

- Aggregazione:

- è un associazione che corrisponde a una relazione intuitiva Intero-Parte (“**part-of**”)
- è rappresentata da un **diamante vuoto** sull’associazione, posto vicino alla classe le cui istanze sono gli “interi”



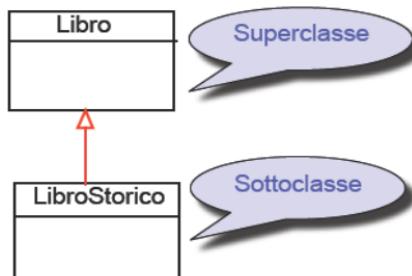
- è una relazione **binaria**
- può essere **ricorsiva**
- **Composizione:**
  - è un aggregazione che rispetta due vincoli ulteriori:
    - una parte può essere inclusa in al massimo un intero in ogni istante
    - solo l’oggetto intero può creare e distruggere le sue parti, cioè, le parti non esistono al di fuori dell’intero; questo implica, nell’implementazione, che non esiste un costruttore per le parti
  - è rappresentata da un **diamante pieno** vicino alla classe che rappresenta gli “interi”



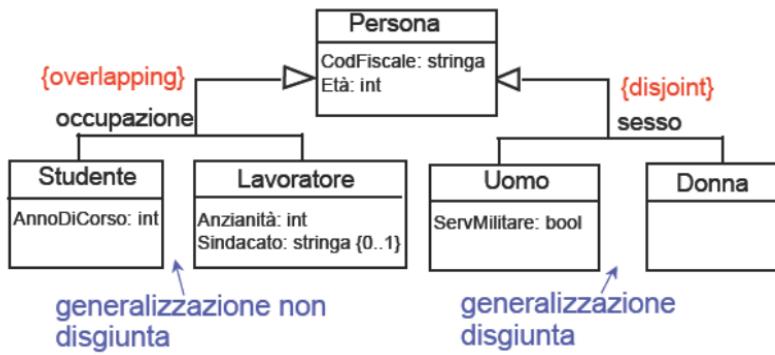
- se l’oggetto che compone viene distrutto, anche i figli vengono distrutti, anche se i figli possono essere creati/distrutti in momenti diversi dalla creazione/ distruzione dell’oggetto che compone
- può essere **ricorsiva**

### 1.7.5 Generalizzazione

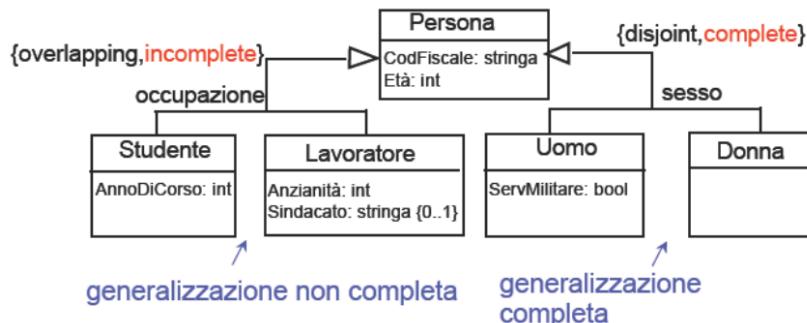
- La generalizzazione è indicata con una **freccia vuota** fra due classi dette **sottoclasse** e **superclasse**
- Il significato della generalizzazione è il seguente: ogni istanza della sottoclasse è anche istanza della superclasse



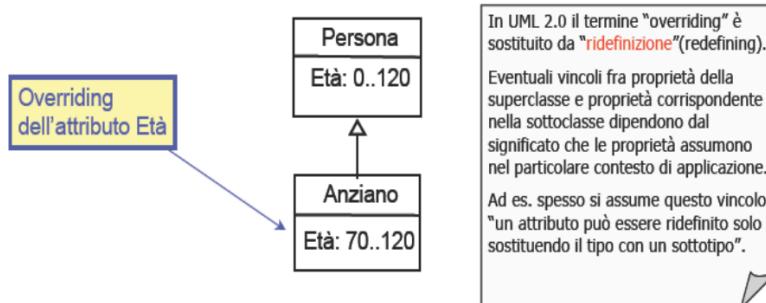
- La stessa superclasse può partecipare a diverse generalizzazioni
- Una generalizzazione può essere **disgiunta**, cioè le sottoclassi sono disgiunte (non hanno istanze in comune), o meno



- Una generalizzazione può essere **completa** (l'unione delle istanze delle sottoclassi è uguale all'insieme delle istanze della superclasse), o meno
- Attenzione:** i valori di default sono {incomplete, disjoint}



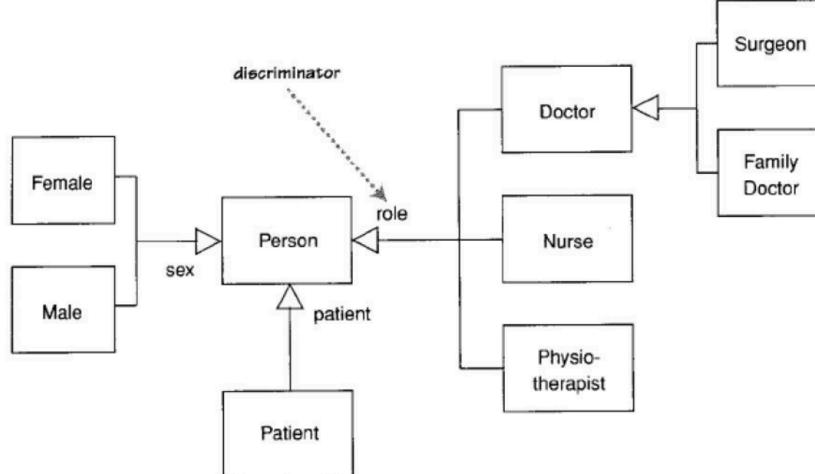
- In una generalizzazione la sottoclasse non solo può avere caratteristiche aggiuntive rispetto alla superclasse, ma può anche **sovrascrivere (overriding)** le proprietà ereditate dalla superclasse



### 1.7.6 Generalizzazione Multipla

- Con la **generalizzazione singola** un oggetto appartiene a un solo tipo, che può eventualmente ereditare dai suoi tipi padri
- Con la **generalizzazione multipla** un oggetto può essere descritto da più tipi, non necessariamente collegati dall'ereditarietà
- Si noti che la generalizzazione multipla è una cosa ben diversa dall'ereditarietà multipla
  - Ereditarietà multipla:** un tipo può avere più supertipi, ma ogni oggetto deve sempre appartenere a un suo tipo ben definito
  - Generalizzazione multipla:** un oggetto viene associato a più tipi senza che per questo debba esserne appositamente definito un altro; ad esempio, ogni istanza di persona può essere sia uomo sia studente
- Se si usa la generalizzazione multipla, ci si deve assicurare di rendere chiare le combinazioni "legali"
- Per questo fatto, UML pone ogni relazione di generalizzazione in un **insieme di generalizzazione**

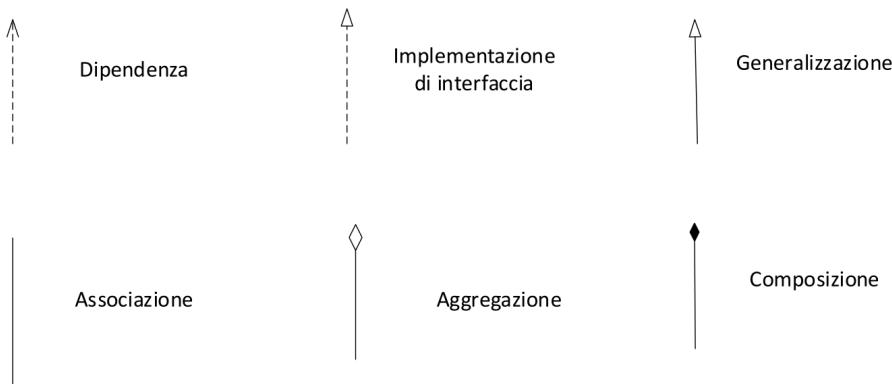
- Sul diagramma delle classi, la freccia che indica una generalizzazione va etichettata con il nome del rispettivo insieme
- La generalizzazione singola corrisponde all'uso di un singolo anonimo insieme di generalizzazioni
- Gli insiemi di generalizzazione sono disgiunti per definizione
  - Ogni istanza del supertipo può essere istanza di uno dei sottotipi all'interno di quel sottoinsieme



- Questa figura ci dice che ci sono tre modi diversi di vedere una persona:
  - dal punto di vista del sesso
  - dal punto di vista dell'essere paziente o meno
  - dal punto di vista del ruolo

### 1.7.6.1 Relazione tra classi: sintassi

- Attenzione all'uso corretto delle frecce: UML è un linguaggio (anche se grafico) e scambiare una freccia per un'altra è un errore non da poco



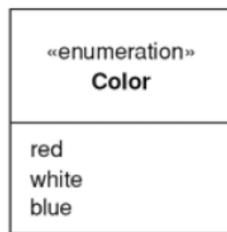
### 1.7.7 Classi Astratte

- Una **classe astratta** è una classe che non può essere direttamente istanziata: per farlo bisogna prima crearne una sottoclasse concreta
- Tipicamente, una classe astratta ha una o più operazioni astratte
- Un'operazione astratta non ha implementazione
  - è costituita dalla sola dichiarazione, resa pubblica affinché le classi client possano usufruirne
- Il modo più diffuso di indicare una classe o un'operazione astratta in UML è **scrivere il nome in corsivo**

- Si possono anche rendere astratte le proprietà indicandole direttamente come tali o rendendo astratti i metodi d'accesso
- A cosa servono?
  - servono come superclassi comuni per un insieme di sottoclassi concrete
  - queste sottoclassi, in virtù del subtyping, sono in qualche misura compatibili e intercambiabili fra di loro
  - infatti sono tutte sostituibili con la superclasse
    - per il principio di sostituzione di Liskov, tutte le istanze delle sottoclassi sono anche istanze della superclasse, quindi possono attuare qualsiasi comportamento descritto dalla superclasse, inclusi i comportamenti astratti

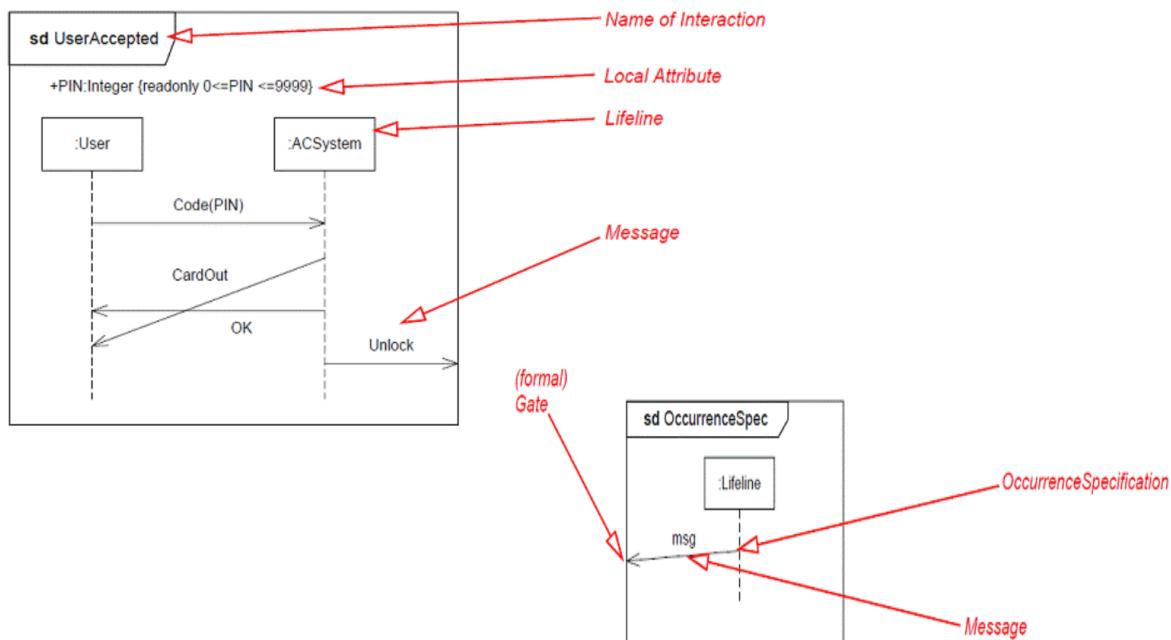
### 1.7.8 Enumerazioni

- Le enumerazioni sono usate per mostrare un insieme di valori prefissati (quelli scritti all'interno) che non hanno altre proprietà oltre al loro valore simbolico
- Sono rappresentate con una classe marcata dalla parola chiave «enumeration»



## 1.8 Diagramma di sequenza

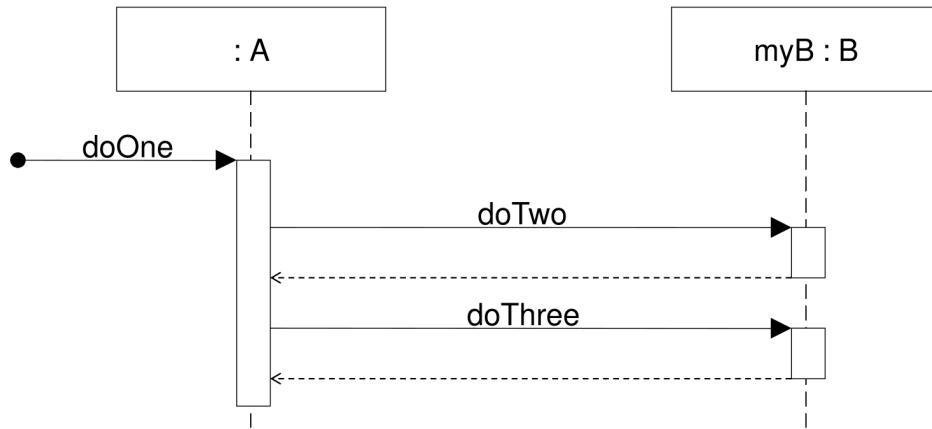
- Diagramma che illustra le interazioni tra le classi / entità disponendole lungo una sequenza temporale
- In particolare mostra i soggetti (chiamati tecnicamente **Lifeline**) che partecipano all'interazione e la sequenza dei messaggi scambiati
- In ascissa troviamo i diversi soggetti (non necessariamente in ordine di esecuzione), mentre in ordinata abbiamo la scala dei tempi sviluppata verso il basso



- Le linee della *Lifeline* partono da un quadrato, un'entità definita con sintassi UML. In questo caso i “:” indicano un'istanza della classe
- Tra la classe `User` e la classe `ACSystem` ci sarà sicuramente un'associazione

- La figura ci dice inoltre che i due messaggi che vengono inviati dopo la ricezione di PIN possono venire inviati in momenti diversi

### 1.8.1 Esempio



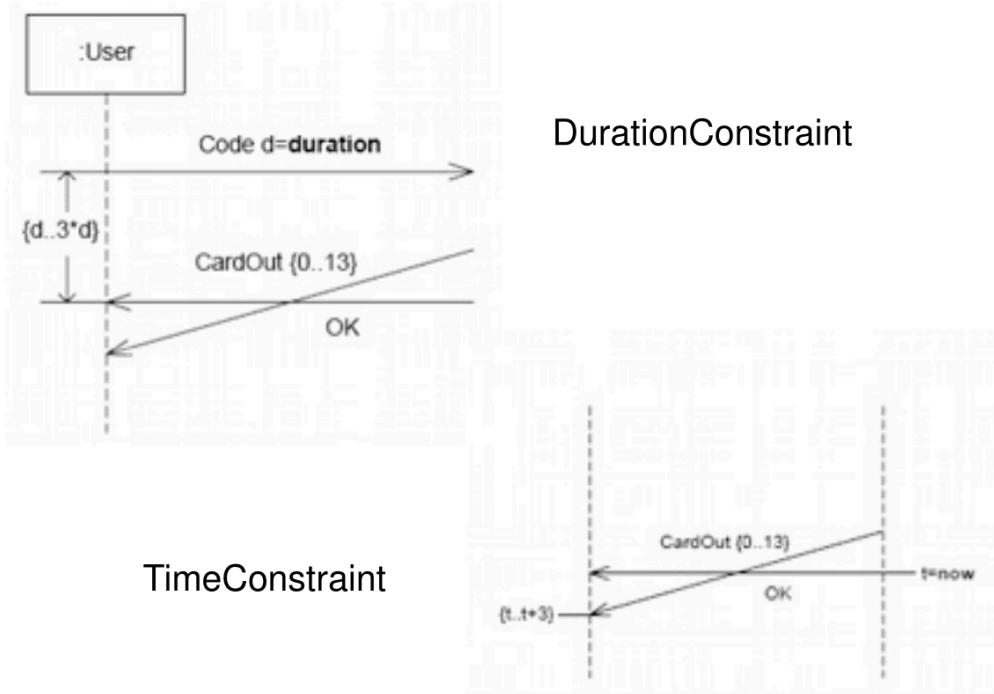
- La freccia con linea continua e piena indica un messaggio sincrono, il chiamante rimane in attesa della risposta del chiamato
- La freccia con linea tratteggiata e vuota è il messaggio che il chiamato invia al chiamante

### 1.8.2 Lifeline

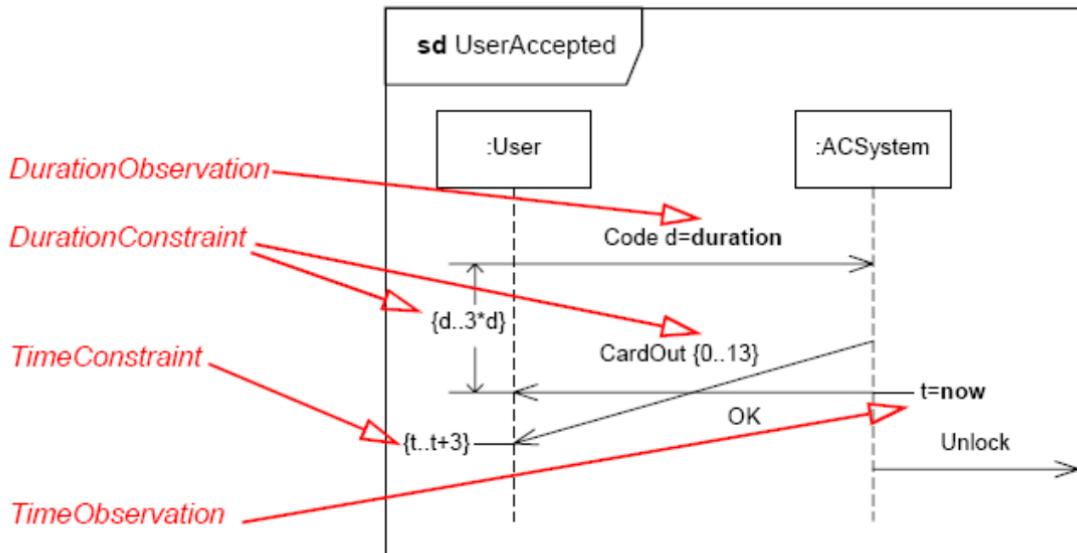
- In un diagramma di sequenza, i partecipanti solitamente sono istanze di classi UML caratterizzate da un nome
- La loro vita è rappresentata da una *Lifeline*, cioè una linea **tratteggiata verticale ed etichettata**, in modo che sia possibile comprendere a quale componente del sistema si riferisce
- In alcuni casi il partecipante non è un'entità semplice, ma composta
  - è possibile modellare la comunicazione fra più sottosistemi, assegnando una Lifeline ad ognuno di essi
- L'ordine in cui le OccurrenceSpecification (cioè l'invio e la ricezione di eventi) avvengono lungo la Lifeline **rappresenta esattamente l'ordine in cui tali eventi si devono verificare**
- La distanza (in termini grafici) tra due eventi non ha rilevanza dal punto di vista semantico
- Dal punto di vista notazionale, una Lifeline è rappresentata da un rettangolo che costituisce la “testa” seguito da una linea verticale che rappresenta il tempo di vita del partecipante
- È interessante notare che nella sezione della notazione, viene indicato espressamente che il “rettangolino” che viene apposto sulla Lifeline rappresenta l'attivazione di un metodo alla ricezione di un messaggio

#### 1.8.2.1 Vincoli Temporali

- Per modellare sistemi real-time, o comunque qualsiasi altra tipologia di sistema in cui la temporizzazione è critica, è necessario specificare un istante in cui un messaggio deve essere inviato, oppure quanto tempo deve intercorrere fra un'interazione e un'altra
- Grazie, rispettivamente, a Time Constraint e Duration Constraint è possibile definire questo genere di vincoli



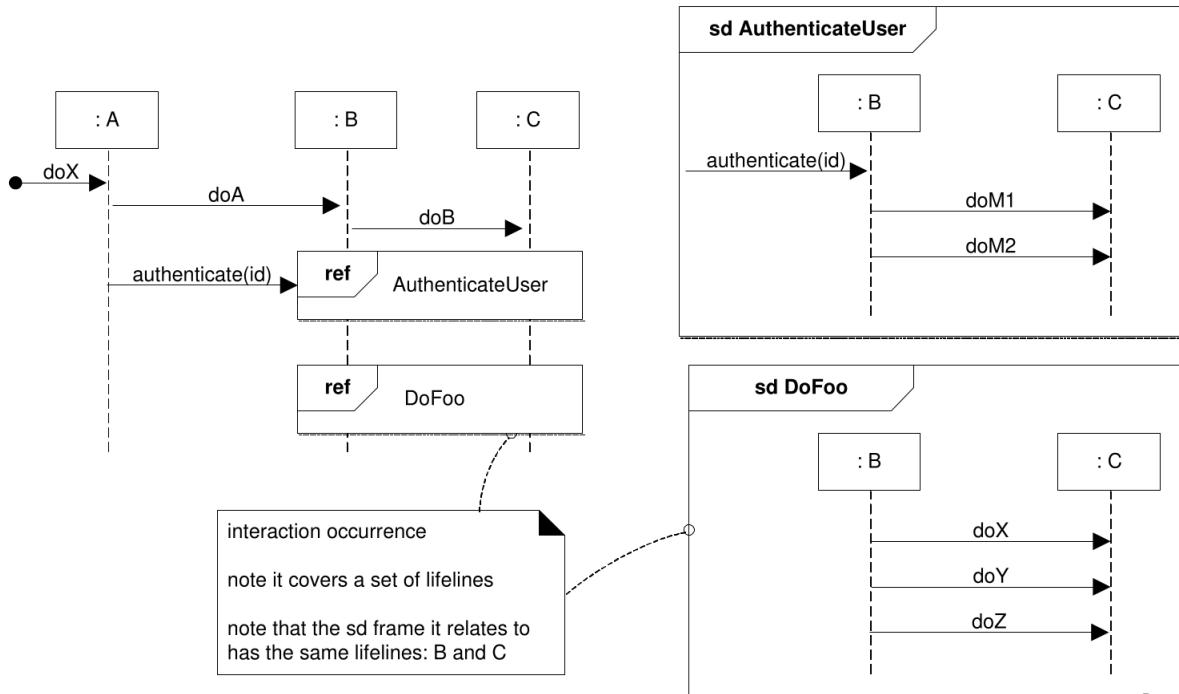
N.B. Noi non utilizzeremo mai vincoli temporali



### 1.8.3 Riferimento ad altri Diagrammi

- Spesso i diagrammi di sequenza possono assumere una certa complessità
  - necessità di poter definire comportamenti più articolati come composizione di nuclei di interazione più semplici
- Oppure, se una sequenza di eventi ricorre spesso, potrebbe essere utile definirla una volta e richiamarla dove necessario
- Per questa ragione, UML permette di inserire **riferimenti ad altri diagrammi** e passare loro degli argomenti
- Ovviamente ha senso sfruttare quest'ultima opzione solo se il diagramma accetta dei parametri sui quali calibrare l'evoluzione del sistema
- Questi riferimenti prendono il nome di **InteractionUse**
- I punti di connessione tra i due diagrammi prendono il nome di **Gate**

- Un Gate rappresenta un **punto di interconnessione** che mette in relazione un messaggio al di fuori del frammento di interazione con uno all'interno del frammento



### 1.8.3.1 Messaggio

- Un messaggio rappresenta un'interazione realizzata come comunicazione fra Lifeline
- Questa interazione può consistere nella creazione o distruzione di un'istanza, nell'invocazione di un'operazione, o nella emissione di un segnale
- UML permette di rappresentare tipi differenti di messaggi

#### 1.8.3.1.1 Tipi di Messaggio

- Se sono specificati mittente e destinatario è un **complete message**
  - la semantica è rappresentata quindi dall'occorrenza della coppia di eventi **<sendEvent, receiveEvent>**
- Se il destinatario non è stato specificato è un **lost message**
  - in questo caso è noto solo l'evento di invio del messaggio
- Se il mittente non è stato specificato è un **found message**
  - in questo caso è noto solo l'evento di ricezione del messaggio
- Nel caso non sia noto né il destinatario né il mittente è un **unknown message**

NODE TYPE	NOTATION	REFERENCE
Message		Messages come in different variants depending on what kind of Message they convey. Here we show an asynchronous message, a call and a reply. These are all <i>complete</i> messages.
Lost Message		Lost messages are messages with known sender, but the reception of the message does not happen.
Found Message		Found messages are messages with known receiver, but the sending of the message is not described within the specification.
GeneralOrdering		

- Attenzione alle frecce che usate nei messaggi, hanno significati diversi:

- riga continua freccia piena: indica un messaggio (**call**) **sincrono** in cui il mittente **aspetta** il completamento dell'esecuzione del destinatario prima di continuare la sua esecuzione



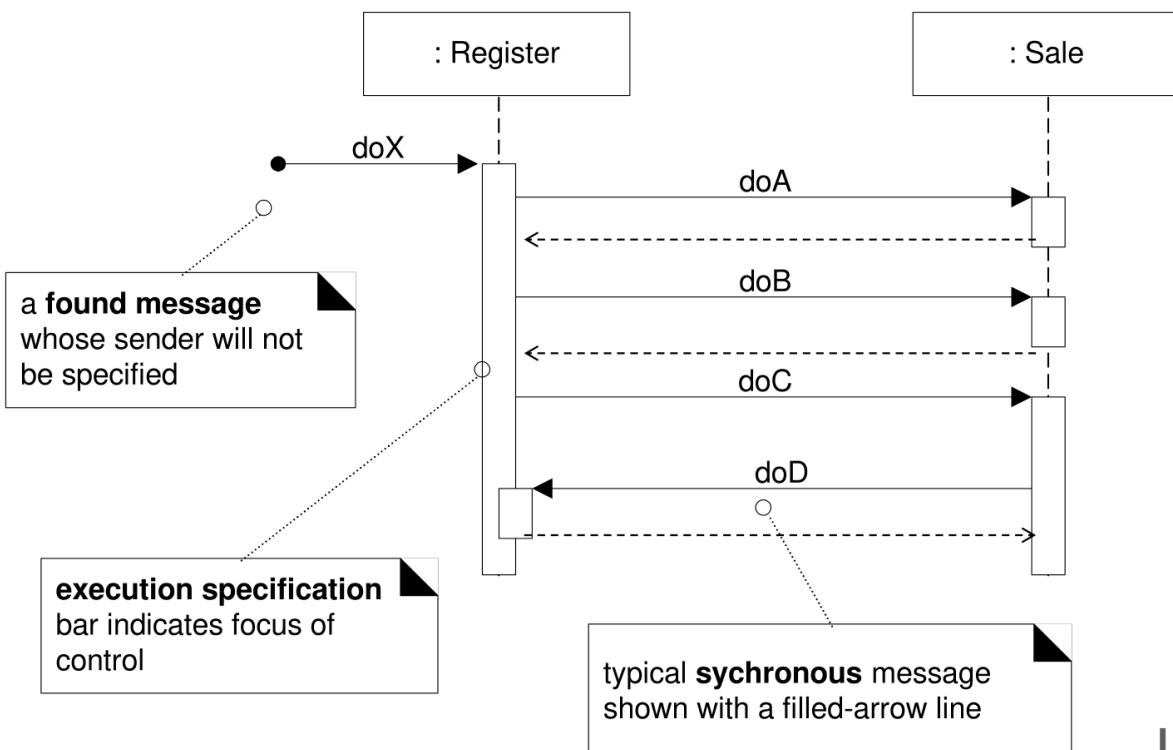
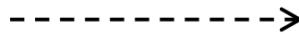
– Necessita di un messaggio di ritorno per sbloccare l'esecuzione del mittente

- riga continua freccia vuota: indica un messaggio **asincrono** in cui il mittente **non aspetta** il completamento dell'esecuzione del destinatario ma continua la sua esecuzione

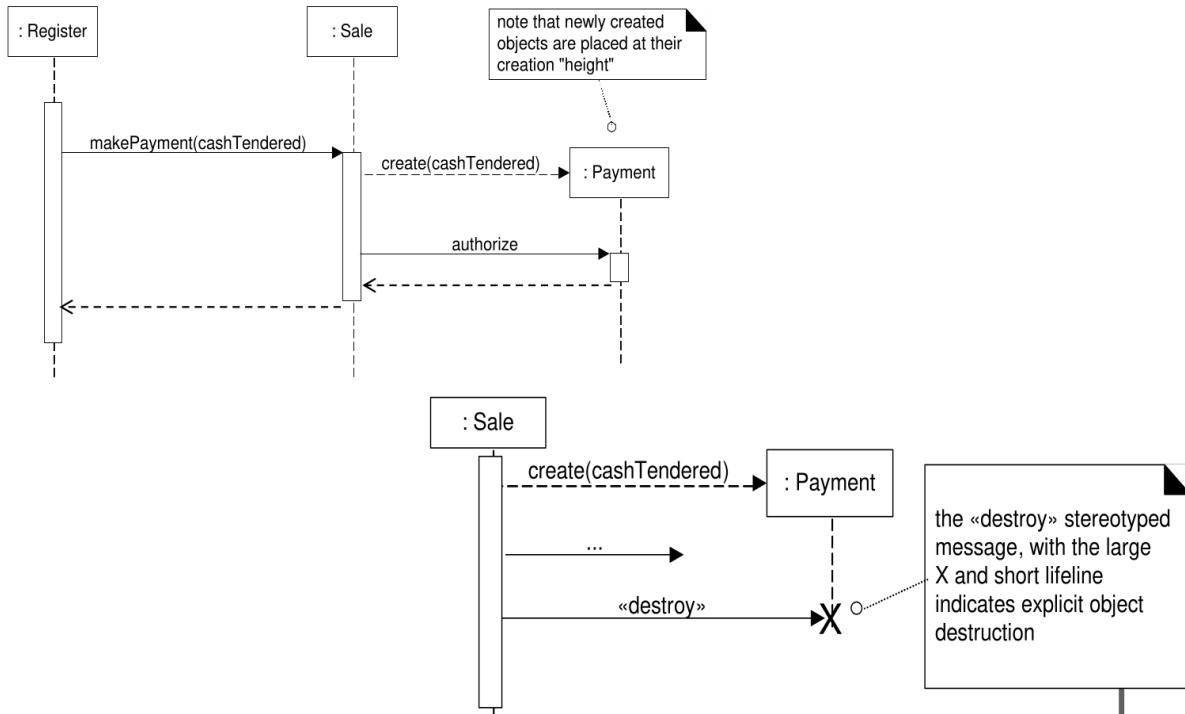


– Il valore di ritorno potrebbe o meno essere necessario, dipende dalla semantica

- riga tratteggiata freccia vuota: indica il ritorno di un messaggio



- Non specifichiamo chi è che ha inviato il messaggio, in questo diagramma non ci interessa
- Register manda il messaggio `doA` all'oggetto Sale e aspetta la sua terminazione
- Poi manda il messaggio `doB` all'oggetto Sale e aspetta la sua terminazione
- Infine invia il messaggio `doC`, e come conseguenza Sale invia `doD` alla classe Register



- **authorize**
  - Per realizzare `makePayament(cashTandered)` la classe `Sale` delega la classe `Payment` (creando un'istanza della classe stessa), che invoca il metodo `create(casheTandered)`; dopodiché invia il messaggio sincrono `authorize`, e restituisce il controllo a `Sale`
  - A questo punto `Sale` può notificare l'avvenuto pagamento
- **destroy**
  - con il messaggio `destroy` viene distrutta l'istanza di quella classe
  - siccome siamo in *modellazione*, non dobbiamo soffermarci sul fatto che il nostro linguaggio di programmazione possa o meno poter distruggere un'istanza di una classe

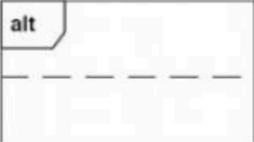
#### 1.8.4 Combined Fragment

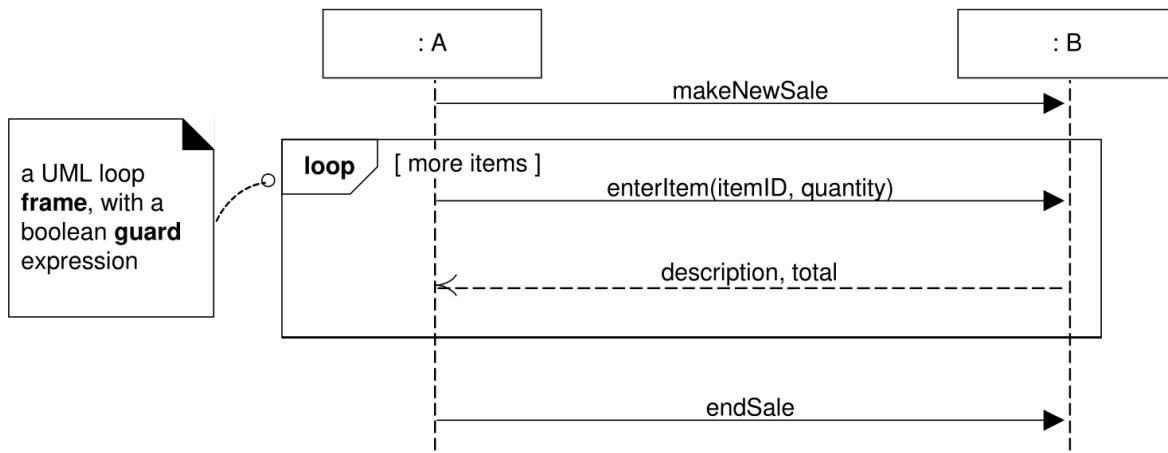
- La specifica di UML permette di esprimere comportamenti più complessi rispetto al singolo scambio di messaggi
- È possibile rappresentare l'esecuzione atomica di una serie di interazioni, oppure che un messaggio debba essere inviato solo in determinate condizioni
- A tale scopo UML mette a disposizione i *Combined Fragment*, cioè contenitori atti a delimitare un'area d'interesse nel diagramma
- Servono per spiegare che una certa catena di eventi, racchiusa in uno o più operandi, si verificherà in base alla semantica dell'operatore associato
- Ogni fragment ha un operatore e una (eventuale) guardia

##### 1.8.4.1 Tipi

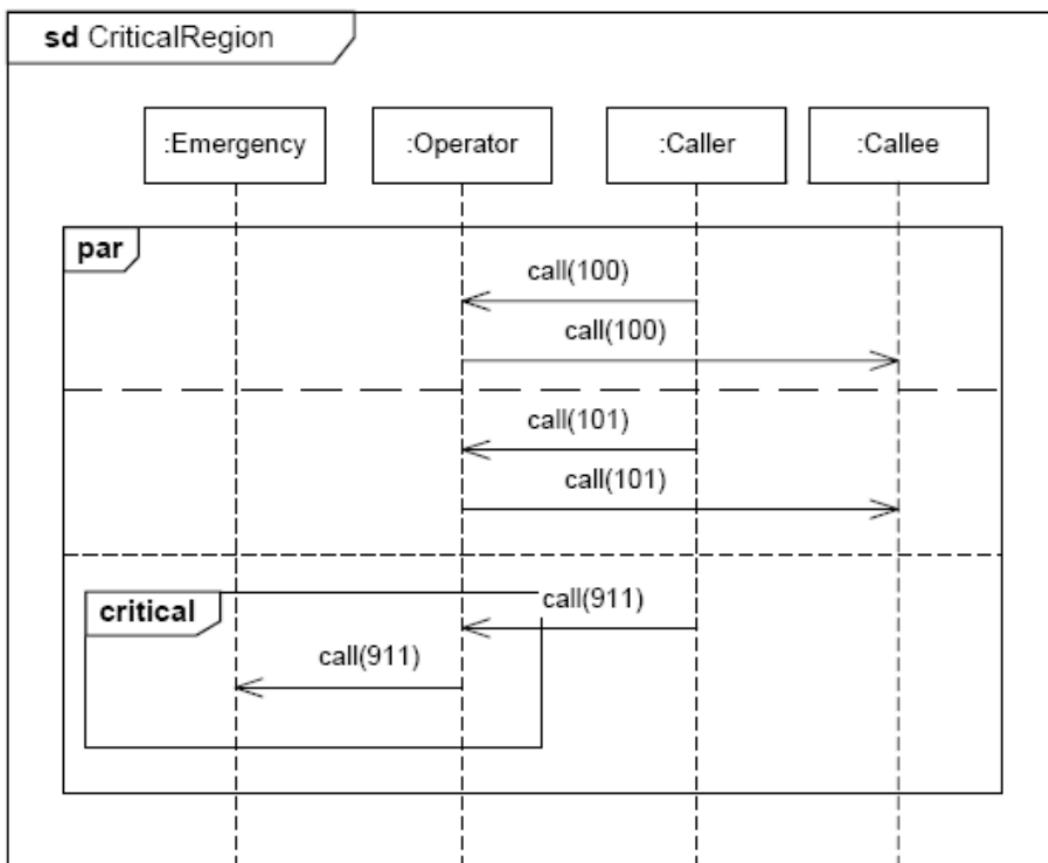
- **Loop:** specifica che quello che è racchiuso nell'operando sarà eseguito ciclicamente finché la guardia sarà verificata

- **Alternatives** (alt): indica che sarà eseguito il contenuto di uno solo degli operandi, quello la cui guardia risulta verificata
- **Optional** (opt): indica che l'esecuzione del contenuto dell'operando sarà eseguita solo se la guardia è verificata
- **Break** (break): ha la stessa semantica di opt, con la differenza che in seguito l'interazione sarà terminata
- **Critical**: specifica un blocco di esecuzione atomico (non interrompibile)
- **Parallel** (par): specifica che il contenuto del primo operando può essere eseguito in parallelo a quello del secondo
- **Weak Sequencing** (seq): specifica che il risultato complessivo può essere una qualsiasi combinazione delle interazioni contenute negli operandi, purché:
  - l'ordinamento stabilito in ciascun operando sia mantenuto nel complesso
  - eventi che riguardano gli stessi destinatari devono rispettare anche l'ordine degli operandi, cioè i messaggi del primo operando hanno precedenza su quelli del secondo
  - eventi che riguardano destinatari differenti non hanno vincoli di precedenza vicendevole
- **StrictSequencing** (strict): indica che il contenuto deve essere eseguito nell'ordine in cui è specificato, anche rispetto agli operandi
- **Ignore**: indica che alcuni messaggi, importanti ai fini del funzionamento del sistema, non sono stati rappresentati, perché non utili ai fini della comprensione dell'interazione
- **Consider**: è complementare ad ignore
- **Negative** (neg): racchiude una sequenza di eventi che non deve mai verificarsi
- **Assertion** (assert): racchiude quella che è considerata l'unica sequenza di eventi valida
  - di solito è associata all'utilizzo di uno State Invariant come rinforzo

<i>NODE TYPE</i>	<i>NOTATION</i>	<i>REFERENCE</i>
Frame		The notation shows a rectangular frame around the diagram with a name in a compartment in the upper left corner.
CombinedFragment		



Fintanto che ci sono degli item, io faccio enter item e quello mi restituisce description total. Quando non esistono più item invio il messaggio endSale.



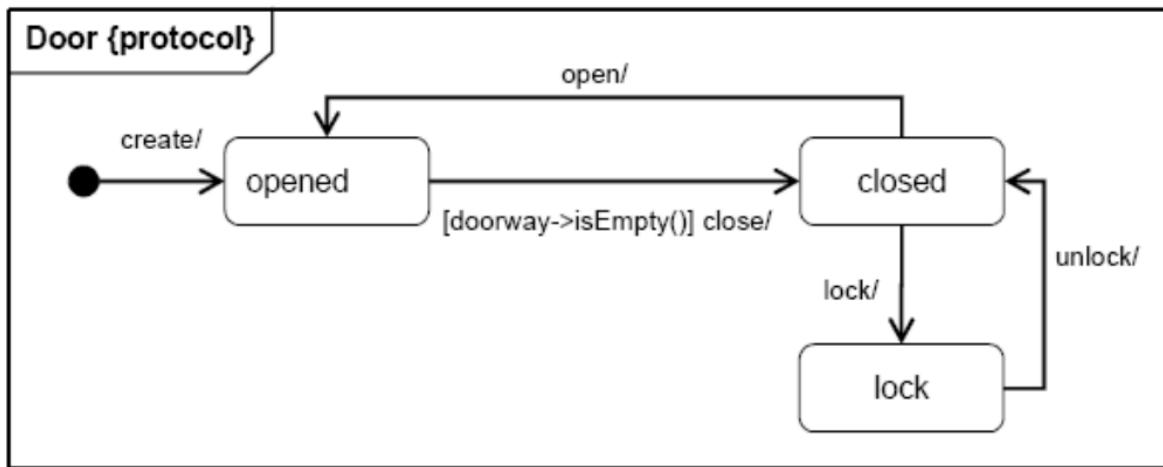
Fintanto che il chiamante invoca certi numeri, ridireziona quei numeri al chiamato, se viene chiamato il 991 quella è una chiamata critica che non può essere interrotta.

## 1.9 Diagrammi di Stato

- I **diagrammi di stato** modellano la dipendenza che esiste tra lo stato di una classe / entità (cioè il valore corrente delle sue proprietà) e i messaggi e/o eventi che questo riceve in ingresso
- Specifica il ciclo di vita di una classe / entità, definendo le regole che lo governano
- Quando una classe / entità si trova in un certo stato può essere interessata a determinati eventi (e non ad altri)
- Come risultato di un evento una classe / entità può passare a un nuovo stato (transizione)

- Uno **stato** è una condizione o situazione nella vita di un oggetto in cui esso soddisfa una condizione, esegue un'attività o aspetta un evento
- Un **evento** è la specifica di un'occorrenza che ha una collocazione nel tempo e nello spazio
- Una **transizione** è il passaggio da uno stato a un altro in risposta ad un evento

### 1.9.1.1 Esempio

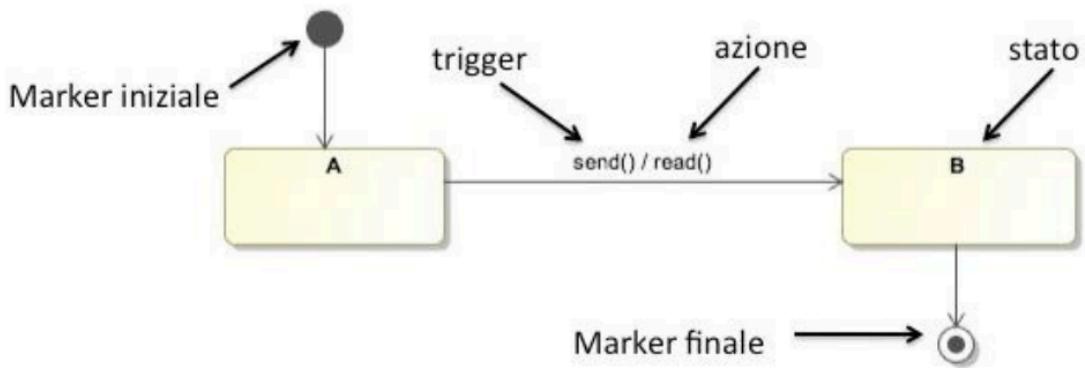


- Quando la porta è creata, la porta è aperta.
- Fintanto che la porta è aperta, è interessata a un unico messaggio: **close**; tutti gli altri messaggi vengono ignorati
- Quando riceve il messaggio **close**, e, parentesi quadra che indica la **guardia**, **doorway -> isEmpty()**, cioè non c'è nessuno sulla soglia, la porta transita dallo stato **open** allo stato **closed**
- Dunque la porta si potrebbe trovare in un stato **closed**, ed è interessata a due messaggi:
  - il messaggio **open**, che la riporta nello stato **open**
  - il messaggio **lock**, che la porta nello stato **lock**
- Quando si trova nello stato **lock**, la porta può essere interessata solo all'evento **unlock**, che la riporta nello stato **closed**

### 1.9.1.2 Concetti

- I concetti più importanti di un diagramma di stato sono:
  - gli **stati**, indicati con rettangoli con angoli arrotondati
  - le **transizioni** tra stati, indicate attraverso frecce
  - gli **eventi** che causano transizioni, la tipologia più comune è rappresentata dalla ricezione di un messaggio, che si indica semplicemente scrivendo il nome del messaggio con relativi argomenti vicino alla freccia; quindi l'oggetto, l'istanza, riceve un messaggio, e ricevendo quel messaggio effettua la transizione
  - i **marker** di inizio e fine rappresentati rispettivamente da un cerchio nero con una freccia che punta allo stato iniziale e come un cerchio nero racchiuso da un anello sottile
  - le **azioni** che una entità è in grado di eseguire in risposta alla ricezione di un evento; cioè, se ricevo quel evento, oltre a fare la transizione, posso fare anche altre cose
  - il **vertice** che rappresenta l'astrazione di nodo nel diagramma e può essere la sorgente o la destinazione di una o più transizioni; rappresento un sotto-diagramma

### 1.9.1.3 Esempio



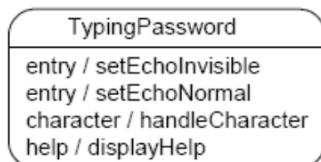
- Marker iniziale, stato iniziale, a seguito di un messaggio ricevuto, effetto quest'azione e vado in transizione verso un nuovo stato
- Da quello stato, quando posso, quando ricevo la distruzione, vado al marker finale

### 1.9.1.4 Stato

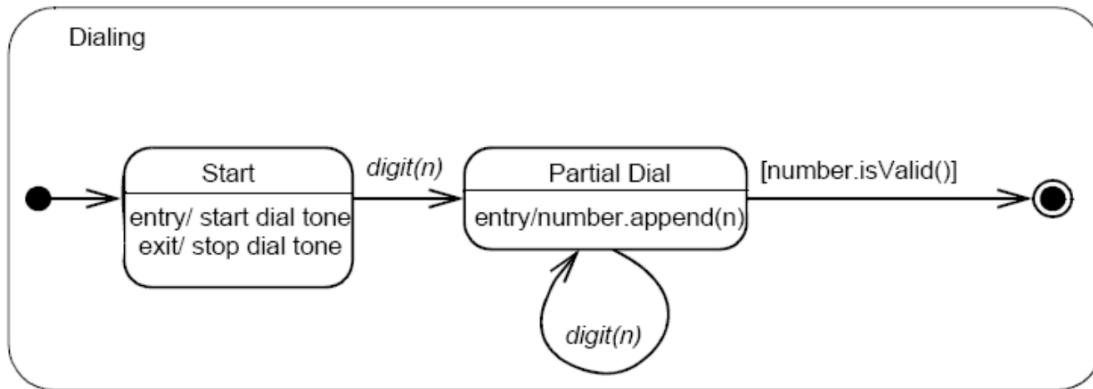
- Uno stato modella una situazione durante la quale vale una condizione invariante (solitamente implicita)
- L'invariante può rappresentare una situazione statica, come un oggetto in attesa che si verifichi un evento esterno
- Tuttavia, può anche modellare condizioni dinamiche, come il processo di esecuzione di un comportamento (cioè, l'elemento del modello in esame entra nello stato quando il comportamento inizia e lo lascia non appena il comportamento è completato)
- Stati possibili:
  - [Simple state](#)
  - [Composite state](#)
  - [Submachine state](#)
- Composite state:
  - può contenere una regione oppure è decomposto in due o più regioni ortogonali
  - ogni regione ha il suo insieme di sotto-vertici mutuamente esclusivi e disgiunti e il proprio insieme di transizioni
  - ogni stato appartenente ad una regione è chiamato substate
  - la transizione verso il marker finale all'interno di una regione rappresenta il completamento del comportamento di quella regione
  - quando tutte le regioni ortogonali hanno completato il loro comportamento questo rappresenta il completamento del comportamento dell'intero stato e fa scattare il trigger dell'evento di completamento
- Simple state:
  - è uno stato che non ha sottostati
- Submachine state:
  - specifica l'inserimento di un diagramma di una sotto-parte del sistema e permette la fattorizzazione di comportamenti comuni e il riuso dei medesimi
  - è semanticamente equivalente a un composite state, quindi le regioni del submachinestate sono come le regioni del composite state
  - le azioni sono definite come parte dello stato
- Uno stato può essere ridefinito:
  - uno stato semplice può essere ridefinito diventando uno stato composito

- uno stato composito può essere ridefinito aggiungendo regioni, vertici, stati, transizioni e azioni

### 1.9.1.5 Esempio



Simple state con azioni

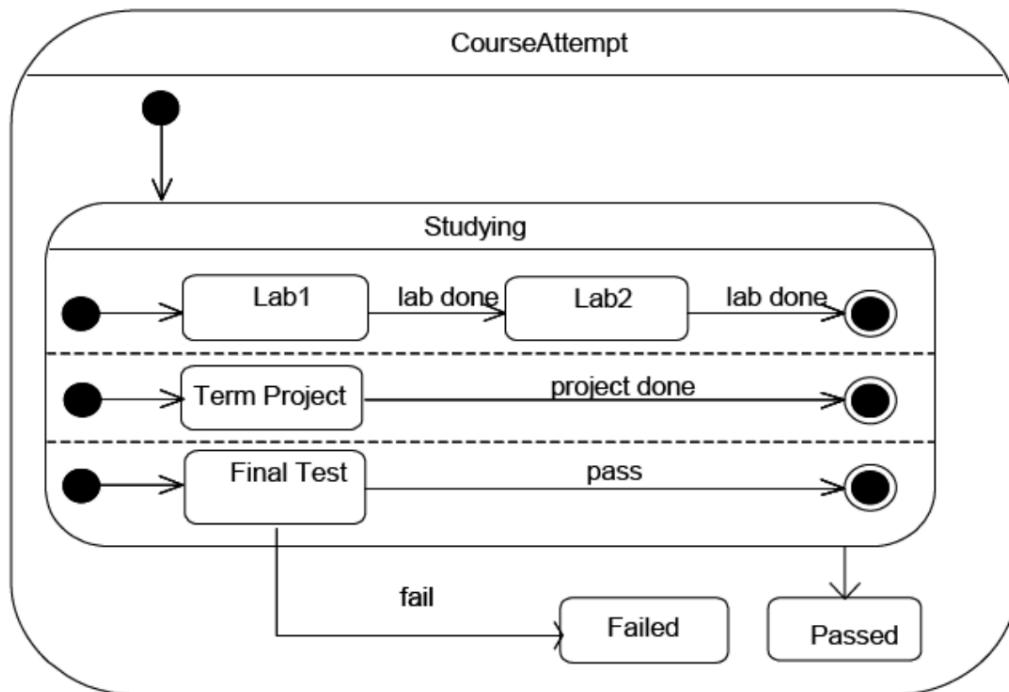


Composite state

Questo è uno stato composito, nel senso che quello stato composito può essere inserito all'interno di un altro diagramma di stato. E quindi rappresenta la fattorizzazione che abbiamo visto, è analoga a quello che abbiamo visto nel diagramma dell'attività.

### 1.9.1.6 Esempio

Composite state con regioni ortogonali

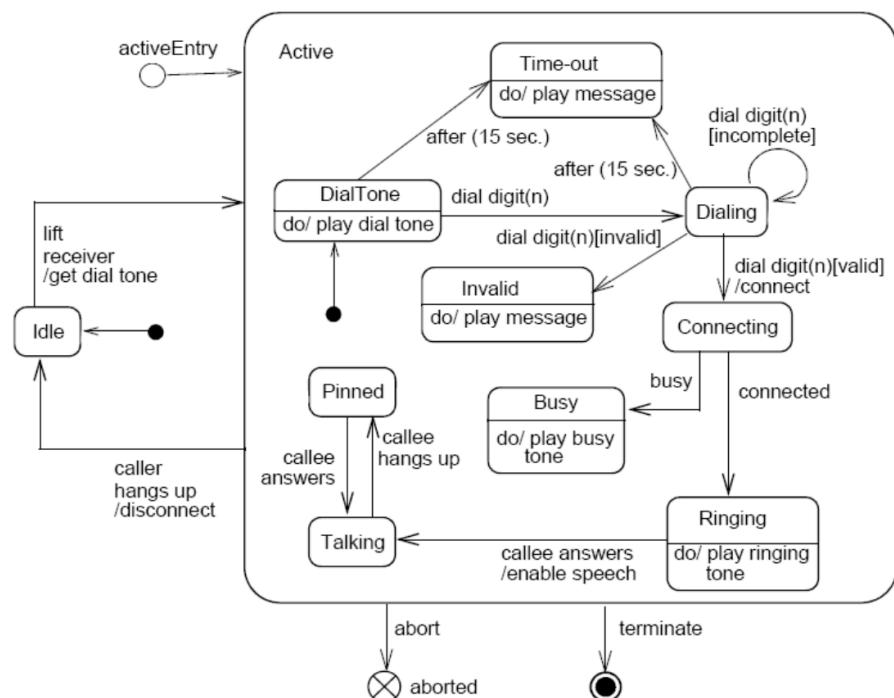


- Una Regione è una parte ortogonale di uno stato composto o di un state machine
- Questo è uno stato composito con regioni ortogonali che rappresenta il diagramma di stato per il superamento di un esame da parte dello studente

- In questo stato di piano ci sono le tre regioni che sono eseguite “concorrentemente”, possono essere eseguite in paralleli, non sto indicando una sequenzialità
- Possono iniziare contemporaneamente ma non finiscono per forza nello stesso istante
- Il diagramma dice che il design prevede lo svolgimento della Lab1, dal quale si esce quando lab done
- A quel punto posso fare Lab2, quando ho fatto Lab2, lab done, ho terminato quella regione
- Quindi si termina quella regione quando ho terminato Lab1 e Lab2
- Nella seconda regione si esce quando il progetto è terminato
- In fine c'è il final test; dal final test si può uscire con due messaggi
  - fail che provoca automaticamente la segnalazione dell'esame come fallito, Failed
  - pass, allora termine quella regione, e l'esame viene segnato come passato, Passed

### 1.9.1.7 Esempio

Submachine state



### 1.9.1.8 Pseudostato

- Gli pseudostati vengono generalmente utilizzati per collegare più transizioni in percorsi di transizioni di stato più complessi
- Ad esempio, combinando una transizione che entra in uno pseudostato fork con un insieme di transizioni che escono dallo pseudostato fork, otteniamo una transizione composta che porta a un insieme di stati ortogonali

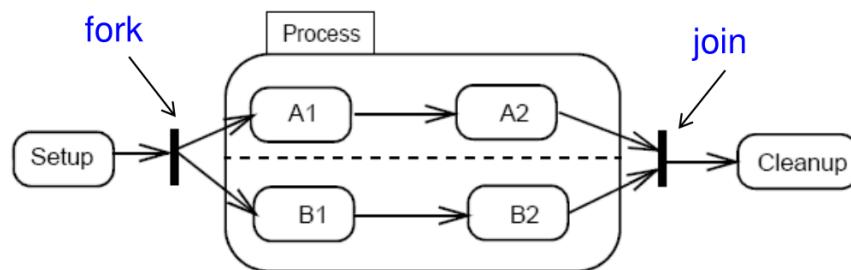
### 1.9.1.9 Tipi di Pseudostati

- **Initial:** vertice di default che è la sorgente della singola transizione verso lo stato di default di un composite state - Ci può essere al massimo un vertice initial e la transizione uscente da questo vertice non può avere trigger o guardie
- **deepHistory:** la più recente configurazione attiva del composite state che contiene direttamente questo pseudostato

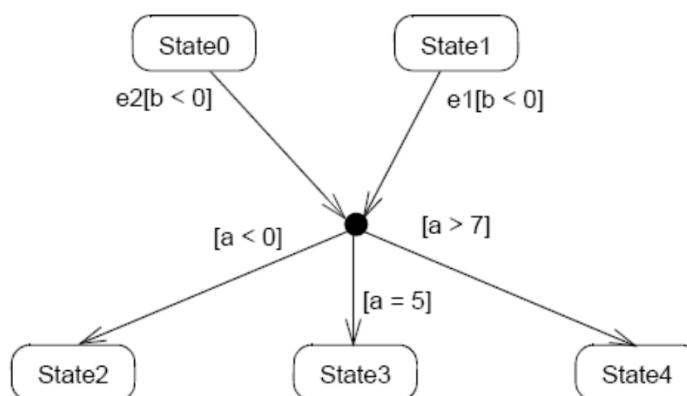
- **shallowHistory**: il più recente substate attivo di un composite state initial deepHistory shallowHistory



- **Join**: permette di eseguire il merge di diverse transizioni provenienti da diverse sorgenti appartenenti a differenti regioni ortogonalali
  - ▶ Le transizioni entranti in questo vertice non possono avere guardie e trigger
- **Fork**: permette di separare una transizione entrante in due o più transizioni che terminano in vertici di regioni ortogonalali
  - ▶ Le transizioni uscenti da questo vertice non possono avere guardie

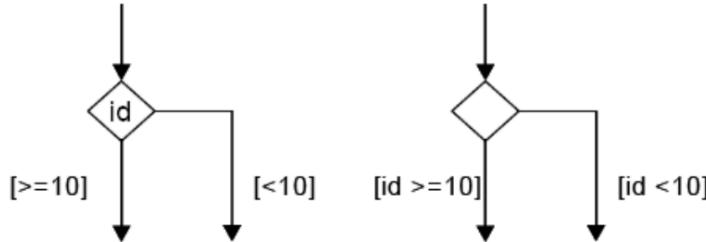


- **Junction**: è un vertice privo di semantica che viene usato per “incatenare” insieme transizioni multiple
  - ▶ È usato per costruire percorsi di transizione composti tra stati
  - ▶ Una junction può essere usata per far convergere transizioni multiple entranti in una singola transizione uscente che rappresenta un percorso condiviso di transizione
  - ▶ Allo stesso modo una junction può essere usata anche per suddividere una transizione entrante in più transizioni uscenti con differenti guardie
  - ▶ Permette di realizzare un branch condizionale statico, quindi quando esco da un certo stato, so già in quale delle biforcati finirò
  - ▶ Esiste una guardia predefinita chiamata “else” che viene attivata quando tutte le guardie sono false

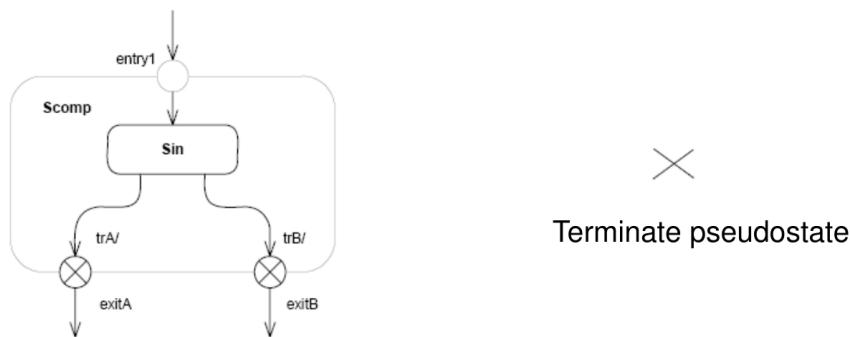


- **Choice**: è un tipo di vertice che quando viene raggiunto causa la valutazione dinamica delle guardie dei trigger delle transizioni uscenti
  - ▶ Le guardie sono quindi tipicamente scritte sotto forma di “funzione” che viene valutata al momento del raggiungimento del vertice choice
  - ▶ Permette di realizzare un branch condizionale dinamico separando le transizioni uscenti e creando diversi percorsi di uscita

- Se più di una guardia è verificata viene scelto il percorso in modo arbitrario
- Se nessuna è verificata il modello viene considerato “ill-formed”
- È quindi caldamente consigliato definire sempre un percorso di uscita di tipo “else”



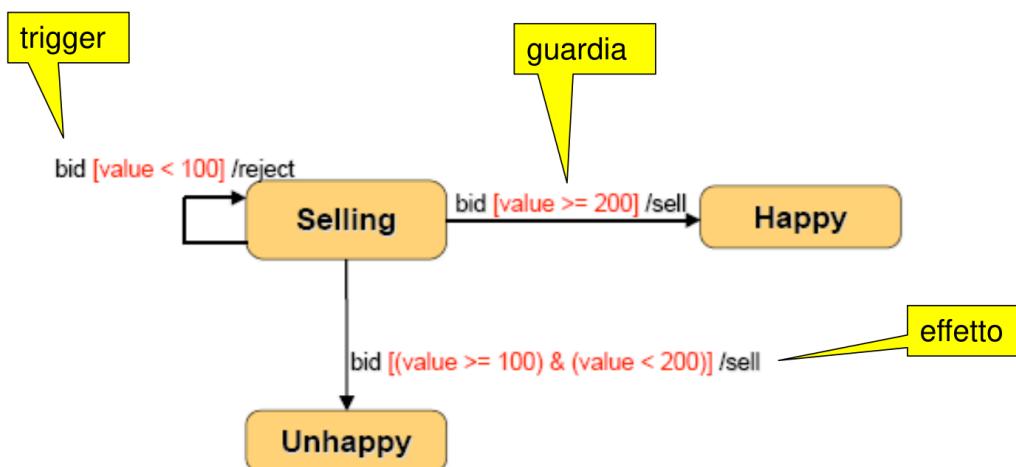
- La differenza tra *Junciton* e *Choice* è che, siccome nelle transizioni io posso eseguire delle azioni, la biforcazione di junction non può dipendere dalle azioni, mentre la biforcazione di choice può dipendere dall'esito di quelle azioni.
- **Entry point:** è l'ingresso di uno state machine o di un composite state
- **Exit point:** è l'uscita da uno state machine o da uno stato composito
  - Entrare in questo vertice significa attivare la transizione che ha questo vertice come sorgente
- **Terminate:** entrare in questo vertice implica che l'esecuzione di questo state machine è terminata
  - Lo state machine non uscirà da nessuno stato né verranno invocate altre azioni a parte quelle associate con la transizione che porta allo stato terminate



### 1.9.1.10 Transizione

- Una transizione è una relazione diretta tra un vertice di origine e un vertice di destinazione, un arco direzionale
- Può essere parte di una transizione composta, che porta la macchina a stati da una configurazione di stato a un'altra, rappresentando la risposta completa della macchina a stati al verificarsi di un evento di un tipo particolare; cioè, c'è un evento, definisco qual è la risposta della macchina a quell'evento, la risposta dell'istanza a quell'evento
- Analizzando la specifica UML è possibile vedere che tra le proprietà di una transizione compaiono diversi concetti molto rilevanti tra cui:
  - **trigger:** cioè i tipi di evento che possono innescare la transizione
  - **guardia:** cioè un vincolo che fornisce un controllo fine sull'innesto della transizione

- La guardia è valutata quando una occorrenza dell'evento è consegnata allo stato
- Se la guardia risulta verificata allora la transizione può essere abilitata altrimenti questa viene disabilitata
- Il problema delle guardie è che, in teoria, la guardia potrebbe essere rappresentata dalla esecuzione di operazione; se la guardia (l'operazione) ha degli effetti collaterali (*side effect*), potrebbe succedere il caso in cui la prima ricezione di un messaggio non blocca la transizione, mentre la seconda ricezione dello stesso messaggio blocca la transizione
- Quindi le guardie dovrebbero essere pure espressioni senza *side effect*, i quali sono assolutamente sconsigliati nella specifica
- **effetto:** specifica un comportamento opzionale (cioè un'azione) che deve essere eseguito quando la transizione scatta trigger guardia effetto (l'effetto viene considerato da choose e non da junction)



- Sono in stato Selling, ricevo bid, scatta la guardia; se il valore è minore di 100 eseguo l'operazione reject, e rimango nello stato selling
- Se il valore è maggiore o uguale a 100 e minore di 200 alla ricezione del messaggio bid, eseguo l'operazione sell e passo in un stato Unhappy
- Se il valore è maggiore o uguale a 200 all'atto della ricezione del messaggio bid, eseguo l'operazione sell e passo nello stato Happy

### 1.9.1.11 Tipi di Eventi

- **Evento di chiamata:** ricezione di un messaggio che richiede l'esecuzione di una operazione
- **Evento di cambiamento:** si verifica quando una condizione passa da “falsa” a “vera”
  - Tale evento si specifica scrivendo <<when>> seguito da un'espressione, racchiusa tra parentesi tonde, che descrive la condizione
  - Utile per descrivere la situazione in cui un oggetto cambia stato perché il valore dei suoi attributi è modificato dalla risposta a un messaggio inviato
- Evento segnale: consiste nella ricezione di un segnale
- Evento temporale: espressione che denota un lasso di tempo che deve trascorrere dopo un evento dotato di nome
  - Con <> si può far riferimento all'istante in cui l'oggetto è entrato nello stato corrente
  - Con <> si esprime qualcosa che deve accadere in un particolare momento

### 1.9.1.12 Azione

- Un'azione è un elemento avente un nome che è l'unità fondamentale della funzionalità eseguibile
- L'esecuzione di un'azione rappresenta una trasformazione o elaborazione nel sistema modellato, sia esso un sistema informatico o altro
- **Entry:** tutte le volte che si entra in uno stato viene generato un evento di entrata a cui può essere associato uno o più specifici comportamenti che vengono eseguiti prima che qualsiasi altra azione possa essere eseguita
- **Exit:** tutte le volte che si esce da uno stato viene generato un evento di uscita a cui può essere associato uno o più specifici comportamenti che vengono eseguiti come ultimo passo prima che lo stato venga lasciato
- **Do:** rappresenta il comportamento che viene eseguito all'interno dello stato
  - il comportamento parte subito dopo l'ingresso nello stato (dopo che è stato eseguito l'entry)
  - se questo comportamento termina mentre lo stato è ancora attivo viene generato un evento di completamento e nel caso sia stata specificata una transizione per il completamento allora viene eseguito l'exit e successivamente la transizione d'uscita
  - se invece viene innescata una transizione di uscita prima che il do termini, allora l'esecuzione del do è abortita, viene eseguito l'exit e poi la transizione
- Il comportamento di entry è stato definito per **permettere di fattorizzare comportamenti comuni** all'ingresso di uno stato, sui quali non possiamo mettere la guardia
- Senza la possibilità di poter specificare un comportamento di entry, il progettista avrebbe dovuto specificare su ogni transizione verso lo stato una azione effetto legata alle transizioni
- Il medesimo discorso vale per le exit, nella quale sono fattorizzati tutti quei comportamenti che devono essere eseguiti prima di uscire dallo stato
- È importante però ricordare che non vi è la possibilità di esprimere condizioni di guardia su questi comportamenti

### 1.9.1.13 Diagramma di Stato

- Transizione interna che esclude azioni di entrata e di uscita



- Attività interna: generazione di un thread concorrente che resta in esecuzione finché:
  - il thread (eventualmente) termina
  - si esce dallo stato attraverso una transizione esterna

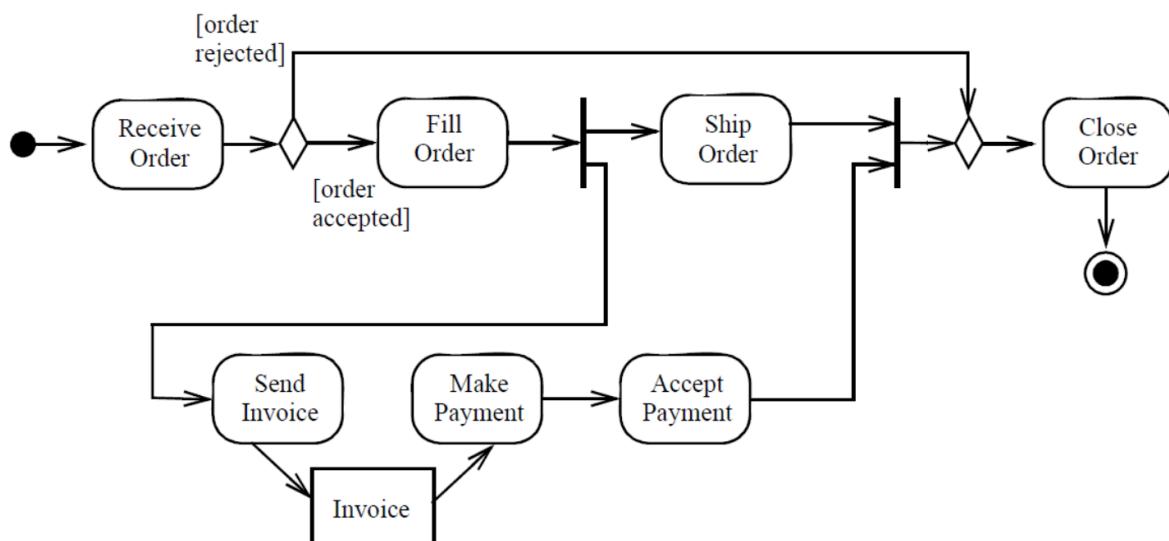


- Il diagramma di stato relativo a una singola classe / entità dovrebbe essere il più semplice possibile

- Le classi / entità con diagrammi di stato complicati hanno diversi problemi:
  - il codice risulta più difficile da scrivere perché le implementazioni dei metodi contengono molti blocchi condizionali
  - il testing è più arduo
  - è molto difficile utilizzare una classe dall'esterno se il comportamento dipende dallo stato in modo complesso
- Se una classe ha un numero troppo elevato di stati è bene considerare se esistono progetti alternativi, come per esempio scomporre la classe in due diverse classi
- Questa però non va presa come regola universale
- La scomposizione di una classe è sempre un'operazione molto delicata e va ponderata molto bene in quanto potrebbe portare a progetti fuorvianti o instabili
- A volte il male minore è proprio avere diagrammi di stato complessi

## 1.10 Diagramma delle attività

- I diagrammi delle attività descrivono il modo in cui diverse attività sono coordinate e possono essere usati per mostrare l'implementazione di una operazione
- Un diagramma delle attività mostra le attività di un sistema in generale e delle sotto-parti, specialmente quando un sistema ha diversi obiettivi e si desidera modellare le dipendenze tra essi prima di decidere l'ordine in cui svolgere le azioni
- I diagrammi delle attività sono utili anche per descrivere lo svolgimento dei singoli casi d'uso e la loro eventuale dipendenza da altri casi
- Sostanzialmente, modellano un processo
- Organizzano più entità in un insieme di azioni secondo un determinato flusso
- Si usano ad esempio per:
  - modellare il flusso di un caso d'uso (analisi)
  - modellare il funzionamento di un'operazione (progettazione)
  - modellare un algoritmo (progettazione)

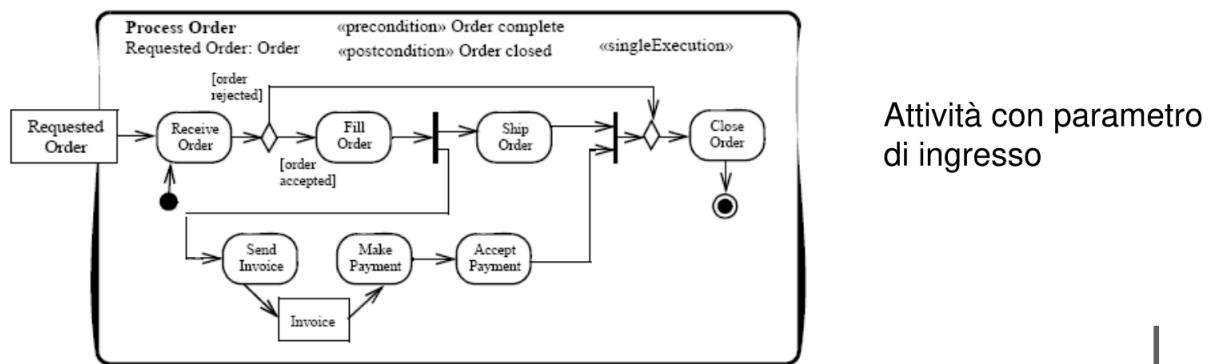
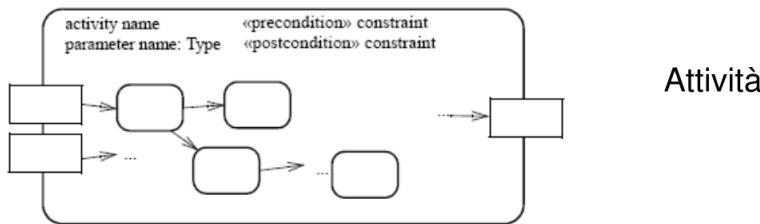


- **Attività:** indica un lavoro che deve essere svolto
  - Specifica un'unità atomica, non interrompibile e istantanea di comportamento
  - Da ogni attività possono uscire uno o più archi, che indicano il percorso da una attività ad un'altra
- **Arco:** è rappresentato come una freccia proprio come una transizione
  - A differenza di una transizione, un arco non può essere etichettato con eventi o azioni

- Un arco può essere etichettato con una condizione di guardia, se l'attività successiva la richiede
- **Oggetto:** rappresenta un oggetto “importante” usato come input/output di azioni
- **Sottoattività:** “nasconde” un diagramma delle attività interno a un’attività
- **Start e End Point:** punti di inizio e fine del diagramma
  - Gli End Point possono anche non essere presenti, oppure essere più di uno
- **Controllo:** nodi che descrivono il flusso delle attività

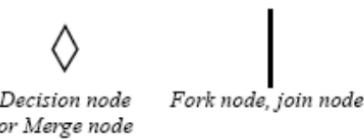
- Per capire la semantica dei diagrammi di attività, bisogna immaginare delle entità, dette **token**, che viaggiano lungo il diagramma
- Il flusso dei token definisce il flusso dell’attività
- I token possono rimanere fermi in un nodo azione/oggetto in attesa che si avveri
  - una condizione su un arco
  - oppure una precondizione o postcondizione su un nodo
- Il movimento di un token è atomico
- Un nodo azione viene eseguito quando sono presenti token su tutti gli archi in entrata, e tutte le precondizioni sono soddisfatte
- Al termine di un’azione, sono generati token su tutti gli archi in uscita

### 1.10.1.1 Notazione



- **Decision e Merge:** determinano il comportamento condizionale
  - **Decision** ha una singola transizione entrante e più transizioni uscenti in cui solo una di queste sarà prescelta
  - **Merge** ha più transizioni entranti e una sola uscente e serve a terminare il blocco condizionale cominciato con un decision
- **Fork e join:** determinano il comportamento parallelo:
  - quando scatta la transizione entrante, si eseguono in parallelo tutte le transazioni che escono dal fork
    - Con il parallelismo non è specificata la sequenza
    - Le fork possono avere delle guardie

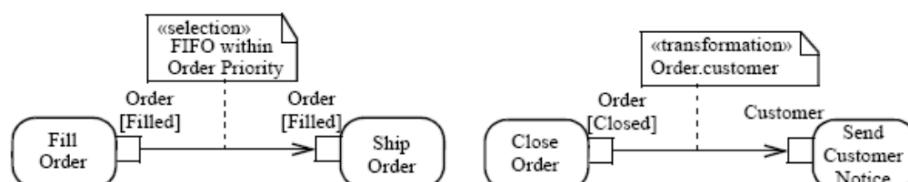
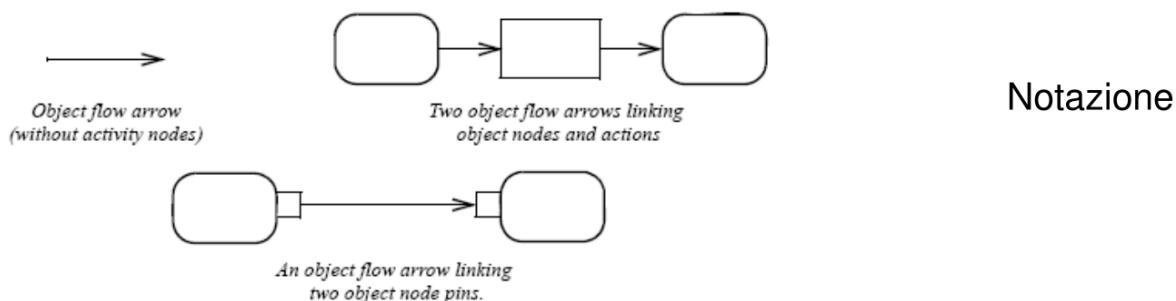
- Per la sincronizzazione delle attività è presente il costrutto di join che ha più transazioni entranti e una sola transazione uscente



{:height 343, :width 434}

### 1.10.2 Object Flow

- È un arco che ha oggetti o dati che fluiscono su di esso

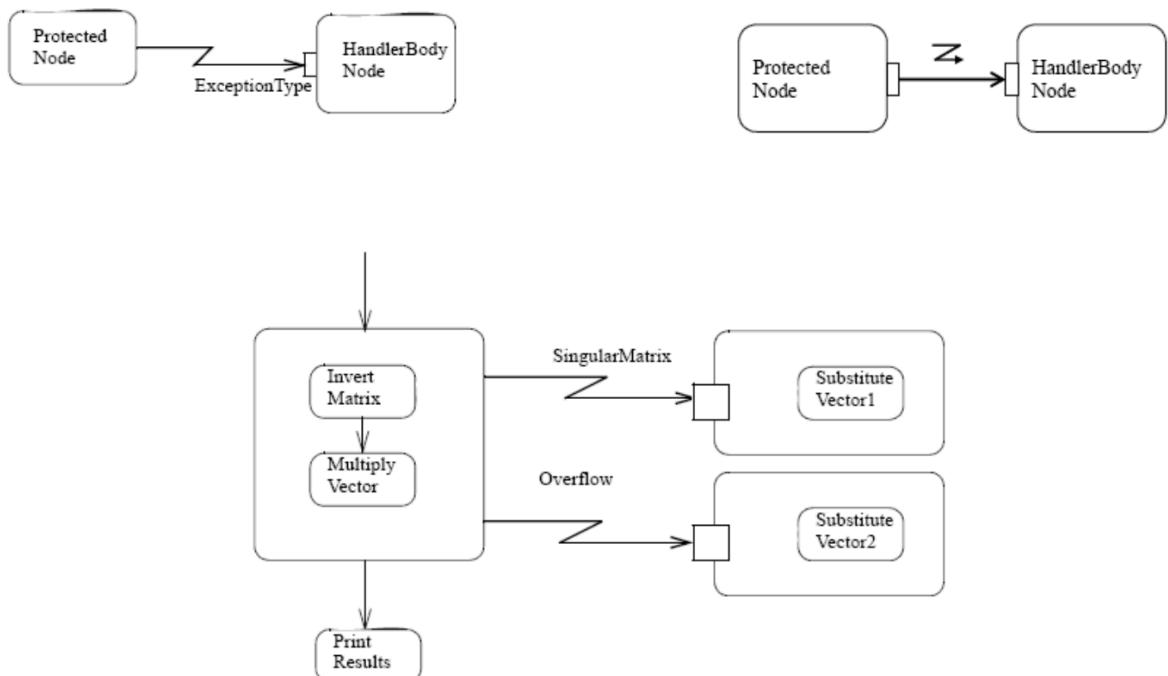


Esempio

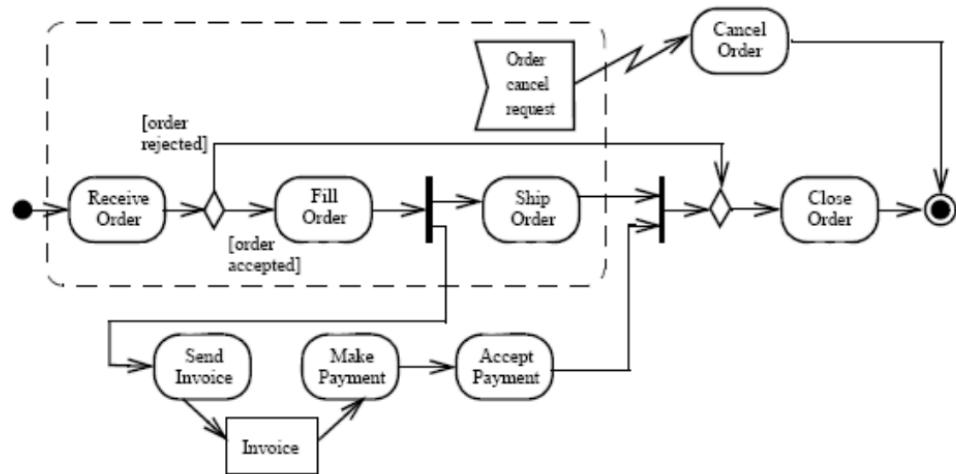
### 1.10.3 Segnali ed Eventi



### 1.10.4 Exception Handler



### 1.10.5 InterruptibleActivityRegion



## 2 Progetto

### 2.1 Analisi dei requisiti

#### 2.1.1 Requisiti

- I requisiti di un sistema rappresentano la descrizione
  - dei servizi forniti (le funzionalità)
  - dei vincoli operativi
- Il processo di ricerca, analisi, documentazione e verifica di questi servizi e vincoli è chiamato **Ingegneria dei Requisiti** (RE - Requirements Engineering)
- I requisiti di solito vengono forniti a livelli diversi di descrizione e questo porta a una prima classificazione...
- I Requisiti utente dichiarano:
  - Quali servizi il sistema dovrebbe fornire
  - I vincoli sotto cui deve operare
  - Sono requisiti molto astratti e di alto livello che vengono specificati nella prima fase interlocutoria con il committente
  - Tipicamente sono espressi in linguaggio naturale e corredati da qualche diagramma
- Requisiti di sistema definiscono:
  - Le funzioni, i servizi e i vincoli operativi del sistema in modo dettagliato
  - È una descrizione dettagliata di quello che il sistema dovrebbe fare
  - Il Documento dei Requisiti del Sistema deve essere preciso e definire esattamente cosa deve essere sviluppato
  - Tale documento fa spesso parte del contratto tra committente e azienda sviluppatrice
- ![[image.png]](images/image\_1710063362882\_0.png)

### **2.1.1.1 Requisiti di Sistema**

- I requisiti di sistema tipicamente vengono divisi in tre diverse tipologie:
  - Requisiti funzionali
  - Requisiti non funzionali
  - Requisiti di dominio

### **2.1.1.2 Requisiti Funzionali**

- Descrivono quello che il sistema dovrebbe fare
- Sono elenchi di servizi che il sistema dovrebbe fornire
- Per ogni servizio dovrebbe essere indicato:
  - come reagire a particolari input
  - come comportarsi in particolari situazioni
  - in alcuni casi specificare cosa il sistema NON dovrebbe fare
- Le specifiche dei requisiti funzionali dovrebbero essere:
  - complete: tutti i servizi definiti
  - coerenti: i requisiti non devono avere definizioni contraddittorie

### **2.1.1.3 Requisiti Non Funzionali**

- Non riguardano direttamente le specifiche funzionali
- Non devono essere necessariamente coerenti
- I principali tipi di requisiti non funzionali:
  - Requisiti del prodotto: specificano o limitano le proprietà complessive del sistema
  - affidabilità, prestazioni, protezione dei dati, disponibilità dei servizi, tempi di risposta, occupazione di spazio, capacità dei dispositivi di I/O, rappresentazione dei dati nelle interfacce, etc.
  - Requisiti Organizzativi: possono vincolare anche il processo di sviluppo adottato
    - politiche e procedure dell'organizzazione cliente e sviluppatrice, specifiche degli standard di qualità da adottare, uso di un particolare CASE tool, linguaggi di implementazione, limiti di budget, requisiti di consegna e milestone, etc.
  - Requisiti esterni: si identificano tutti i requisiti che derivano da fattori non provenienti dal sistema e dal suo processo di sviluppo
    - necessità di interoperabilità con altri sistemi software o hardware prodotti da altre organizzazioni
    - requisiti legislativi che devono essere rispettati affinché il sistema operi nella legalità → legislazioni sulla privacy dei dati
    - requisiti etici perché il sistema possa essere accettato dagli utenti e dal grande pubblico
- ![[image.png]](images/image\_1710063695471\_0.png)
- Uno dei maggiori problemi di questi requisiti è che possono essere difficili da verificare perché spesso sono espressi in modo vago e sono mescolati ai requisiti funzionali
  - Esempi: facilità d'uso, capacità di ripristino dopo un malfunzionamento
- Spesso contrastano o interagiscono con i requisiti funzionali
  - La protezione e la privacy dei dati contrastano con la facilità d'uso perché richiedono procedure spesso macchinose per l'utente...
  - ...occorre trovare un compromesso tra i requisiti o chiedere al committente quale sia più prioritario

- Vanno studiati ed analizzati con molta cura e precisione e indicati quanto più chiaramente possibile nel documento dei requisiti

#### **2.1.1.4 Requisiti di Dominio**

- I requisiti di dominio descrivono i vincoli che esistono nel dominio e che potrebbero non essere condivisi tra tutti coloro che saranno impegnati nello sviluppo del software
- Derivano dal dominio di applicazione del sistema
- Solitamente includono una terminologia propria del dominio del sistema o si riferiscono ai suoi concetti
- Poiché sono requisiti “specialistici” spesso gli ingegneri del software trovano difficile capire come questo tipo di requisiti si rapportino agli altri requisiti del sistema
- Sono requisiti che vanno comunque analizzati con molta cura perché riflettono i fondamenti del dominio dell’applicazione
  - l’analisi deve coinvolgere gli esperti del dominio per chiarire ogni dubbio sulla terminologia

#### **2.1.2 Analisi**

- Obiettivo
  - Specificare (cioè definire) le proprietà che il sistema dovrà avere senza descrivere una loro possibile realizzazione
- Risultato: una serie di documenti
  - contenenti la descrizione dettagliata dei requisiti
  - base di partenza per l’analisi del problema
- Per determinare in dettaglio i requisiti del sistema, è necessario
  - interagire il più possibile con l’utente, o con il potenziale utente
  - conoscere il più possibile l’area applicativa

##### **1. Raccolta dei requisiti**

Obiettivo: raccogliere tutte le informazioni su cosa il sistema deve fare secondo le intenzioni del cliente

##### **2. Analisi dei requisiti**

Obiettivo: definire il comportamento del sistema

##### **3. Analisi del dominio**

Obiettivo: definire la porzione del mondo reale, rilevante per il sistema

##### **4. Analisi e gestione dei rischi**

Obiettivo: identificare e gestire i possibili rischi che possono fare fallire o intralciare la realizzazione del sistema ![image.png](images/image\_1710064987414\_0.png)

#### **2.1.2.1 Raccolta dei Requisiti**

- Obiettivo: raccogliere tutte le informazioni su cosa il sistema deve fare secondo le intenzioni del cliente (si chiede al cliente cosa voglia che faccia l’applicazione)
- Non prevede passi formali, né ha una notazione specifica, perché dipende moltissimo dal particolare tipo di problema
- Risultato
  - un documento (testuale)
    - scritto dall’analista
    - discusso e approvato dal cliente

- ▶ una versione iniziale del vocabolario o glossario contenente la definizione precisa e non ambigua di tutti i termini e i concetti utilizzati
- Tipologie di persone coinvolte
  - ▶ Analista
  - ▶ Utente
  - ▶ Esperto del dominio (non sempre indispensabile)
- Metodi utilizzati
  - ▶ Interviste, questionari
  - ▶ Studio di documenti che esprimono i requisiti in forma testuale
  - ▶ Osservazione passiva o attiva del processo da modellare
  - ▶ Studio di sistemi software esistenti
  - ▶ Prototipi
- La gestione delle interviste è molto complessa
- I clienti potrebbero
  - ▶ Avere solo una vaga idea dei requisiti
  - ▶ Non essere in grado di esprimere i requisiti in termini comprensibili
  - ▶ Chiedere requisiti non realizzabili o troppo costosi
  - ▶ Fornire requisiti in conflitto
  - ▶ Essere poco disponibili a collaborare

### **2.1.2.2 Validazione dei Requisiti**

- Ogni requisito deve essere validato e negoziato con i clienti prima di essere riportato nel Documento dei Requisiti (se non è completamente validato si rifà l'intervista)
- Attività svolta in parallelo alla raccolta
- **Validità** - il nuovo requisito è inerente il problema da risolvere?
- **Consistenza** - il nuovo requisito è in sovrapposizione e/o in conflitto con altri requisiti?
- **Realizzabilità** - il nuovo requisito è realizzabile con le risorse disponibili (hardware, finanziamenti, ...)?
- **Completezza** - esiste la possibilità che ci siano requisiti rimasti ignorati?

### **2.1.2.3 Documento dei Requisiti**

- Il Documento dei Requisiti deve specificare in modo chiaro e univoco cosa il sistema dovrà fare
- I requisiti dovrebbero essere:
  - ▶ Chiari
  - ▶ Corretti
  - ▶ Completati
  - ▶ Concisi
  - ▶ Non ambigui
  - ▶ Precisi
  - ▶ Consistenti
  - ▶ Tracciabili
  - ▶ Modificabili
  - ▶ Verificabili
- Il Documento dei Requisiti deve contenere la versione iniziale del dizionario dei termini

- Un buon modo per organizzare i requisiti e facilitare la tracciabilità è quello di elencare i requisiti in una tabella
- ! [image.png] (images/image\_1710065295316\_0.png)
- Se durante il processo di sviluppo ci si riferisce sempre allid del requisito diventa facile collegare le diverse fasi e garantire una tracciabilità requisito-codice

#### **2.1.2.4 Cambiamento dei Requisiti**

- È normale che i requisiti subiscano modificazioni ed evolvano nel tempo
- Requisiti esistenti possono essere rimossi o modificati
- Nuovi requisiti possono essere aggiunti in una fase qualsiasi del ciclo di sviluppo
- Tali cambiamenti
  - Sono la norma, non l'eccezione; i sistemi vanno progettati avendo in mente il cambiamento
  - Possono diventare un grosso problema se non opportunamente gestiti
- Più lo sviluppo è avanzato, più il cambiamento è costoso
  - Modificare un requisito appena definito è facile
  - Modificare lo stesso requisito dopo che è stato implementato nel software potrebbe essere molto costoso
- Ogni cambiamento deve essere accuratamente analizzato per valutare
  - La fattibilità tecnica, cioè, lo si può implementare, indipendentemente dal costo?
  - L'impatto sul resto del sistema
  - Il costo
- Consiglio - sviluppare sistemi che
  - Siano il più possibile resistenti ai cambiamenti dei requisiti
    - Inizialmente, eseguano esclusivamente e nel modo migliore i soli compiti richiesti
    - In seguito, siano in grado di sostenere l'aggiunta di nuove funzionalità senza causare "disastri" strutturali e/o comportamentali
- Tenete traccia dei cambiamenti anche nella tabella dei requisiti

#### **2.1.2.5 Analisi del Dominio**

- Obiettivo: definire la porzione del mondo reale rilevante per il sistema, quindi una visione astratta del mondo reale
- Principio fondamentale: Astrazione Permette di gestire la complessità intrinseca del mondo reale
  - ignorare gli aspetti che non sono importanti per lo scopo attuale
  - concentrarsi maggiormente su quelli che lo sono
- Risultato: prima versione del vocabolario partendo dai "sostantivi" che si trovano nei requisiti
- L'analisi del dominio può essere effettuata anche considerando un gruppo di sistemi afferenti alla stessa area applicativa
- Esempi di aree applicative:
  - il controllo del traffico aereo
  - la gestione aziendale
  - le operazioni bancarie
  - ...
- In tal caso, è possibile
  - identificare entità e comportamenti comuni a tutti i sistemi

- ▶ realizzare schemi di progettazione e componenti software riutilizzabili nei diversi sistemi

### **2.1.2.6 Analisi dei Requisiti**

- Obiettivo: definire il comportamento del sistema da realizzare
- Risultato: un modello comportamentale (o modello dinamico) che descrive in modo chiaro e conciso le funzionalità del sistema
- L'analisi dei requisiti deve partire dalla raccolta, che è un elenco di requisiti, e definire per bene qual è il comportamento del sistema
- Strategia:
  - ▶ Scomposizione funzionale (mediante analisi top-down) ▶ identificare le singole funzionalità previste dal sistema
  - ▶ Astrazione procedurale ▶ considerare ogni operazione come una singola entità, nonostante tale operazione sia effettivamente realizzata da un insieme di operazioni di più basso livello
- Attenzione: La scomposizione in funzioni è molto volatile a causa del continuo cambiamento dei requisiti funzionali
- Come prima cosa vanno analizzati in modo sistematico tutti i requisiti inseriti nella Tabella dei requisiti
- Bisogna porre particolare attenzione ai sostantivi e ai verbi che compaiono nel testo di specifica dei requisiti
- Dall'analisi dei sostantivi è possibile formalizzare la conoscenza sul dominio applicativo → costruzione di un primo modello del dominio
- Dall'analisi dei verbi è possibili individuare linsieme delle azioni che il sistema dovrà compiere → modello dei casi duso
- Aggiornare costantemente la Tabella dei Requisiti → dall'analisi nascono sempre nuovi requisiti

### **2.1.2.7 Vocabolario**

- Nella modellazione del dominio è di fondamentale importanza usare solo la terminologia di quello specifico dominio
- Il vocabolario è una lista dei termini usati nella specifica dei requisiti a cui viene data una definizione precisa e non ambigua
- È uno dei fattori chiave per migliorare la comunicazione tra i diversi attori del processo di sviluppo, in particolare tra analisti e progettisti
- Ciascuna entità del dominio che si evince dai requisiti può essere espressa come classe UML e messa in relazione logica con le altre entità andando a creare il primo modello del dominio

### **2.1.2.8 Analisi e gestione dei rischi**

- Ci si riferisce generalmente ai rischi di tipo legale
- Analisi sistematica e completa di tutti i possibili rischi che possono fare fallire o intralciare la realizzazione del sistema in una qualsiasi fase del processo di sviluppo
- Ogni rischio presenta due caratteristiche:
  - ▶ Probabilità che avvenga non esistono rischi con una probabilità del 100% (sarebbero vincoli al progetto)
  - ▶ Costo se il rischio si realizza, ne seguono effetti indesiderati e/o perdite
- Rischi relativi ai requisiti
  - ▶ I requisiti sono perfettamente noti?

- Il rischio maggiore è quello di costruire un sistema che non soddisfa le esigenze del cliente
- Rischi relativi alle risorse umane
  - È possibile contare sulle persone e sull'esperienza necessarie per lo sviluppo del progetto?
- Rischi relativi alla protezione e privacy dei dati
  - Di che tipo sono i dati che dobbiamo trattare?
  - Quali sono i possibili attacchi informatici a cui il sistema può essere soggetto?
- Rischi tecnologici
  - Si sta scegliendo la tecnologia corretta?
  - Si è in grado di aggregare correttamente i vari componenti del progetto (ad es., GUI, DB, ...)?
  - Quali saranno i possibili futuri sviluppi della tecnologia?
- Rischi politici
  - Ci sono delle forze politiche (anche in senso lato) in grado di intralciare lo sviluppo del progetto?
- Strategia reattiva o “alla Indiana Jones”
  - “Niente paura, troverò una soluzione”
- Strategia preventiva
  - Si mette in moto molto prima che inizi il lavoro tecnico
  - Si individuano i rischi potenziali, se ne valutano le probabilità e gli effetti e si stabilisce un ordine di importanza
  - Si predisponde un piano che permetta di reagire in modo controllato ed efficace
    - Più grande è un rischio, dove per grande possiamo intendere il prodotto di probabilità per costo
    - Maggiore sarà l'attenzione che bisognerà dedicargli

### 2.1.3 Sicurezza e Privacy

#### 2.1.3.1 Nuova normativa

##### 2.1.3.1.1 General Data Protection Regulation

- Dal 25/5/2018 sostituisce la Data Protection Directive
- Obbligo di aderenza (compliance) di un prodotto software che tratti dati personali ai principi della GDPR
  - privacy by design & by default(misure tecniche e organizzative)
  - minimalità, proporzionalità
  - anonimizzazione, pseudonimizzazione
  - trasferimento dati fuori dalla EU (occhio al cloud!)
  - adeguatezza delle misure di sicurezza
- Non è qualcosa che si possa “aggiungere dopo”, a sistema già progettato: va considerato fin dall'inizio
  - ma non è (solo) un vincolo: è un'opportunità per creare valore

**Pseudonimizzazione:** processo di trattamento dei dati personali in modo tale che i dati non possano più essere attribuiti a un interessato specifico senza l'utilizzo di informazioni aggiuntive, sempre che tali informazioni aggiuntive siano conservate separatamente e soggette a misure tecniche e organizzative intese a garantire la non attribuzione a una persona identificata o identificabile.

### **2.1.3.1.2 Principi**

I dati personali devono essere:

- trattati in modo lecito, equo e trasparente nei confronti dell'interessato (“[liceità, equità e trasparenza](#)”)
- raccolti per finalità determinate, esplicite e legittime, e successivamente trattati in modo non incompatibile con tali finalità
- adeguati, pertinenti e limitati a quanto necessario rispetto alle finalità per le quali sono trattati (“[minimizzazione dei dati](#)”)
- esatti e, se necessario, aggiornati
  - devono essere prese tutte le misure ragionevoli per cancellare o rettificare tempestivamente i dati inesatti rispetto alle finalità per le quali sono trattati (“[esattezza](#)”); quindi bisogna dare all'utente la possibilità di aggiornarli
- conservati in una forma che consenta l'identificazione degli interessati per un arco di tempo non superiore al conseguimento delle finalità per le quali sono trattati
  - i dati personali possono essere conservati per periodi più lunghi a condizione che siano trattati esclusivamente per finalità di archiviazione nel pubblico interesse o per finalità di ricerca scientifica e storica o per finalità statistiche, conformemente all'articolo 83
- trattati in modo da garantire un'adeguata sicurezza dei dati personali, compresa la protezione, mediante misure tecniche e organizzative adeguate, da trattamenti non autorizzati o illeciti e dalla perdita, dalla distruzione o dal danno accidentali (“[integrità e riservatezza](#)”); quindi bisogna garantire che non ci siano trattamenti illeciti dei dati degli utenti, e che non vengano persi

### **2.1.3.1.3 Articolo 25**

#### **Protezione dei dati fin dalla progettazione e protezione di default**

1. Tenuto conto dello stato dell'arte e dei costi di attuazione, nonché della natura, del campo di applicazione, del contesto e delle finalità del trattamento, come anche dei rischi di varia probabilità e gravità per i diritti e le libertà delle persone fisiche costituiti dal trattamento, [sia al momento di determinare i mezzi del trattamento sia all'atto del trattamento stesso](#) il responsabile del trattamento mette in atto [misure tecniche e organizzative adeguate](#), quali la pseudonimizzazione, volte ad attuare i principi di protezione dei dati, quali la minimizzazione, in modo efficace e a integrare nel trattamento le necessarie garanzie al fine di soddisfare i requisiti del presente regolamento e tutelare i diritti degli interessati
2. Il responsabile del trattamento mette in atto misure tecniche e organizzative adeguate per garantire che siano trattati, [di default](#), solo i dati personali necessari per ogni specifica finalità del trattamento; ciò vale per la quantità dei dati raccolti, l'estensione del trattamento, il periodo di conservazione e l'accessibilità. [In particolare dette misure garantiscono che, di default, non siano resi accessibili dati personali a un numero indefinito di persone fisiche senza l'intervento della persona fisica](#); cioè, normalmente, nessuno può far vedere quei dati, tranne la persona stessa

### **2.1.3.1.4 Articolo 32:**

#### **Sicurezza del Trattamento**

1. Tenuto conto dello stato dell'arte e dei costi di attuazione, nonché della natura, del campo di applicazione, del contesto e delle finalità del trattamento, come anche del rischio di varia probabilità e gravità per i diritti e le libertà delle persone fisiche, il responsabile del trattamento e l'incaricato del trattamento mettono in atto misure tecniche e organizzative adeguate per garantire un **livello di sicurezza adeguato al rischio**, che comprendono tra l'altro:
  - la **pseudonimizzazione** e la **cifratura dei dati personali**
  - la capacità di assicurare la **continua riservatezza**, integrità, disponibilità e resilienza dei sistemi e dei servizi che trattano i dati personali;
  - la capacità di **ripristinare tempestivamente la disponibilità** e l'accesso dei dati in caso di incidente fisico o tecnico;
  - una procedura per provare, verificare e valutare regolarmente l'efficacia delle misure tecniche e organizzative al fine di garantire la sicurezza del trattamento; cioè bisogna non solo mettere in campo queste misure, ma anche testarle regolarmente
2. Nel valutare l'adeguato livello di sicurezza, si tiene conto in special modo [dei rischi presentati] da trattamenti di dati derivanti in particolare dalla distruzione, dalla perdita, dalla modifica, dalla divulgazione non autorizzata o dall'accesso, in modo accidentale o illegale, a dati personali trasmessi, memorizzati o comunque trattati
3. L'adesione a un **codice di condotta** approvato di cui all'articolo 40 o a un **meccanismo di certificazione** approvato di cui all'articolo 42 può essere utilizzata come elemento per dimostrare la conformità ai requisiti di cui al paragrafo 1 del presente articolo

Quindi per dimostrare di aver seguito tutti i passi descritti nell'articolo ci si avvale di una certificazione o di un codice di condotta.

#### 2.1.3.1.5 Trasferimento Dati a Paesi Terzi

- Articolo 45:
  - Il trasferimento di dati personali verso un paese terzo o un'organizzazione internazionale è ammesso se la Commissione ha deciso che il paese terzo, o un territorio o uno o più settori specifici all'interno del paese terzo, o l'organizzazione internazionale in questione garantiscano un livello di protezione adeguato. In tal caso il trasferimento non necessita di autorizzazioni specifiche.
- Articolo 46:
  - In mancanza di una decisione ai sensi dell'articolo 45, il responsabile del trattamento o l'incaricato del trattamento può trasferire dati personali verso un paese terzo o un'organizzazione internazionale solo se ha offerto garanzie adeguate e a condizione che siano disponibili diritti azionabili degli interessati e mezzi di ricorso effettivi per gli interessati.

#### 2.1.3.2 Concetti di Base

##### 2.1.3.2.1 Sicurezza Informatica

Salvaguardia dei sistemi informatici da potenziali rischi e/o violazioni dei dati

- Obiettivo dell'attacco = contenuto informativo

- Sicurezza informatica = preoccuparsi di
  - impedire l'accesso ad utenti non autorizzati
  - regolamentare l'accesso ai diversi soggetti...
  - ... che potrebbero avere autorizzazioni diverse (relative solo a certe operazioni e non altre)
- per evitare che i dati appartenenti al sistema informatico vengano copiati, modificati o cancellati

Cosa Proteggere:

- L'informazione...
  - Riservatezza: solo determinati utenti possono avere accesso alle informazioni
  - Integrità ed Autenticità: nessuno deve modificare impropriamente quelle informazioni
  - Disponibilità: le informazioni sono disponibili nel momento c'è la necessità di accedervi
- trattata per mezzo di calcolatori e reti
  - Accesso controllato
    - Tre aspetti: **identificazione, autenticazione, autorizzazione**
  - Funzionamento affidabile; per il concetto di affidabilità riguardare gli argomenti precedenti

### 2.1.3.2.2 Violazioni

- Le violazioni possono essere molteplici:
  - tentativi non autorizzati di accesso a zone riservate
  - furto di identità digitale o di file riservati
  - utilizzo di risorse che l'utente non dovrebbe potere utilizzare
  - ecc.
- La sicurezza informatica si occupa anche di prevenire eventuali **Denial of Service** (DoS)
  - sono attacchi sferrati al sistema con l'obiettivo di rendere inutilizzabili alcune risorse onde danneggiare gli utenti

### 2.1.3.2.3 Fattori Influenti

- Nella scelta delle misure di sicurezza **incidono diverse caratteristiche** dell'informazione e del contesto:
  - Dinamicità
  - Dimensione e tipo di accesso
  - Tempo di vita
  - Costo di generazione
  - Costo in caso di violazione (di riservatezza, di integrità, di disponibilità)
  - Valore percepito (dall'utente e dal detentore) e tipologia di attaccante

### La Catena degli Anelli

La forza di un sistema è pari alla forza dell'anello più debole che lo compone. Non bisogna tralasciare nulla, nemmeno il più piccolo dettaglio

### 2.1.3.2.4 Metodi di Protezione

- Legali: ci sono leggi che regolano i casi in cui un sistema informatico viene violato, attraverso sanzioni ad esempio
- Organizzativi
- Tecnologici: metodi che ci interessano di più
  - Fisici: un oggetto fisico che viene utilizzato per proteggere l'accesso ai dati, ad esempio un tesserino
  - Crittografici: cifrare i dati in modo che siano difficilmente interpretabili da un attaccante
  - Biometrici: caratteristiche fisiche misurabili che identificano univocamente un individuo

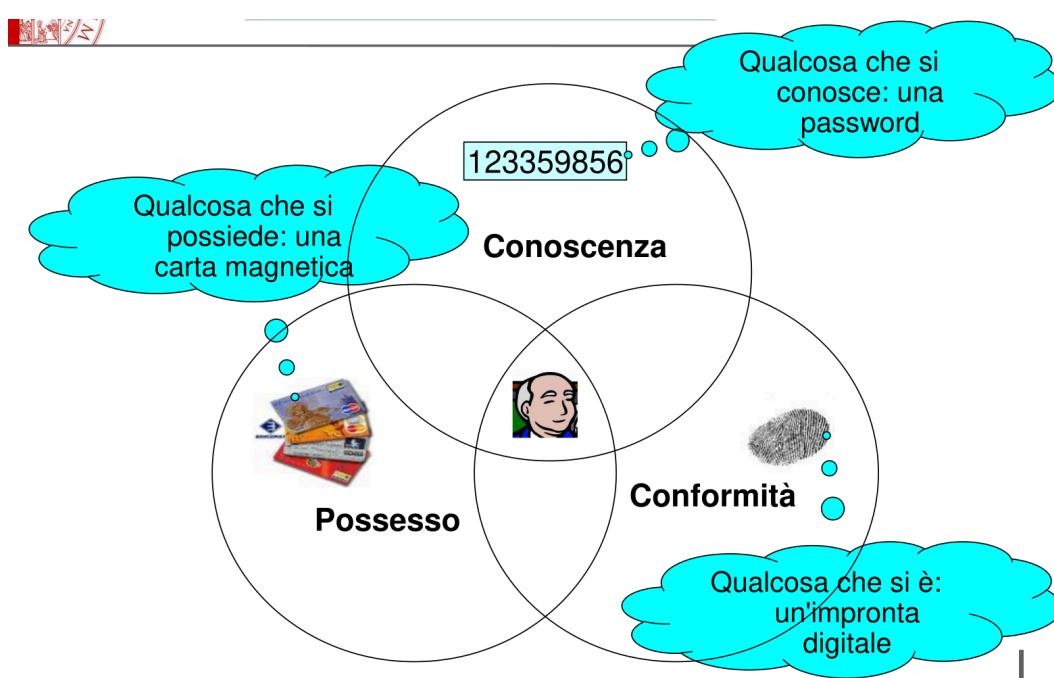
Attuiamo questi metodi per:

- Prevenzione: evitare che un attaccante entri nel sistema
- Rilevazione: scopriamo eventuali attacchi o violazioni
- Contenimento: se c'è una violazione dei sistemi si cerca di tutti i modi di contenere eventuali danni
- Ripristino: ripristino del sistema all'istante precedente al danno

#### 2.1.3.2.5 Protezione Fisica

- Essenziale: la vulnerabilità fisica non sia la più facilmente attaccabile
- Efficace per i sistemi
  - criteri che esistono da ben prima dei problemi di sicurezza informatica
- Efficace per i dati
  - purché si conosca il comportamento dei sistemi che li trattano (percorsi accessibili, copie temporanee in memoria e su disco, ...)
  - costo di generazione
  - purché si prevedano metodi aggiuntivi per contenere gli effetti delle violazioni fisiche dei sistemi (es. furto)

#### 2.1.3.2.6 Autenticazione Forte



L'autenticazione forte è garantita esistono le seguenti condizioni

- Possesso
- Conoscenza

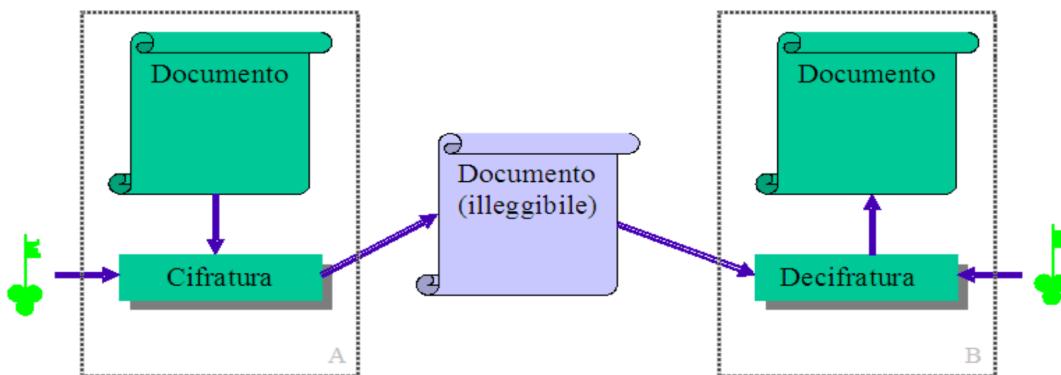
- Conformità

Ovviamente garantire tutte e tre le condizioni ha un costo.

### 2.1.3.3 Crittografia

- Crittografia simmetrica
  - garantire riservatezza
  - non identifica né autentica
  - **funzionamento:** esiste una sola chiave che viene utilizzata per cifrare; ad esempio prendo un dato, lo cifro, e ottengo un dato che non è immediatamente comprensibile; la stessa chiave è usata per decifrare.
- Crittografia asimmetrica
  - garantisce riservatezza
  - obiettivo: identificazione e quindi autenticazione e paternità
  - **funzionamento:** le chiavi sono due; c'è una che è in esclusivo possesso del cfratore e una che invece è pubblica, quindi chiave privata e chiave pubblica
- Infrastrutture per la certificazione della chiave pubblica
  - Terze parti fidate che possano certificare l'autenticità di una chiave pubblica

#### 2.1.3.3.1 Crittografia Simmetrica



- Classica e moderna, implementata con dispositivi segreti, algoritmi segreti o chiavi segrete
- Tipicamente tecniche derivanti dalla teoria dell'informazione (confusione e diffusione)
- Una singola chiave cifra e decifra

#### 2.1.3.3.2 Crittografia Asimmetrica

- Moderna (ufficialmente 1976)
- Basata sulla teoria della complessità computazionale
- Due chiavi correlate ma non (facilmente) calcolabili l'una dall'altra
  - La **chiave privata** è strettamente personale e quindi *identifica* il possessore
  - L'uso di una determinata chiave privata può essere verificato da chiunque per mezzo della corrispondente **chiave pubblica**

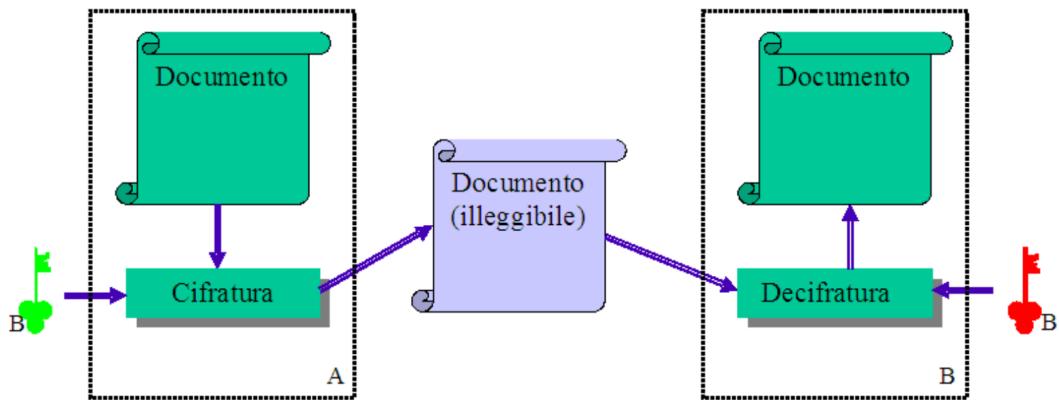


Figure 64: Procedimento di cifratura

Se io voglio che il mio documento sia letto solamente da chi deve poterlo leggere, io cifro quel documento con la chiave pubblica, quindi se io cifro con la chiave privata è decifrabile solo dalla chiave pubblica, se io cifro con la chiave pubblica è decifrabile solo dalla chiave privata

### Firma Digitale:

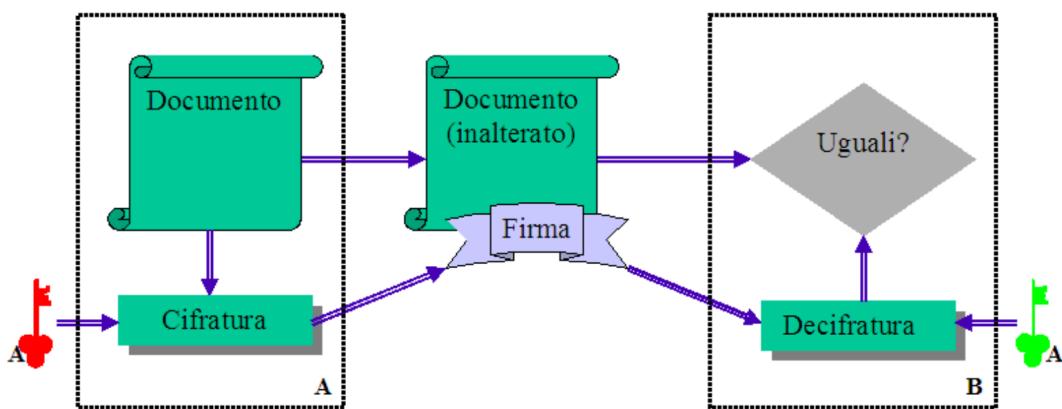


Figure 65: Procedimento di firma

Viceversa, firma, autenticità: se io cifro il documento con la mia chiave pubblica genero un documento che è leggibile, ma che di fianco ha la mia firma, a questo punto chiunque sia in possesso della chiave pubblica può cifrare o decifrare la firma, e vedere se i documenti sono uguali, cioè controllo che la chiave che è stata usata per cifrare, per produrre la firma, corrisponde alla chiave pubblica che ho in mano io.

#### 2.1.3.3.3 Confronto tra i due tipo di crittografia

	Simmetrica	Asimmetrica
Autenticità		
Integrità		
Riservatezza		
Efficienza		
Robustezza		
Chiavi		
Lunghezza		
Numero per ogni utente		
Protezione		

La cifratura asimmetrica purtroppo è poco efficiente perché le procedure di cifratura e decifratura evidentemente sono complicate. Però, per quanto riguarda le chiavi simmetriche, bisogna averne un numero importante, perché se ho una sola chiave che cifra e decifra tutto, se capita in mano a un malintenzionato ha accesso a tutto.

#### 2.1.3.3.4 Biometria

- Componente cardine per la terna di fattori per l'**autenticazione forte**
  - qualcosa che sei, qualcosa che hai, qualcosa che sai
- Problemi non ancora risolti:
  - Ostinatamente usata per l'*identificazione*
    - lunghe operazioni di confronto
    - cattivo bilanciamento tra falsi positivi e falsi negativi
  - Meglio per l'*autenticazione*
    - un solo confronto, minore probabilità di errori
  - Prestazioni più sfumate rispetto ad altre tecniche
    - difficile tuning tra falsi positivi e falsi negativi
  - Impossibilità di sostituzione in caso di compromissione

#### 2.1.3.3.5 Sicurezza delle Password

- Aspetto da sempre fondamentale per:
  - impedire l'accesso a utenti non autorizzati
  - nascondere e/o vincolare l'accesso a documenti
- Diverse categorie:
  - Deboli
  - Semplici
  - Intelligenti
  - Strong

#### 2.1.3.4 Analizzare e Progettare la Sicurezza

##### 2.1.3.4.1 Sistema Sicuro

- La definizione di una politica di sicurezza deve tenere conto di vincoli tecnici, logistici, amministrativi, politici ed economici, imposti dalla struttura organizzativa in cui il sistema opera

- Per questo serve introdurre la sicurezza sin dalle prime fasi di analisi dei requisiti di un nuovo sistema
  - le vigenti leggi, le politiche e i vincoli aziendali sono la base di partenza per la definizione di un **piano per la sicurezza**
- Un'applicazione o un servizio possono consistere di uno o più componenti funzionali allocati localmente o distribuiti sulla rete
- La sicurezza viene vista come un **processo complesso**, come una catena di caratteristiche
  - dalla computer system security, network security, application-level security sino alle problematiche di protezione dei dati sensibili
- La sfida maggiore lanciata ai progettisti è quella di **progettare applicazioni sicure e di qualità** che tengano conto in modo strutturato di tutti gli aspetti della sicurezza sin dalle prime fasi di analisi del sistema

#### 2.1.3.4.2 Sistemi Critici

- I **sistemi critici** sono sistemi tecnici o socio-tecnici da cui dipendono persone o aziende
- Se questi sistemi non forniscono i loro servizi come ci si aspetta possono verificarsi seri problemi e importanti perdite
- Ci sono tre tipi principali di sistemi critici:
  - **Sistemi safety-critical:** i fallimenti possono provocare incidenti, perdita di vite umane o seri danni ambientali
  - **Sistemi mission-critical:** i malfunzionamenti possono causare il fallimento di alcune attività e obiettivi diretti
  - **Sistemi business-critical:** i fallimenti possono portare a costi molto alti per le aziende che li usano
- La proprietà più importante di un sistema critico è la sua **fidatezza**
- Sistema fidato = **disponibilità + affidabilità + sicurezza e protezione**
- Ci sono diverse ragioni per le quali la fidatezza è importante:
  - I sistemi non affidabili, non sicuri e non protetti sono rifiutati dagli utenti
  - I costi di un fallimento del sistema potrebbero essere enormi
  - I sistemi inaffidabili possono causare perdita di informazioni
- Componenti del sistema che possono causare fallimenti (in ordine dal più affidabile al meno affidabile):
  - Hardware: può fallire a causa di errori nella progettazione, di un guasto a un componente o perché i componenti hanno terminato la loro vita naturale
  - Software: può fallire a causa di errori nelle sue specifiche, nella sua progettazione o nella sua implementazione
  - Operatori umani: possono sbagliare a interagire con il sistema
- Con l'aumentare dell'affidabilità di software e hardware gli errori umani sono diventati la più probabile causa di difetto di un sistema. Una persona può non usarlo come dovrebbe, come abbiamo pensato che dovesse utilizzarlo o che volontariamente usa il sistema in maniera inopportuna per evitare la fidatezza del

sistema e quindi per distruggere l'affidabilità, distruggere la sicurezza, distruggere la protezione e distruggere la disponibilità.

- La sicurezza e la protezione dei sistemi critici sono diventate sempre più importanti con l'aumentare delle connessioni di rete
- Da una parte le connessioni di rete espongono il sistema ad attacchi da parte di malintenzionati
- Dall'altra parte la rete fa in modo che i dettagli delle vulnerabilità siano facilmente divulgati e facilita la distribuzione di patch
- Esempi di attacchi sono:
  - virus
  - usi non autorizzati dei servizi
  - modifiche non autorizzate al sistema e ai suoi dati
  - **Exploit:** metodo che sfrutta un bug o una vulnerabilità, per l'acquisizione di privilegi
  - **Buffer Overflow:** fornire al programma più dati di quanto esso si aspetti di ricevere, in modo che una parte di questi vadano scritti in zone di memoria dove sono, o dovrebbero essere, altri dati o lo stack del programma stesso
  - **Shell code:** sequenza di caratteri che rappresenta un codice binario in grado di lanciare una shell, può essere utilizzato per acquisire un accesso alla linea di comando
  - **Sniffing:** attività di intercettazione passiva dei dati che transitano in una rete
  - **Cracking:** modifica di un software per rimuovere la protezione dalla copia, oppure per ottenere accesso a un'area riservata
  - **Spoofing:** tecnica con la quale si simula un indirizzo IP privato da una rete pubblica facendo credere agli host che l'IP della macchina server da contattare sia il suo
  - **Trojan:** programma che contiene funzionalità maliziose; la vittima è indotta a eseguire il programma poiché questo viene spesso inserito nei videogiochi pirati
  - **Denial of Service:** il sistema viene forzatamente messo in uno stato in cui i suoi servizi non sono disponibili, influenzando così la disponibilità del sistema

### 2.1.3.5 Introduzione alla Security Engineering

#### 2.1.3.5.1 Security Engineering

**Security Engineering is concerned with how to develop and maintain systems that can resist malicious attacks intended to damage a computer-based system or its data**



Da: Software Engineering 8 – I.Sommerville - Addison Wesley – Cap. 30

- L'ingegneria della sicurezza è parte del più vasto campo della sicurezza informatica
- Nell'ingegnerizzazione di un sistema software non si può prescindere dalla consapevolezza delle minacce che il sistema dovrà affrontare e dei modi in cui tali minacce possono essere neutralizzate
- Quando si considerano le problematiche di sicurezza nell'ingegnerizzazione di un sistema vanno presi in considerazione due aspetti diversi:

- la sicurezza dell'applicazione
- la sicurezza dell'infrastruttura su cui il sistema è costruito



#### 2.1.3.5.2 Applicazione e Infrastruttura

- La sicurezza di una applicazione è un problema di ingegnerizzazione del software dove gli ingegneri devono garantire che il **sistema sia progettato per resistere agli attacchi**
- La sicurezza dell'infrastruttura è invece un problema manageriale nel quale gli amministratori dovrebbero garantire che **l'infrastruttura sia configurata per resistere agli attacchi**
- Gli amministratori dei sistemi devono:
  - inizializzare l'infrastruttura in modo tale che tutti i servizi di sicurezza siano disponibili
- monitorare e riparare eventuali falle di sicurezza che emergono durante l'uso del software

#### 2.1.3.5.3 Gestione della Sicurezza

- Gestione degli utenti e dei permessi:
  - inserimento e rimozione di utenti dal sistema
  - autenticazione degli utenti
  - creazione di appropriati permessi per gli utenti
- Deployment e mantenimento del sistema:
  - installazione e configurazione dei software e del middleware
  - aggiornamento periodico del software con tutte le patch disponibili
- Controllo degli attacchi, rilevazione e ripristino
  - controllo del sistema per accessi non autorizzati
  - identificazione e messa in opera di strategie contro gli attacchi
  - backup per ripristinare il normale utilizzo dopo un attacco

#### 2.1.3.6 Glossario e Minacce

##### 2.1.3.6.1 Glossario della Sicurezza

- **Bene** (Asset): una risorsa del sistema che deve essere protetta
- **Esposizione** (Exposure): possibile perdita o danneggiamento come risultato di un attacco andato a buon fine
  - Potrebbe essere una perdita o un danneggiamento di dati o una perdita di tempo nel ripristino del sistema dopo l'attacco

- **Vulnerabilità** (Vulnerability): una debolezza nel sistema software che potrebbe essere sfruttata per causare una perdita o un danno
- **Attacco** (Attack): sfruttamento di una vulnerabilità del sistema allo scopo di esporre un bene
- **Minaccia** (Threat): circostanza che ha le potenzialità per causare perdite e danni
- **Controllo** (Control): una misura protettiva che riduce una vulnerabilità del sistema

#### **2.1.3.6.2 Tipi di Minacce**

- **Minacce alla riservatezza del sistema o dei suoi dati**: le informazioni possono essere svelate a persone o programmi non autorizzati
- **Minacce all'integrità del sistema o dei suoi dati**: i dati o il software possono essere danneggiati o corrotti
- **Minacce alla disponibilità del sistema o dei suoi dati**: può essere negato l'accesso agli utenti autorizzati al software o ai dati
- Queste minacce sono interdipendenti
  - Se un attacco rende il sistema non disponibile, la modifica sulle informazioni potrebbe non avvenire, rendendo così il sistema non integro

#### **2.1.3.6.3 Tipi di Controllo**

- **Controlli per garantire che gli attacchi non abbiano successo**: la strategia è quella di progettare il sistema in modo da evitare i problemi di sicurezza
  - i sistemi militari sensibili non sono connessi alla rete pubblica
  - crittografia
- **Controlli per identificare e respingere attacchi**: la strategia è quella di monitorare le operazioni del sistema e identificare pattern di attività atipici, nel caso agire di conseguenza (spegnere parti del sistema, restringere l'accesso agli utenti, ..)
- **Controlli per il ripristino**
  - backup, replicazione, polizze assicurative

#### **Esempio**

- Un sistema informativo ospedaliero mantiene le informazioni personali sui pazienti e sui loro trattamenti
- Il sistema deve essere accessibile da differenti ospedali e cliniche attraverso un'interfaccia web
- Lo staff ospedaliero deve utilizzare una specifica coppia <username, password> per autenticarsi, dove lo username è il nome del dipendente
- Il sistema richiede password che siano lunghe almeno 8 caratteri, ma consente ogni password senza ulteriori verifiche

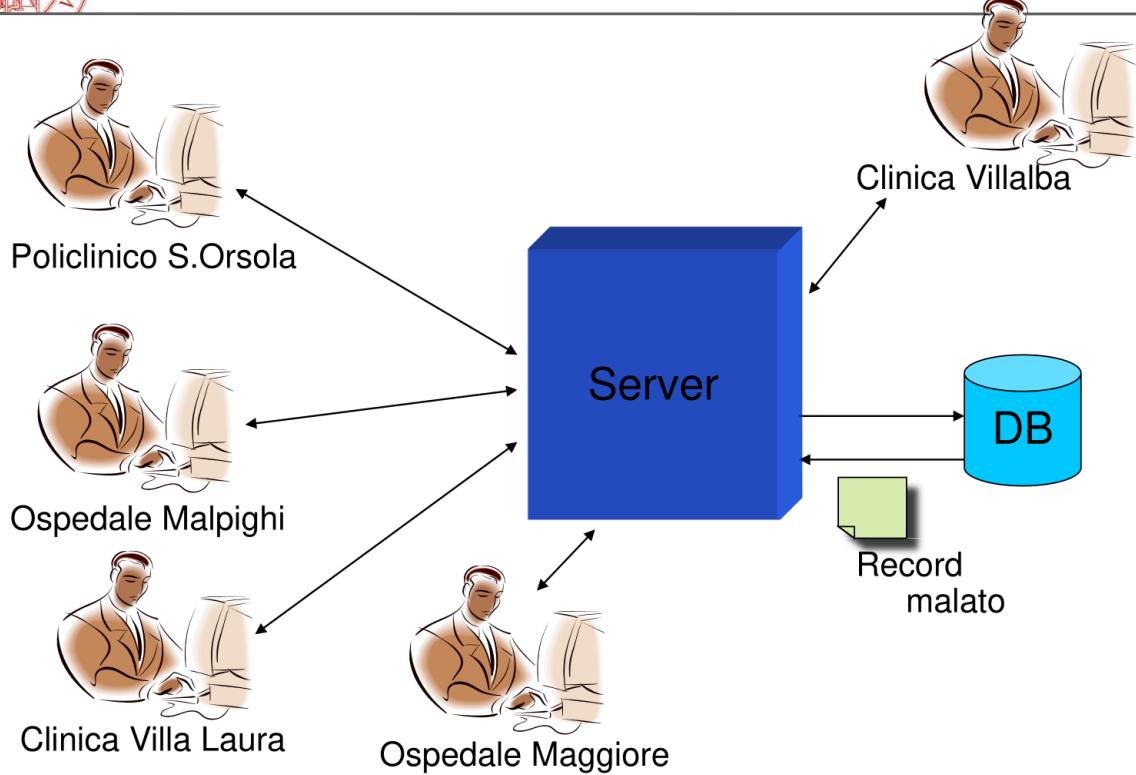


Figure 69: Schema logico

Termine	Descrizione
Beni	I record dei pazienti che ricevono o hanno ricevuto trattamenti sanitari; il database; il sistema informativo
Esposizione	Possibile perdita di futuri pazienti che non si fidano della clinica. Perdita finanziaria e perdita d'immagine
Vulnerabilità	Un sistema di password debole rende facile agli utenti la memorizzazione di password banali; lo username è uguale al nome del dipendente
Attacchi	Furto di identità di un utente autorizzato e successiva violazione e sottrazione di dati; denial of service
Minacce	Possibilità di indovinare le password di utenti autorizzati; arrivo contemporaneo di un numero elevato di richieste
Controllo	Sistema di controllo delle password che obblighi gli utenti a utilizzare password di tipo strong; replicazione del servizio su più server

### 2.1.3.7 Analisi del Rischio

- L'analisi del rischio si occupa di
  - valutare le possibili perdite che un attacco può causare ai beni di un sistema
  - bilanciare queste perdite con i costi richiesti per la protezione dei beni stessi

***costo protezione << costo della perdita***

- L'analisi del rischio è una problematica più manageriale che tecnica
- Il ruolo degli ingegneri della sicurezza è quindi quello di fornire una [guida tecnica e giuridica](#) sui problemi di sicurezza del sistema

- Sarà poi compito dei manager decidere se accettare i costi della sicurezza o i rischi che derivano dalla mancanza di procedure di sicurezza
- L'analisi del rischio inizia dalla valutazione delle politiche di sicurezza di organizzazione che spiegano cosa dovrebbe e cosa non dovrebbe essere consentito fare
- Le politiche di sicurezza propongono le condizioni che dovrebbero sempre essere mantenute dal sistema di sicurezza, quindi aiutano ad identificare le minacce che potrebbero sorgere
- La valutazione del rischio è un processo in più fasi:
  - **valutazione preliminare del rischio**: determina i requisiti di sicurezza dell'intero sistema
- **ciclo di vita della valutazione del rischio**: avviene contestualmente e segue il ciclo di vita dello sviluppo del software; valutare i rischi ogni volta che viene cambiato qualcosa

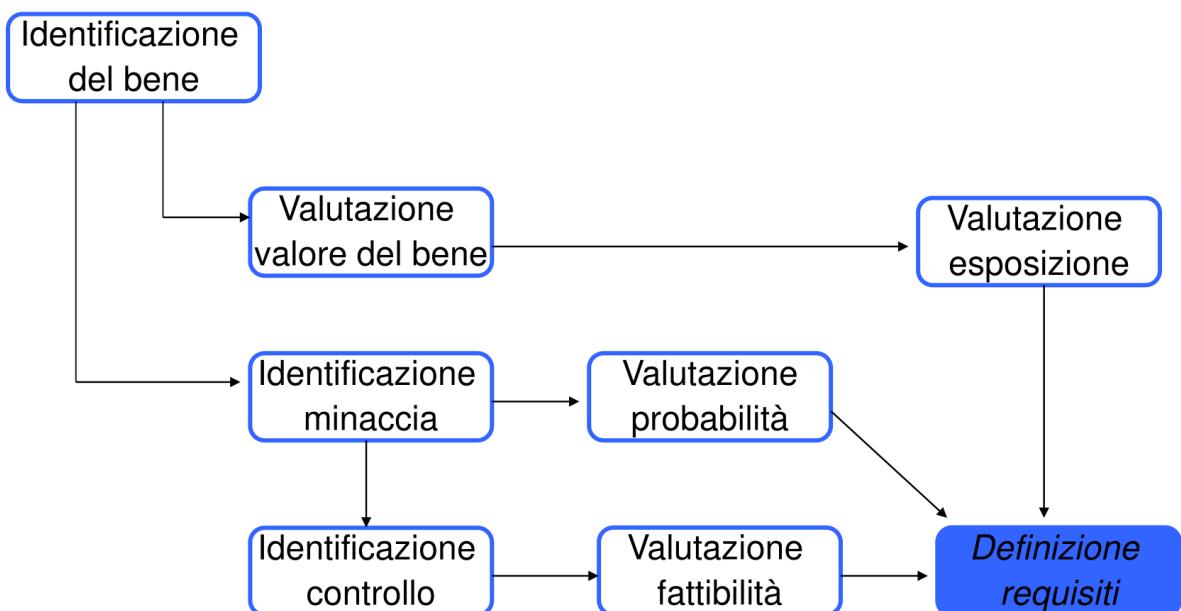


Figure 72: Valutazione Preliminare del Rischio

### 2.1.3.8 Identificazione del bene

#### 2.1.3.8.1 Analisi del Sistema Informatico

- Durante questa fase si può stabilire la seguente agenda delle attività:
  - Analisi delle risorse fisiche
  - Analisi delle risorse logiche
  - Analisi delle dipendenze fra risorse

#### 2.1.3.8.2 Analisi delle Risorse Fisiche

- In questa attività, il sistema informatico viene visto come insiemi di dispositivi che, per funzionare, hanno bisogno di spazio, alimentazione, condizioni ambientali adeguate, protezioni da furti e danni materiali
- In particolare occorre:
  - individuare sistematicamente tutte le risorse fisiche
  - ispezionare e valutare tutti i locali che ospiteranno le risorse fisiche
  - verificare la cablatura dei locali

### **2.1.3.8.3 Analisi delle Risorse Logiche**

- Il sistema viene visto come insieme di informazioni, flussi e processi
- In particolare occorre:
  - **Classificare le informazioni** in base al valore che hanno per l'organizzazione, il grado di riservatezza e il contesto di afferenza
  - **Classificare i servizi** offerti dal sistema informatico affinché non presentino effetti collaterali pericolosi per la sicurezza del sistema

### **2.1.3.8.4 Analisi delle Dipendenze tra Risorse**

- Per ciascuna risorsa (fisica o logica) occorre individuare di quali altre risorse essa ha bisogno per funzionare correttamente
- Questa analisi tende ad evidenziare le risorse potenzialmente critiche, ovvero quelle da cui dipende il funzionamento di un elevato numero di altre risorse
- I risultati di questa analisi sono usati anche nella fase di valutazione del rischio
  - in particolare, sono di supporto allo studio della propagazione dei malfunzionamenti a seguito dell'occorrenza di eventi indesiderati

### **2.1.3.9 Identificazione delle minacce**

- In questa fase si cerca di definire quello che **non deve poter accadere** nel sistema
- Si parte dal considerare come evento indesiderato qualsiasi accesso che non sia esplicitamente permesso
- A tal fine è possibile in generale distinguere tra
  - attacchi intenzionali
  - eventi accidentali

#### **2.1.3.9.1 Attacchi Intenzionali**

- Gli attacchi vengono caratterizzati in funzione della risorsa (sia fisica che logica) che viene attaccata e delle possibili tecniche usate per l'attacco
- Le tecniche di attacco possono essere classificate in funzione del livello al quale operano
- Si distingue tra **tecniche a livello fisico** e **a livello logico**
  - Gli attacchi a livello fisico sono principalmente tesi a sottrarre o danneggiare le risorse critiche
  - Si tratta di
    - Furto = un attacco alla disponibilità e alla riservatezza
    - Danneggiamento = un attacco alla disponibilità e alla integrità

#### **2.1.3.9.2 Attacchi a Livello Logico**

- Gli attacchi a livello logico sono principalmente tesi a sottrarre informazione o degradare l'operatività del sistema
- Dal punto di vista dei risultati che è indirizzato a conseguire un attacco logico può essere classificato come:
- **Intercettazione e deduzione** (attacco alla riservatezza): sniffing, spoofing, emulatori...
- **Intrusione** (attacco all'integrità e alla riservatezza): IP-spoofing, backdoor...

- Disturbo (attacco alla disponibilità): virus, worm, denial of service...

#### 2.1.3.9.3 Eventi Accidentali

- Una possibile casistica degli eventi accidentali che accadono più di frequente:
- **a livello fisico:**
  - guasti ai dispositivi che compongono il sistema
  - guasti di dispositivi di supporto (es. condizionatori)
- **a livello logico:**
  - perdita di password o chiave hardware
  - cancellazione di file
  - corruzione del software di sistema (ad esempio, a seguito dell'installazione di estensioni incompatibili)

#### 2.1.3.9.4 Valutazione dell'Esposizione

- A ogni minaccia occorre associare un rischio così da indirizzare l'attività di individuazione delle contromisure verso le aree più critiche
- Per **rischio** s'intende una combinazione della probabilità che un evento accada con il danno che l'evento può arrecare al sistema
- Nel valutare il danno si tiene conto
  - delle dipendenze tra le risorse
  - dell'eventuale propagazione del malfunzionamento

#### 2.1.3.9.5 Valutazione delle Probabilità: Attacchi Intenzionali

- La probabilità di occorrenza di attacchi intenzionali dipende principalmente dalla **facilità** di attuazione e dai **vantaggi** che potrebbe trarne l'intruso
  - Il danno si misura come **grado di perdita** dei tre requisiti fondamentali (riservatezza, integrità, disponibilità)
- MA l'attaccante applicherà sempre tutte le tecniche di cui dispone, su tutte le risorse attaccabili → necessità di valutare anche il rischio

di un **attacco composto**

- un insieme di attacchi elementari concepiti con un medesimo obiettivo e condotti in sequenza

#### 2.1.3.9.6 Individuazione del Controllo

- Occorre scegliere il controllo da adottare per neutralizzare gli attacchi individuati:
  - Valutazione del rapporto costo/efficacia
  - Analisi di standard e modelli di riferimento
  - Controllo di carattere organizzativo
  - Controllo di carattere tecnico

#### 2.1.3.9.7 Valutazione del Rapporto Costo/Efficienza

- Valuta il **grado di adeguatezza** di un controllo
- Mira ad evitare che i controlli presentino un costo ingiustificato rispetto al rischio dal quale proteggono
- **Efficienza del controllo** definita come funzione del rischio rispetto agli eventi indesiderati che neutralizza

- Il costo di un controllo deve essere calcolato senza dimenticare i [costi nascosti](#)

#### **2.1.3.9.8 Costi Nascosti**

- Occorre tenere presenti le limitazioni che i controlli impongono e le operazioni di controllo che introducono nel flusso di lavoro del sistema informatico e dell'organizzazione
- Le principali voci di costo sono le seguenti:
  - costo di messa in opera del controllo
  - peggioramento dell'ergonomia dell'interfaccia utente
  - decadimento delle prestazioni del sistema nell'erogazione dei servizi
  - aumento della burocrazia

#### **2.1.3.9.9 Controlli di Carattere Organizzativo**

- [Condizione essenziale](#) affinché la tecnologia a protezione del sistema informatico risulti efficace è che venga [utilizzata nel modo corretto](#) da personale pienamente [consapevole](#)
- Devono quindi essere definiti con precisione [ruoli e responsabilità](#) nella gestione sicura di tale sistema
- Per ciascun ruolo, dall'amministratore al semplice utente, devono essere definite [norme comportamentali e procedure precise](#) da rispettare

#### **2.1.3.9.10 Controlli di Carattere Tecnico**

- Controlli di base
  - a livello del sistema operativo e dei servizi di rete
- Controlli specifichi del particolare sistema
  - si attestano normalmente a livello applicativo
- Controlli tecnici più frequenti
  - configurazione sicura del sistema operativo di server e postazioni di lavoro (contromisura di base)
  - confinamento logico delle applicazioni server su server dedicati
- Etichettatura delle informazioni, allo scopo di avere un controllo più fine dei diritti di accesso
  - Moduli software di cifratura integrati con le applicazioni
  - Apparecchiature di telecomunicazione in grado di cifrare il traffico dati in modo trasparente alle applicazioni
  - Firewall e server proxy in corrispondenza di eventuali collegamenti con reti TCP/IP
  - Chiavi hardware e/o dispositivi di riconoscimento degli utenti basati su rilevamenti biofisici
- Integrazione dei Controlli
- Un insieme di controlli non deve presentarsi come una “collezione di espedienti” non correlati tra loro
- È importante integrare i vari controlli

in una politica di sicurezza organica

- Operare una selezione dei controlli adottando un sottoinsieme di costo minimo che rispetti alcuni vincoli:

- completezza delle contromisure
- omogeneità delle contromisure
- ridondanza controllata delle contromisure
- effettiva attuabilità delle contromisure

Vincoli del Sottoinsieme

- Completezza:

il sottoinsieme deve fare fronte a tutti gli eventi indesiderati

- Omogeneità:

le contromisure devono essere compatibili e integrabili tra loro

- Ridondanza controllata:

la ridondanza delle contromisure ha un costo e deve essere rilevata e vagliata accuratamente

- Effettiva attuabilità:

l'insieme delle contromisure deve rispettare tutti i vincoli imposti dall'organizzazione nella quale andrà ad operare

Esempio

- Torniamo all'esempio del sistema per la gestione di dati ospedalieri

- Quali sono i beni del sistema?
- il sistema informativo
- il database dei pazienti
- i record di ogni paziente

- Quali sono le minacce?

- furto identità dell'amministratore
- furto identità di utente autorizzato
- DoS

Esempio

Valutazione del Bene

Bene Valore Esposizione

Sistema

Informativo

Alto. Supporto a tutte le consultazioni cliniche

Alta. Perdita finanziaria; costi ripristino sistema; danni a pazienti se cure non date

Database pazienti

Alto. Supporto a tutte le consultazioni cliniche Critico dal punto di vista della sicurezza

Alta. Perdita finanziaria; costi ripristino sistema; danni a pazienti se cure non date  
Record paziente

Normalmente basso. Potrebbe essere alto per pazienti particolari

Perdita diretta bassa, ma possibile perdita di reputazione della clinica

Esempio

Analisi Minacce e Controlli

Minaccia Probabilità Controllo Fattibilità

Furto identità

Amministratore

Bassa Accesso solo da postazioni specifiche fisicamente sicure

Basso costo implementativo ma attenzione alla distribuzione delle chiavi

Furto identità utente

Alta Meccanismi biometrici Molto costoso e non accettato Log di tutte le operazioni

Semplice, trasparente e supporta il ripristino

DoS Media Progettazione adeguata, controllo e limitazione degli accessi

Basso costo. Impossibile prevedere e impedire questo tipo di attacco

Esempio

Requisiti di Sicurezza

- Alcuni dei requisiti ricavati dalla valutazione preliminare dei rischi
- le informazioni relative ai pazienti devono essere scaricate all'inizio della sessione clinica dal database e memorizzate in un'area sicura sul client
- le informazioni relative ai pazienti non devono essere mantenute sul client dopo la chiusura della sessione clinica
- deve essere mantenuto un log su una macchina diversa dal server
- che memorizzi tutti i cambiamenti effettuati sul database

Bene Valore Esposizione Sistema Informativo Alto. Supporto a tutta la gestione del villaggio turistico

Alta. Perdita finanziaria e di immagine Informazioni relative agli Ospiti

Medio. Dati generali degli ospiti del villaggio turistico

Media. Perdita di immagine se vengono divulgati dati degli Ospiti

Informazioni relative alle GuestCard

Alto. L'elenco degli acquisti è associato alle GuestCard

Molto Alta. Perdita finanziaria nel caso gli Ospiticontestino acquisti ingiustamente addebitati. Perdita di immagine Informazioni relative alle vendite

Alto. Sulla base dei movimenti nei Punti di Vendita, la Catena Punti Vendita gestisce i magazzini e i dati fiscali

Alta. Perdita finanziaria se i dati delle vendite e delle forniture non coincidono.  
Perdita di immagine se il servizio che vuole essere acquistato per qualche motivo non è presente Informazioni relative al personale

Alto. Ci sono tutte le informazioni relative al personale, comprese le credenziali di accesso

Alta. Perdita finanziaria, se vengono rubate le credenziali del personale si possono perpetuare svariate frodi. Perdita di immagine

Villaggio Turistico

Valutazione del Bene Villaggio Turistico

Analisi Minacce e Controlli Minaccia Probabilità Controllo Fattibilità Furto credenziali Operatore

Alta. La username è fissata e facile da identificare

Accesso da macchine sicure

Basso costo di realizzazione ma attenzione se le macchine vengono lasciate incustodite Log delle Operazioni Basso costo implementativo

Furto credenziali personale dei punti di vendita

Alta. La username è fissata e facile da identificare

Accesso da macchine sicure

Basso costo di realizzazione ma attenzione se le macchine vengono lasciate incustodite Log delle Operazioni Basso costo implementativo

Intercettazione comunicazioni

Alta. Il sistema è distribuito e client/server avvengono tantissime interazioni tra i diversi client e il server.

Cifratura delle comunicazioni

Il costo dipende dal tipo di cifratura scelto. Se simmetrica basso, se asimmetrica più alto dovuto alla necessità di rilascio di coppie di chiavi da un ente di certificazione  
Log di tutte le operazioni

Basso costo implementativo

DoS Bassa Progettazione adeguata, controllo e limitazione degli accessi

Basso costo. Impossibile prevenire un DoS

Ciclo di vita della valutazione

del rischio

## Ciclo di Vita della Valutazione del Rischio

- È necessaria la conoscenza dell'architettura del sistema
- e dell'organizzazione dei dati
- La piattaforma e il middleware sono già stati scelti,
- così come la strategia di sviluppo del sistema
- Questo significa che si hanno molti più dettagli
- riguardo a che cosa è necessario proteggere
- e sulle possibili vulnerabilità del sistema
- Le vulnerabilità possono essere “ereditate”
- dalle scelte di progettazione ma non solo
- La valutazione del rischio dovrebbe essere parte
- di tutto il ciclo di vita del software: dall'ingegnerizzazione
- dei requisiti al deployment del sistema

## Ciclo di Vita della Valutazione del Rischio

- Il processo seguito è simile a quello della valutazione
- preliminare dei rischi, con l'aggiunta di attività riguardanti
- l'identificazione e la valutazione delle vulnerabilità
- La valutazione delle vulnerabilità identifica i beni
- che hanno più probabilità di essere colpiti
- da tali vulnerabilità
- Vengono messe in relazione le vulnerabilità
- con i possibili attacchi al sistema
- Il risultato della valutazione del rischio è
- un insieme di decisioni ingegneristiche
- che influenzano la progettazione o l'implementazione
- del sistema o limitano il modo in cui esso è usato

## Esempio

- Si supponga che il provider dei servizi ospedalieri decida di acquistare un prodotto commerciale per la gestione dei dati dei pazienti
- Questo sistema deve essere configurato per ogni tipo di clinica in cui è utilizzato
  - Scelte progettuali del sistema acquistato:
  - autenticazione solo con username e password
  - architettura client-server: il client accede attraverso un'interfaccia web standard
  - l'informazione è presentata agli utenti
  - attraverso una web form editabile,
  - è quindi possibile modificare le informazioni

## Esempio: Vulnerabilità

Autenticazione

Login/password

Tecnologia Vulnerabilità

Password banali

Utente rivela password

Architettura

client-server

Denial of Service

Informazioni nella cache

Uso web form editabili

Log dettagliati non possibili

Stessi permessi per gli utenti

Bachi nel browser

Esempio

- Valutate le vulnerabilità del sistema adottato

si deve decidere quali mosse attuare al fine di limitare i rischi associati

- Introduzione di nuovi requisiti di sicurezza
- Introduzione di un meccanismo di verifica delle password
- l'accesso al sistema deve essere permesso
- solo ai client approvati e registrati dall'amministratore
- tutti i client devono avere un solo browser installato
- e approvato dall'amministratore

Villaggio Turistico

Vulnerabilità

Tecnologia Vulnerabilità

Autenticazione username/password

- Password banali
- Utente rivela volontariamente password
- Utente rivela password a seguito di un attacco di
- Ingegneria Sociale

Cifratura comunicazioni Le vulnerabilità dipendono dal tipo di cifratura. Cifratura Simmetrica:

- Tempo di vita della chiave. Più informazioni cifro
- con la stessa chiave più materiale offre per l'analisi
- del testo ad un attaccante
- Lunghezza della chiave
- Memorizzazione della chiave
- Cifratura Asimmetrica:
- Memorizzazione chiave privata

Architettura Client/Server • DoS

- Man in the Middle
- Sniffing delle comunicazioni

## Security Use Case e Misuse Case Security Use Case e Misuse Case

- I misuse case si concentrano sulle interazioni tra l'applicazione e gli attaccanti che cercano di violarla
- La condizione di successo di un misuse case è l'attacco andato a buon fine
- Questo li rende particolarmente adatti per analizzare le minacce, ma non molto utili per la determinazione dei requisiti di sicurezza
- È invece compito dei security use case specificare i requisiti tramite i quali l'applicazione dovrebbe essere in grado di proteggersi dalle minacce

Security Use Cases -Donald G. Firesmith-JOT Vol. 2, No. 3, May-June 2003

## Security Use Case e Misuse Case

### MisuseCase Security Use Case

Uso Analizzare e specificare le minacce

Analizzare e specificare i requisiti di sicurezza

Criteri di successo Successo attaccanti Successo applicazione

Prodotto da Security Team Security Team

Usato da Security Team RequirementsTeam

Attori Esterni Attaccanti, Utenti Utenti

Guidato da Analisi vulnerabilità dei beni e analisi minacce

Misusecase

## Security Use Case e Misuse Case Esempio

### INTRODUCTION

VOL. 2 , NO. 3 JOURNAL OF OBJECT TECHNOLOGY 55

The following table summarizes the primary differences between misuse cases and security use cases.

Misuse Cases Security Use Cases Usage Analyze<sup>^</sup> and<sup>^</sup> specify<sup>^</sup> security<sup>^</sup> threats.

Analyze and specify security requirements Success Criteria Misuser<sup>^</sup> Succeeds<sup>^</sup> Application<sup>^</sup> Succeeds<sup>^</sup> Produced By Security<sup>^</sup> Team<sup>^</sup> Security<sup>^</sup> Team<sup>^</sup> Used By Security<sup>^</sup> Team<sup>^</sup> Requirements<sup>^</sup> Team<sup>^</sup> External Actors Misuser,<sup>^</sup> User<sup>^</sup> User<sup>^</sup> Driven By Asset<sup>^</sup> Vulnerability<sup>^</sup> Analysis<sup>^</sup> Threat Analysis

## Misuse Cases

To further illustrate the differences between normal use cases, security use cases, and associated misuse cases, consider Figure 3. The traditional use cases for an automated teller machine might include Deposit Funds, Withdraw Funds, Transfer Funds, and Query Balance, all of which are specializations of a general Manage Accounts use

case. To securely manage one's accounts, one can specify security use cases to control access (identification, authentication, and authorization), ensure privacy (of data and communications), ensure integrity (of data and communications), and ensure nonrepudiation of transactions. The resulting four security use cases specify requirements that protect the ATM and its users from three security threats involving attacks by either crackers or thieves.

Customer Manage Accounts

Deposit Funds

Withdraw Funds

Transfer Funds

Query Balance

Control Access (Security)

Ensure Privacy (Security)

Ensure Integrity (Security)

Ensure Nonrepudiation (Security)

Spoof User (Misuse)

Invade Privacy (Misuse)

Perpetrate Fraud (Misuse)

Cracker

Thief

Misuse Case Misuser

Security Use Case

Fig. 3 : Example Security Use Cases and Misuse Cases Security Use Cases -Donald G. Firesmith-JOT Vol. 2, No. 3, May-June 2003 Esempio: Scenario

Security Use Cases -Donald G. Firesmith-JOT Vol. 2, No. 3, May-June 2003 Villaggio Turistico

Punto Vendita

CatenaPuntiVendita

ElencoVendite

Registrazione

NuovoMovimento

Login

ChiusuraCredito

GestioneOspite

ElencoAcquisti FineSettimana<>

FineVacanza<>

Operatore

OspitePagante

<>

<>

<>

AperturaCredito <>

<><>

<>

Gestione Villaggio Turistico

Garantire Protezione (security) Controllo Accesso (security)

(misuse)Frode SniffingInformazioni(misuse) FurtoCredenziali(misuse)

Truffatore Hacker

Villaggio Turistico

Titolo ControlloAccesso Descrizione Gli accessi al sistema devono essere controllati

Misuse case SniffingInformazioni,FurtoCredenziali Relazioni Precondizioni

L'Attaccante ha immezzato per scoprire la password del user name degli operatori e del personale dei punti vendita

Postcondizioni

Il Sistema blocca momentaneamente l'accesso all'utente in notifica un tentativo di accesso fraudolento  
Scenario principale

Sistema Attaccante Dopo aver scoperto qualche username tenta di accedere al sistema inserendo password con un attacco condizionario

Controlla le credenziali immesse e blocca l'accesso nel caso tali

credenziali risultino errate dopo un certo numero di tentativi. Scenario di attacco avvenuto con successo

Sistema Attaccante Attacco condizionato riuscito

Il Sistema controlla le credenziali immesse e consente l'accesso al sistema Naviga tra le maschere del sistema e cerca di capire più informazioni possibili

Il Sistema scrive nel log tutte le operazioni seguite dall'utente Il Sistema controlla periodicamente il log alla ricerca di pattern di accesso atipici e se rileva una notifica un accesso fraudolento

Security Use Case: Linee Guida

- I casi d'uso non dovrebbero mai specificare meccanismi

di sicurezza

- Le decisioni relative ai meccanismi devono essere lasciate alla progettazione
- Requisiti attentamente differenziati

dalle informazioni secondarie

- interazioni del sistema, azioni del sistema e post-condizioni
- sono i soli requisiti

- Evitare di specificare vincoli progettuali non necessari
  - Documentare esplicitamente i percorsi individuali attraverso i casi d'uso al fine di specificare i reali requisiti di sicurezza
  - Basare i security use case su differenti tipi di requisiti di sicurezza
- fornisce una naturale organizzazione dei casi d'uso Security Use Case: Linee Guida
- Documentare le minacce alla sicurezza che giustificano i percorsi individuali attraverso i casi d'uso
  - Distinguere chiaramente tra interazioni degli utenti e degli attaccanti
  - Distinguere chiaramente tra le interazioni che sono visibili esternamente e le azioni nascoste del sistema
  - Documentare sia le precondizioni che le post-condizioni che catturano l'essenza dei percorsi individuali SPECIFICA DEI REQUISITI DI SICUREZZA

#### Requisiti di Sicurezza

- Non è sempre possibile specificare i requisiti associati alla sicurezza in modo quantitativo
- Quasi sempre questa tipologia dei requisiti è espressa nella forma “non deve”
- definisce comportamenti inaccettabili per il sistema
- non definisce funzionalità richieste al sistema
- L'approccio convenzionale della specifica dei requisiti è basato sul contesto, sui beni da proteggere e sul loro valore per l'organizzazione

#### Categorie Requisiti di Sicurezza

- Requisiti di identificazione
- specificano se un sistema deve identificare gli utenti
- prima di interagire con loro
- Requisiti di autenticazione
- specificano come identificare gli utenti
- Requisiti di autorizzazione
- specificano i privilegi e i permessi di accesso
- degli utenti identificati
- Requisiti di immunità
- specificano come il sistema deve proteggersi da virus, worm
- e minacce simili

#### Categorie Requisiti di Sicurezza

- Requisiti di integrità
- specificano come evitare la corruzione dei dati

- Requisiti di scoperta delle intrusioni
- specificano quali meccanismi utilizzare per scoprire gli attacchi al Sistema
- Requisiti di non-ripudiazione
- specificano che una parte interessata in una transazione non può negare il proprio coinvolgimento
- Requisiti di riservatezza
- specificano come deve essere mantenuta la riservatezza delle informazioni

#### Categorie Requisiti di Sicurezza

- Requisiti di controllo della protezione
- specificano come può essere controllato e verificato l'uso del sistema
- Requisiti di protezione della manutenzione del sistema
- specificano come una applicazione può evitare modifiche autorizzate da un accidentale annullamento dei meccanismi di protezione

#### Villaggio Turistico

##### Requisiti di Sicurezza

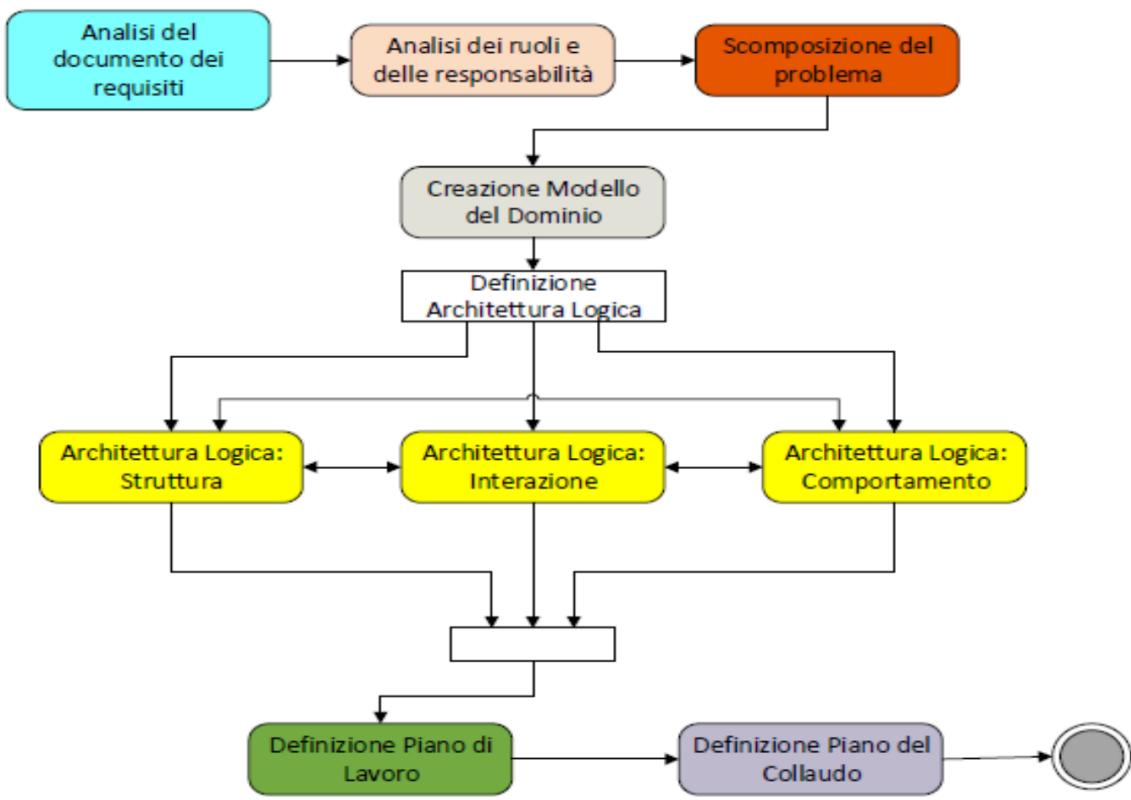
- Creazione di un log per tracciare tutte le azioni che avvengono sul sistema
- i messaggi scambiati tra le parti del sistema
- - che vanno protetti in un qualche modo per evitare che un accesso fraudolento al sistema di log possa rivelare dati riservati
- Adottare meccanismi di analisi del log per identificare pattern di accesso atipici
- identificare discrepanze tra i messaggi spediti e ricevuti
- Individuare una corretta politica di controllo degli accessi
- I dati memorizzati e scambiati nel sistema

devono essere protetti

## 2.2 Analisi del problema

### 2.2.1 Introduzione

- **Obiettivo:** esprimere fatti il più possibile “oggettivi” sul **problema** focalizzando l’attenzione su **sottosistemi, ruoli e responsabilità** insiti negli scenari prospettati durante l’analisi dei requisiti senza descrivere la sua possibile soluzione
- **Risultato:**
  - Architettura Logica
  - Piano di Lavoro
  - Piano del Collaudo (in che modo collauderemo il sistema)



## 2.2.2 Passi dell'Analisi del Problema

### 1. Analisi del Documento dei Requisiti

Obiettivo: concentrarsi sull'analisi delle funzionalità e dei rischi evidenziati nel documento dei requisiti

### 2. Analisi dei Ruoli e delle Responsabilità

Obiettivo: analizzare bene i ruoli emersi nei casi d'uso e porre particolare attenzione nell'attribuzione delle responsabilità a tali ruoli tenendo conto dell'analisi del rischio

### 3. Scomposizione del problema

Obiettivo: se possibile suddividere il problema in sotto-problemi più piccoli

### 4. Creazione del Modello del Dominio

Obiettivo: partendo dal vocabolario si costruisce il diagramma delle classi del dominio; il modello di dominio è il modello delle entità che sono importanti nel dominio applicativo e definisce le relazioni tra tali entità

### 5. Architettura Logica: Struttura

Obiettivo: creazione dei diagrammi strutturali (package e classi) dell'Architettura Logica

### 6. Architettura Logica: Interazione

Obiettivo: creazione dei diagrammi di interazione (diagrammi di sequenza) dell'Architettura Logica; specificano le funzionalità

### 7. Architettura Logica: Comportamento

Obiettivo: creazione dei diagrammi di comportamento (stato e attività) dell'Architettura Logica, se è opportuno che ci siano

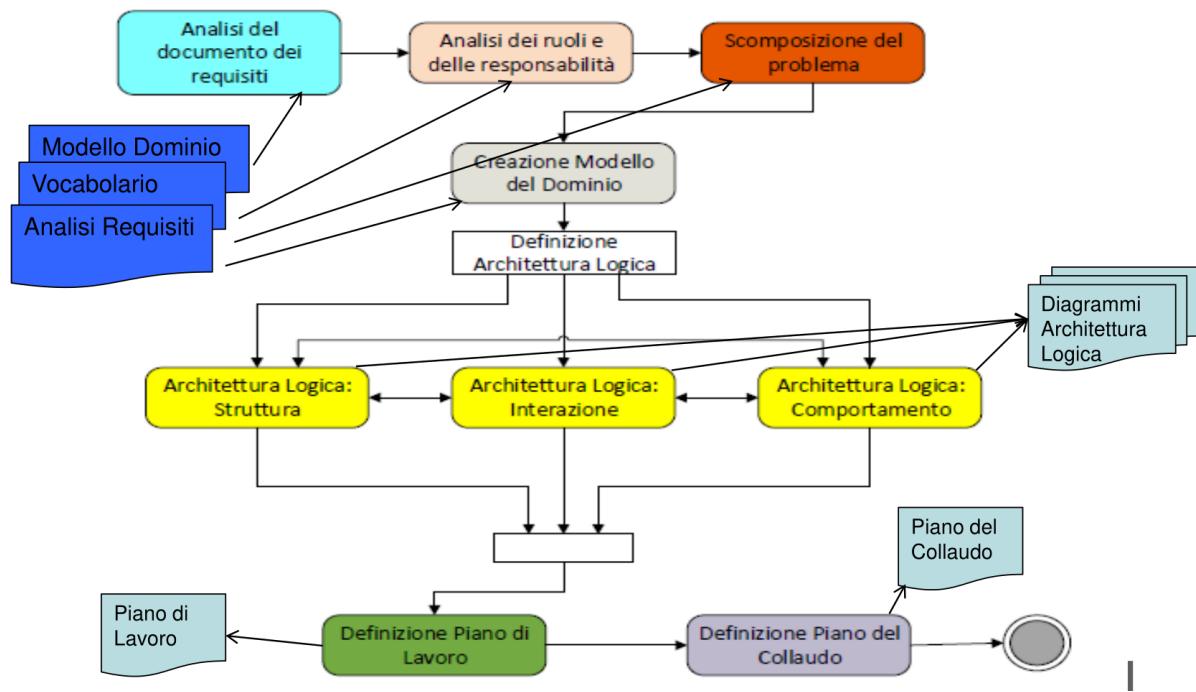
### 8. Definizione del Piano di Lavoro

Obiettivo: assegnare le responsabilità a ciascun membro del team di progetto, stabilire i vincoli temporali, impostare eventuali milestone, ...

### 9. Definizione del Piano del Collaudo

Obiettivo: definire i risultati attesi da ogni entità (classe, sottosistema) che

compare nell'Architettura Logica; scrivere le classi di test per verificare tali risultati



### 2.2.2.1 Architettura Logica

- L'**Architettura Logica** è un insieme di modelli UML costruiti per definire una struttura concettuale del problema robusta e modificabile
- Attraverso l'Architettura Logica l'analista descrive
  - **la struttura** (insieme delle parti)
  - **il comportamento atteso** (da tutto e da ciascuna parte)
  - **le interazioni**

così come possono essere dedotte dai requisiti, dalle caratteristiche del problema e del dominio applicativo **senza alcun riferimento alle tecnologie realizzative**

### 2.2.2.2 Piano di Lavoro

- Il **Piano di Lavoro** esprime l'articolazione delle attività in termini di risorse
  - umane
  - temporali
  - di elaborazione
  - ...

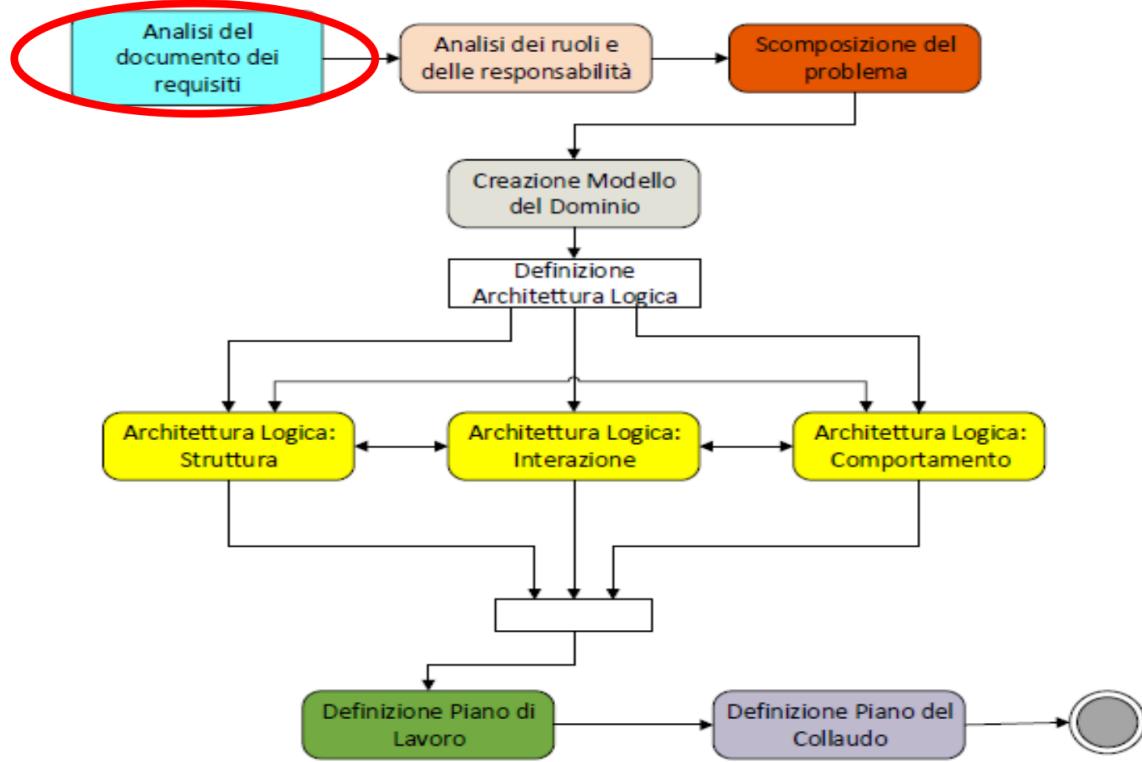
necessarie allo sviluppo del prodotto in base ai risultati dell'Analisi del Problema

### 2.2.2.3 Piano del Collaudo

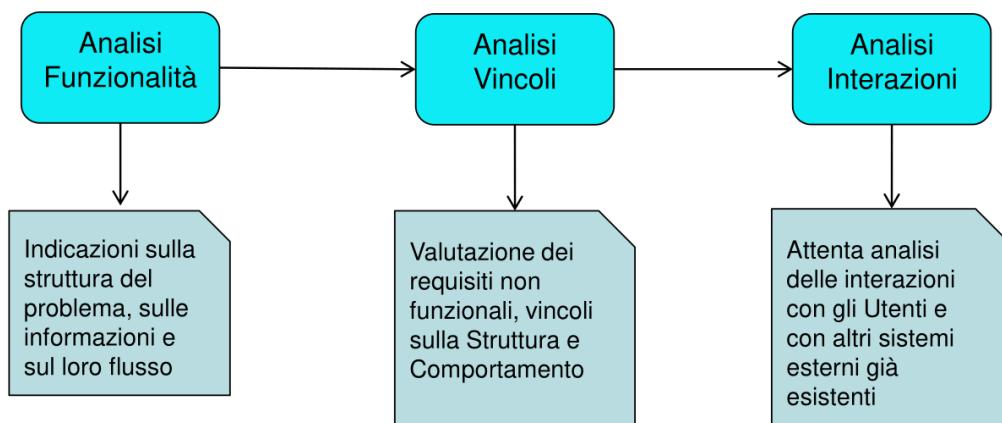
- Il **Piano del Collaudo** definisce l'insieme dei risultati attesi da ogni entità definita nell'Architettura Logica in relazione a specifiche sollecitazioni stimolo-risposta prevista
- Un modo per impostare il Piano del Collaudo è quello di fornire i test delle classi definite nell'Architettura Logica, così da pianificare già in questa fase i test di integrazione dei vari sottosistemi
- Ciò agevola il lavoro della successiva fase di progetto, riducendo il rischio di brutte sorprese in fase di integrazione delle sotto-parti
- Strumenti tipici per scrivere i test-case:
  - JUnit nel mondo Java

- ▶ NUnit(<http://nunit.org/>) nel mondo C#

## 2.2.3 Analisi Documento dei Requisiti



- Il Documento dei Requisiti evidenzia
  - le **funzionalità** e i **servizi** che dovranno essere sviluppati (requisiti funzionali)
  - i **vincoli** che dobbiamo tenere in considerazione (requisiti non funzionali)
  - il “**mondo esterno**” con cui si dovrà interagire (attori e sistemi esterni)
  - i “**rischi**” legati a possibili attacchi alla protezione, integrità e privacy dei dati (requisiti di sicurezza)
- Partendo da tale documento occorre applicare un’attenta analisi delle singole funzionalità, vincoli, interazioni e rischi



### 2.2.3.1 Analisi delle Funzionalità

- Per ogni funzionalità espressa nei Casi d’Uso vanno analizzati:
  - **il tipo, ovvero l’obiettivo della funzionalità** : memorizzazione dei dati, interazione con l’esterno, gestione/manipolazione dei dati
    - etichettiamo la funzionalità in modo da rendere evidente durante la creazione dell’Architettura Logica dove “posizionarla”
    - se la funzionalità è molto complessa potrebbe appartenere a più tipi differenti:marcare la funzionalità come “complessa”

- ▶ **le informazioni coinvolte**: analisi sistematica di tutte le informazioni sulle quali deve “operare” la funzionalità
  - In particolare individuare il tipo di dato (composto/singolo) e il grado di riservatezza richiesto
- ▶ **il “flusso delle informazioni”**: quali sono le informazioni che
  - si ricevono in input: permette di dare indicazioni sulla validazione dell’input e se sono presenti vincoli sui valori ammessi
  - vanno prodotte in output: permette di valutare dall’esterno se la funzionalità si comporta come ci si aspetta

### 2.2.3.2 Analisi delle Funzionalità

- Si possono predisporre diverse tabelle di analisi

Tabella Funzionalità

Funzionalità	Tipo	Grado Complessità	Requisiti Collegati
nome funzionalità come compare nei casi d’uso	tipo della funzionalità	indicare se complessa o semplice	Requisiti a cui è collegata la funzionalità

Tabella Informazioni / Flusso

Informazione	Tipo	Livello di riservatezza/privacy	Input/Output	Vincoli
nome (o id)	composto/ semplice	grado di riservatezza richiesto	specificare se input o output	eventuali vincoli sui valori attesi

Se il grado di complessità è alto (Funzionalità complessa), allora bisognerà scomporla in sotto-funzionalità.

- La Tabella delle Funzionalità è una sola tabella per tutte le funzionalità
- La Tabella delle Informazioni/Flusso è una tabella per ogni funzionalità; è importante specificare ogni funzionalità, anche se prima si sono tralasciate funzionalità semplici

### 2.2.3.3 Esempio

- Tabella Funzionalità per il Villaggio Turistico

Funzionalità	Tipo	Grado Complessità	Requisiti Collegati
GestioneOspite	Memorizzazione dati, gestione dati	complessa	R1F, R2F, R3F, R4F, R5F, R7F, R8F, R10F, R12F, R13F
Login	Interazione esterno, gestione dati	semplice	R14F, R15F
NuovoMovimento	Memorizzazione dati, gestione dati	semplice	R11F
ElencoVendite	Gestione dati	semplice	R6F
ScritturaLog	Memorizzazione dati	semplice	R16F
AnalisiLog	Gestione dati	semplice	R17F

- Tabella Informazioni/Flusso per NuovoMovimento

Informazione	Tipo	Livello protezione / privacy	Input/output	Vincoli
Data	semplice	Protezione media	Input	Non più di 10 caratteri
Ora	semplice	Protezione media	Input	Non più di 5 caratteri
Tipo di acquisto	semplice	Protezione alta	Input	Non più di 400 caratteri
Importo addebitato	semplice	Protezione alta	Input	Non più di 30 caratteri
Identificativo GuestCard	semplice	Protezione molto alta	Input	Non più di 20 caratteri
Identificativo Punto Vendita	semplice	Protezione molto alta	Input	Non più di 20 caratteri
Identificativo Catena Punti di Vendita	semplice	Protezione molto alta	Input	Non più di 20 caratteri

#### 2.2.3.4 Analisi dei Vincoli

- Per ogni requisito non funzionale vanno analizzati:
  - **il tipo, ovvero a quale categoria appartiene:** vincoli sulle performance, sui tempi di risposta, sulla scalabilità, sull'usabilità, sulla protezione dei dati, etc...
    - etichettiamo il requisito in modo da rendere evidente la categoria
    - alcuni requisiti potrebbero influenzare diversi aspetti del sistema, nel caso indicarli tutti, insieme anche al tipo di impatto (es: portano ad un peggioramento)
  - **le funzionalità coinvolte:** indicare le funzionalità che vengono influenzate dal requisito
- Si può predisporre una tabella

Tabella Vincoli

Requisito	Categorie	Impatto	Funzionalità
Requisito non funzionale	tipo	indicare il tipo di impatto	funzionalità coinvolte



Una sola tabella per tutti i vincoli

### 2.2.3.5 Esempio

- Tabella Vincoli per il Villaggio Turistico

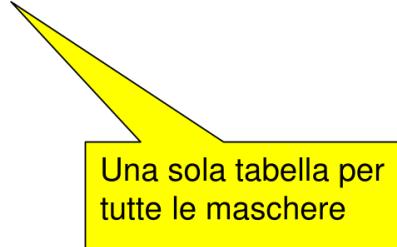
Requisito	Categorie	Impatto	Funzionalità
Velocità ricerca dati (R1NF)	Tempo di risposta	Cercare di migliorare	GestioneOspite, NuovoMovimento, ElencoVendite, Login
Velocità memorizzazione dati (R3NF)	Tempo di risposta	Cercare di migliorare	GestioneOspite, NuovoMovimento, ElencoVendite
Facile navigabilità delle schermate (R2NF)	Usabilità	Cercare di migliorare	GestioneOspite, NuovoMovimento, ElencoVendite, Login
Protezione dei dati (R3NF, R4NF, R6NF, R7NF, R8NF, R12NF)	Sicurezza	Peggiorano tempo di risposta, migliorano la privacy dei dati	GestioneOspite, NuovoMovimento, ElencoVendite, Login
Controllo Accessi (R5NF, R8NF, R9NF, R11NF)	Sicurezza	Peggiorano tempo di risposta e usabilità, migliorano la privacy dei dati	GestioneOspite, NuovoMovimento, ElencoVendite, Login

### 2.2.3.6 Analisi delle Interazioni

- Vanno distinte le interazioni con gli **umani** da quelle con **sistemi esterni**
- Nel caso delle interazioni con gli umani
  - analizzare le eventuali interfacce identificate con il cliente nella fase di Analisi dei Requisiti, oppure delineare le possibili interfacce
  - individuare le maschere di inserimento/output dati
  - individuare le sole informazioni necessarie da mostrare in ogni maschera
  - creare un legame tra maschere - informazioni - funzionalità
- Si può predisporre una tabella

Tabella Maschere

Maschera	Informazioni	Funzionalità
Nome della maschera	indicare le informazioni gestite nella maschera	funzionalità coinvolte



Una sola tabella per tutte le maschere

### 2.2.3.7 Esempio

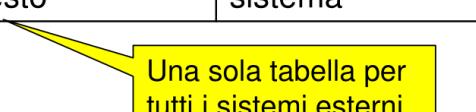
- Tabella Maschere per il Villaggio Turistico

Maschera	Informazioni	Funzionalità
Home Gestione	Messaggio benvenuto e scelta funzionalità	GestioneOspite
View Registrazione	Nome Ospite, Cognome Ospite, DataNascita, Indirizzo di residenza, Numero di telefono, Estremi del documento di identificazione, numero della stanza	GestioneOspite
View Apertura Credito	GuestCard, Identificativo OspitePagante	GestioneOspite
View Chiusura Credito	GuestCard, Identificativo OspitePagante, saldo	GestioneOspite
View Elenco Acquisti	Elenco dei Movimenti	GestioneOspite
Home PuntoVendita	Messaggio benvenuto e scelta funzionalità	NuovoMovimento
View Inserimento	Data, Ora, Tipo di acquisto, importo addebitato, GuestCard	NuovoMovimento
Home Catena PuntiVendita	Elenco dei report di vendita per ogni PuntoDiVendita	ElencoVendite
Home Log	Scelta del tipo di analisi o di visione di tutto il log di una parte del sistema	AnalisiLog
View Log	Data, Ora, Operazione eseguita, Messaggio	AnalisiLog
View Anomalie	Elenco delle anomalie	AnalisiLog
View Login	Username, password	Login

- Nel caso delle interazioni con sistemi esterni
  - analizzare ai morsetti i sistemi esterni con cui si dovrà interagire
  - individuare i protocolli di interazione con tali sistemi

Tabella Sistemi Esterni

Sistema	Descrizione	Protocollo di Interazione	Livello di Protezione
Nome del sistema	Descrizione del sistema e delle sue principali funzionalità	Specificare il protocollo di interazione richiesto	Specificare il livello di protezione garantito dal sistema



Una sola tabella per tutti i sistemi esterni

### 2.2.3.8 Esempio

- Tabella Sistemi Esterni per il Villaggio Turistico

Sistema	Descrizione	Protocollo di Interazione	Livello di Sicurezza
Gestione Personale (R16F/R9NF)	Sistema che si occupa della gestione del personale che lavora presso il Villaggio Turistico.	GestionePersonale mette a disposizione una funzionalità di controllo delle credenziali. Le credenziali devono essere inviate in modo sicuro e come risultato si ha una stringa che rappresenta il nome del ruolo assegnato al dipendente	Alto livello di sicurezza perché protegge i dati personali e sensibili dei dipendenti

## 2.2.4 Analisi Ruoli e Responsabilità

- Per ogni Attore individuato nei Casi d’Uso specificare
  - le responsabilità - cosa deve fare in ogni funzionalità
  - specificare le **informazioni a cui può accedere** relativamente a ciascuna funzionalità in cui è coinvolto con relativa indicazione del **tipo di accesso**
  - le maschere che può visualizzare
  - il suo livello di riservatezza - quale livello di riservatezza è necessario avere per poter ricoprire quel ruolo
  - la numerosità attesa - il numero di persone che possono giocare quel ruolo

### 2.2.4.1 Analisi dei Ruoli e Responsabilità

- Si possono predisporre alcune tabelle

Tabella Ruoli

Ruolo	Responsabilità	Maschere	Riservatezza	Numerosità
Nome del ruolo	Indicare le responsabilità assegnate	Indicare le maschere che devono essere visualizzate	Livello di riservatezza necessaria	Indicare la numerosità massima

Tabella Ruolo-Informazioni

Informazione	Tipo Accesso
Informazione	Specificare il tipo di accesso

Come prima, una sola tabella per tutti i ruoli e una tabella per ogni ruolo.

Non ha senso specificare una numerosità potenzialmente infinita, perché il numero delle persone che interagiranno con il sistema è una informazione che influenza come il sistema verrà progettato. Se la numerosità cambia nel tempo, probabilmente servirà riprogettare il sistema. È possibile inserire come specifica di numerosità “si progetti il sistema in modo che possa supportare il maggior numero di utenti”.

### 2.2.4.2 Esempio

- Tabella Ruoli per il Villaggio Turistico

Ruolo	Responsabilità	Maschere	Riservatezza	Numerosità
Operatore	Gestione di tutte le informazioni relative agli Ospiti del villaggio turistico	Home Gestione, View Registrazione, View Apertura Credito, View Chiusura Credito, View Elenco Acquisti, View Login	È richiesto un alto grado di riservatezza	Massimo 10 Operatori, considerando l'alternanza dei turni di lavoro e dei giorni di risposto
..	...	...	...	...

### 2.2.4.3 Esempio

- Tabella Operatore-Informazioni per il Villaggio Turistico

Informazione	Tipo di Accesso
Nome Ospite	Lettura/scrittura
Cognome Ospite	Lettura/scrittura
DataNascita	Lettura/scrittura
Indirizzo di residenza	Lettura/scrittura
Numero di telefono	Lettura/scrittura
Estremi del documento di identificazione utilizzato	Lettura/scrittura
Numero della stanza	Lettura/scrittura
CartaCredito	Scrittura
Identificativo GuestCard	Lettura/scrittura
DataInizioSoggiorno	Lettura/scrittura
DataFineSoggiorno	Lettura/Scrittura
Valuta	Lettura/scrittura
Saldo	Lettura
Username	Scrittura
Password	Scrittura

### 2.2.5 Scomposizione del Problema

- Il punto di partenza è l'Analisi delle Funzionalità
- Per ogni funzionalità marcata come “complessa” nella Tabella delle Funzionalità occorre valutare se sia possibile operare una scomposizione
  - ▶ la funzionalità può essere suddivisa in sotto-funzionalità più semplici?
  - ▶ quale “legame” sussiste tra le sotto-funzionalità?
  - ▶ quali informazioni devono “fluire” tra le sotto-funzionalità
- Si possono predisporre alcune tabelle

Tabella Scomposizione Funzionalità

Funzionalità	Scomposizione
nome funzionalità	elenco delle sotto-funzionalità

Tabella Sotto-Funzionalità

Sotto-Funzionalità	Sotto-Funzionalità	Legame	Informazioni
nome sotto-funzionalità	nome sotto-funzionalità	specificare il tipo di dipendenza/legame logico	specificare le informazioni scambiate

La tabella Sotto-Funzionalità elenca le dipendenze tra funzionalità.

- Tabella Scomposizione Funzionalità per il Villaggio Turistico

Funzionalità	Scomposizione
GestioneOspite	Registrazione, AperturaCredito, ChiusuraCredito, ElencoAcquisti

- Tabella Sotto-Funzionalità per il Villaggio Turistico

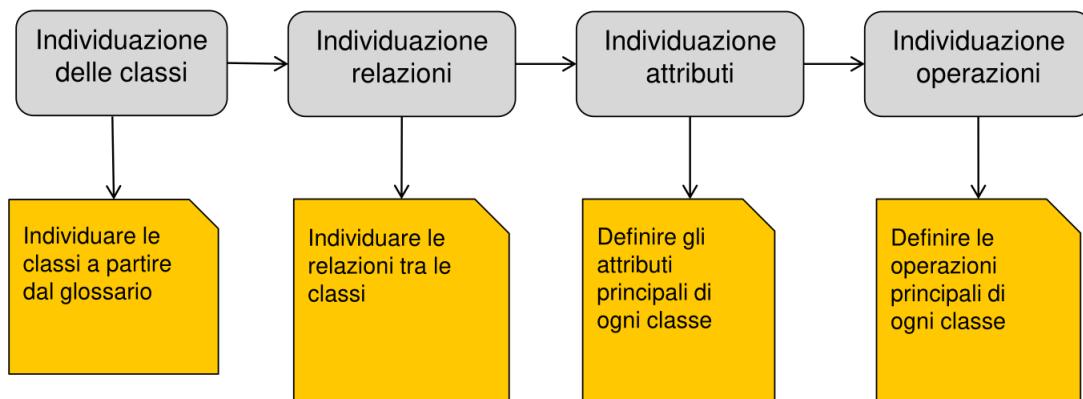
Sotto-funzionalità	Sotto-funzionalità	Legame	Informazioni
AperturaCredito	Registrazione	AperturaCredito dipende da Registrazione	Identificativo OspitePagante
ChiusuraCredito	AperturaCredito	Non si può chiudere un credito se non è mai stato aperto	Identificativo OspitePagante
ElencoAcquisti	AperturaCredito	Non si può mostrare l'elenco acquisti se non è stato aperto prima il credito	Identificativo OspitePagante

## 2.2.6 Creazione Modello del Dominio

- Il punto di partenza è costituito dall'insieme di:
  - [glossario](#) definito nell'Analisi dei Requisiti
  - tabelle informazioni/flusso
- Sulla base di questi si costruisce il diagramma delle classi che rappresenta il Modello del Dominio
- Tale modello poi sarà riusato nell'Architettura Logica come modello dei dati

- Occorre tenere presente che non tutti i vocaboli elencati nel glossario diventeranno classi, si devono infatti evitare:
  - ridondanze: classi uguali ma con diverso nome
  - costruzione di classi a partire da termini ambigui nel glossario
  - nomi che si riferiscono a eventi o operazioni
  - nomi appartenenti al meta-linguaggio del processo come, per esempio, requisiti e sistema
  - nomi al di fuori dell'ambito del sistema
  - nomi che possono essere attributi
- Individuare
  - **Oggetti e classi** rilevanti per il problema che si sta analizzando
    - Limitarsi esclusivamente a quelle classi che fanno parte del vocabolario del dominio del problema, non dell'applicazione in se, quindi evitare, ad esempio, di pensare a “che tipo di classe Java devo sviluppare”
  - **Relazioni tra le classi**
  - Per ogni classe
    - **Attributi**
    - **Operazioni fondamentali** cioè servizi forniti all'esterno

Pensare sempre al “*cosa*” non al “*come*”; il “*come*” non è l’oggetto di questa fase.



Questo schema non è per forza in successione, nel senso che se, mentre stiamo definendo le operazioni di una classe, mi accorgo che c’è bisogno di un attributo, è possibile aggiungerlo.

### 2.2.6.1 Individuazione delle Classi

- Dal glossario eliminare i nomi che sicuramente
  - non si riferiscono a classi
  - indicano attributi (dati di tipo primitivo)
  - indicano operazioni
- Scegliere un solo termine significativo se più parole indicano lo stesso concetto (sinonimi)
- Il nome della classe deve essere un nome familiare
  - all’utente o all’esperto del dominio del problema
  - non allo sviluppatore!
- **Attenzione agli aggettivi e agli attributi**, possono

- ▶ indicare oggetti diversi
- ▶ indicare usi diversi dello stesso oggetto
- ▶ essere irrilevanti
- Ad esempio:
  - ▶ “Studente bravo” potrebbe essere irrilevante
  - ▶ “Studente fuori corso” potrebbe essere una nuova classe
- **Attenzione alle frasi passive, impersonali o con soggetti fuori dal sistema:**
  - ▶ devono essere rese attive ed esplicite, perché potrebbero mascherare entità rilevanti per il sistema in esame
- Individuare **Attori** con cui il sistema in esame deve interagire
  - ▶ **Persone**: Docente, Studente, Esaminatore, Esaminando, ...
  - ▶ **Sistemi esterni**: ReteLocale, Internet, DBMS, ...
  - ▶ **Dispositivi**: attuatori, sensori, ...
- Individuare **modelli e loro elementi specifici**:
  - ▶ Insegnamento - “Ingegneria del Software T”
  - ▶ CorsoDiStudio - “Ingegneria Informatica”
  - ▶ Facoltà - “Ingegneria”
- Individuare **cose tangibili**, cioè oggetti reali appartenenti al dominio del problema
  - ▶ Banco, LavagnaLuminosa, Schermo, Computer, ...
- Individuare **contenitori** (fisici o logici) di altri oggetti
  - ▶ Facoltà, Dipartimento, Aula, SalaTerminali, ...
  - ▶ ListaEsame, CommissioneDiLaurea, OrdineDegliStudi, ...
- Individuare **eventi o transazioni** che il sistema deve gestire e memorizzare
  - ▶ possono avvenire in un certo istante (es., una vendita) o
  - ▶ possono durare un intervallo di tempo (es., un affitto)
  - ▶ Appello, EsameScritto, Registrazione, AppelloDiLaurea, ...
- Per determinare se includere una classe nel modello, porsi le seguenti domande:
  - ▶ il sistema deve interagire in qualche modo con gli oggetti della classe?
    - **utilizzare informazioni** (attributi) contenute negli oggetti della classe
    - **utilizzare servizi** (operazioni) offerti dagli oggetti della classe
  - ▶ quali sono le responsabilità della classe nel contesto del sistema?
- Attributi e operazioni devono essere applicabili a tutti gli oggetti della classe
- Se esistono
  - ▶ attributi con un valore ben definito solo per alcuni oggetti della classe
  - ▶ operazioni applicabili solo ad alcuni oggetti della classe

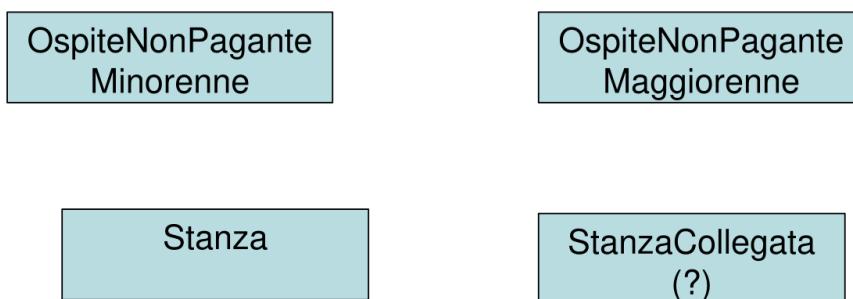
siamo in presenza di **ereditarietà**

- Esempio: dopo una prima analisi, la classe Studente potrebbe contenere un attributo booleano inCorso, ma un’analisi più attenta potrebbe portare alla luce la gerarchia:
  - ▶ Studente: StudenteInCorso, StudenteFuoriCorso

## 2.2.6.2 Individuazione delle Classi: Villaggio Turistico

Voce	Definizione	Sinonimi
Villaggio Turistico	Luogo dove si effettua una vacanza	
Ospite	Persona che è in vacanza nel Villaggio Turistico, senza fare distinzione se Pagante o non Pagante	Cliente
OspitePagante	Persona in vacanza nel villaggio turistico a cui sono associate una o più stanze. Persona che effettua il saldo del conto	Cliente
OspiteNonPagante	Persona in vacanza nel villaggio turistico a cui è associata una stanza. Non effettua pagamenti ed è sempre associato a un OspitePagante	
OspiteNonPagante Maggiorenne	Persona in vacanza nel villaggio turistico a cui è associata una stanza. Ha raggiunto la maggiore età. Gli viene consegnata una GuestCard, ma questa viene associata all'OspitePagante.	
OspiteNonPagante Minorenne	Persona in vacanza nel villaggio turistico a cui è associata una stanza. Non ha raggiunto la maggiore età.	
Stanza	Ambiente fisico in cui gli Ospiti dormono e tengono i loro beni. Ha un certo numero di posti disponibili	camera
StanzaCollegata	Ambiente fisico in cui gli Ospiti dormono e tengono i loro beni. È logicamente collegata ad un'altra stanza	camera

- Villaggio Turistico è in verde perché è una classe che non viene creata, dato che non serve; se, invece, avessimo avuto un progetto che prevedeva molteplici villaggi turistici, allora avrebbe avuto senso definire la classe VillaggioTuristico
- I nomi in rosso sono delle classi, tra l'altro con una probabile gerarchia (alcune)
- Forse Stanza e StanzaCollegata sono una gerarchia



### 2.2.6.3 Individuazione delle Relazioni

- La maggior parte delle classi (degli oggetti) **interagisce** con altre classi (altri oggetti) in vari modi
- In ogni tipo di relazione, esiste un cliente C che dipende da un fornitore di servizi F
  - C ha bisogno di F per lo svolgimento di alcune funzionalità che C non è in grado di effettuare autonomamente
  - Conseguenza per il corretto funzionamento di C è indispensabile il corretto funzionamento di F

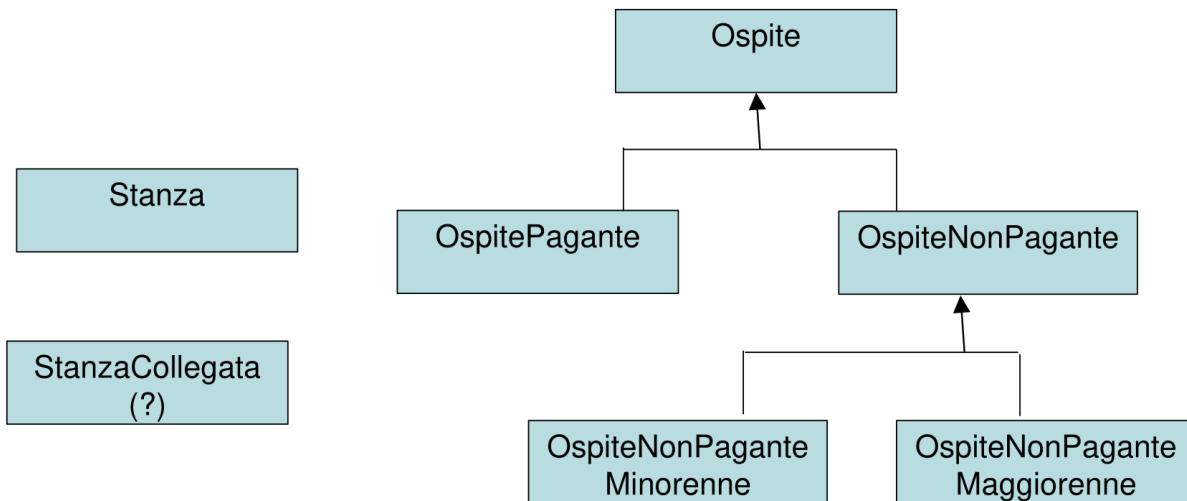
Tipo di relazione	Cliente	Fornitore
Ereditarietà	sottoclasse	superclasse
Associazione (composizione / aggregazione)	contenitore	contenuto
Dipendenza	classe dipendente (source)	classe da cui si dipende (target)

- Nell'associazione il fornitore è il contenuto e il cliente è il contenitore, perché il contenitore contenendo il contenuto può usare i servizi del contenuto

#### 2.2.6.4 Individuazione dell'Ereditarietà

- L'ereditarietà deve rispecchiare una tassonomia effettivamente presente nel dominio del problema
  - Non usare l'ereditarietà dell'implementazione (siamo ancora in fase di analisi!)
  - Non usare l'ereditarietà solo per riunire caratteristiche comuni
    - ad es., Studente e Dipartimento hanno entrambi un indirizzo, ma non per questo c'è ereditarietà!

#### 2.2.6.5 Ereditarietà: Villaggio Turistico



- Tutte le ereditarietà di questo caso sono complete e disgiunte; cioè non esistono ospiti che non siano paganti e non paganti e un ospite è o pagante o non pagante, non esistono ospiti non paganti che non siano minorenni e maggiorenni e un ospite non pagante è o minorenne o maggiorenne

#### 2.2.6.6 Individuazione delle Associazioni

- Un'associazione rappresenta una relazione strutturale tra due istanze di classi diverse o della stessa classe
- Un'associazione può
  - Rappresentare un contenimento logico (**aggregazione**)
    - Una lista d'esame contiene degli studenti
  - Rappresentare un contenimento fisico (**composizione**)
    - Un triangolo contiene tre vertici

- Non rappresentare un reale contenimento
  - Una fattura si riferisce a un cliente
  - Un evento è legato a un dispositivo

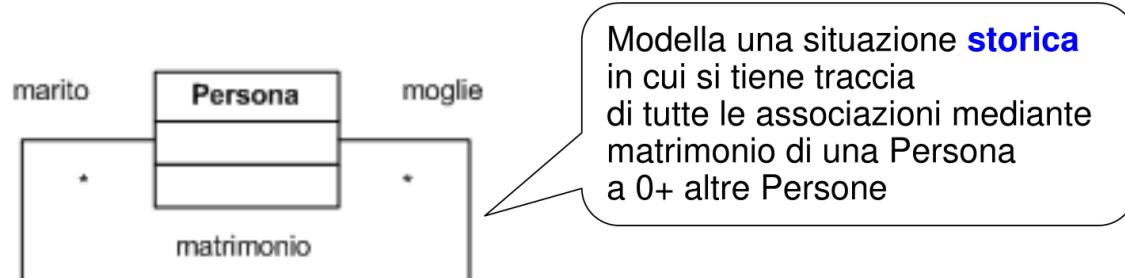
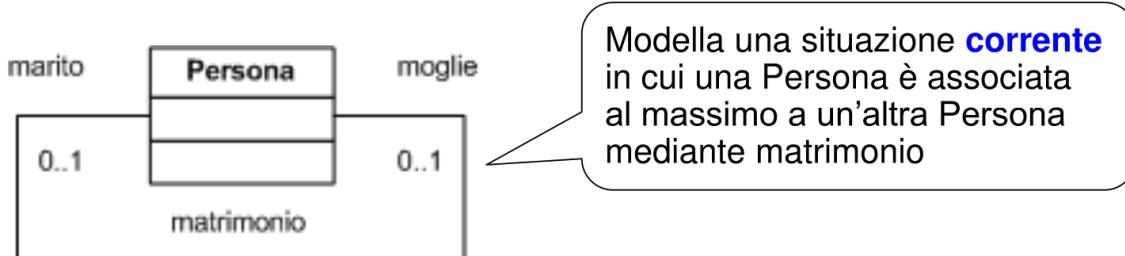
### 2.2.7 Individuazione delle Associazioni

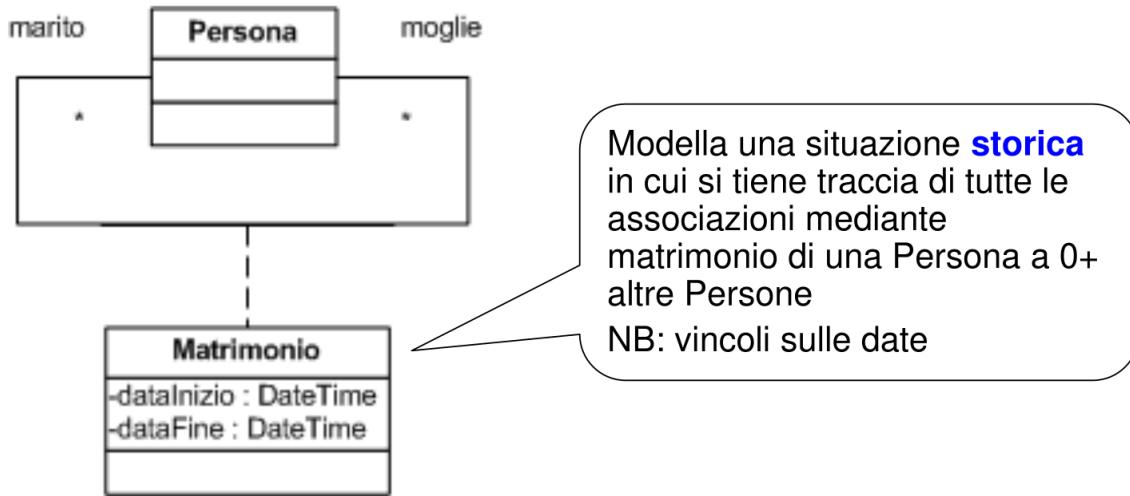
- **Aggregazione**

Un oggetto x di classe X è associato a (contiene) un oggetto y di classe Y in modo non esclusivo x può condividere y con altri oggetti anche di tipo diverso (che a loro volta possono contenere y)

- Una lista d'esame contiene degli studenti
  - Uno studente può essere contemporaneamente in più liste d'esame
  - La cancellazione della lista d'esame non comporta l'eliminazione "fisica" degli studenti in lista
- **Composizione**  
Un oggetto x di classe X è associato a (contiene) un oggetto y di classe Y **in modo esclusivo** y esiste solo in quanto contenuto in x
- Un triangolo contiene tre punti (i suoi vertici)
  - L'eliminazione del triangolo comporta l'eliminazione dei tre punti
- Se la distruzione del contenitore comporta la distruzione degli oggetti contenuti, si tratta di composizione, altrimenti si tratta di aggregazione
- Attenzione alle associazioni molti a molti possono nascondere una classe (classe di associazione) del tipo "evento da ricordare"
- Ad esempio,
  - la connessione "matrimonio" tra Persona e Persona può nascondere una classe Matrimonio, che lega due Persone
  - la connessione "possiede" tra Proprietario e Veicolo può nascondere una classe CompraVendita, che lega un Proprietario a un Veicolo

#### 2.2.7.1 1° Esempio di Associazione

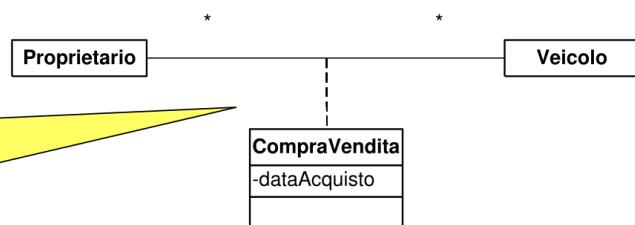
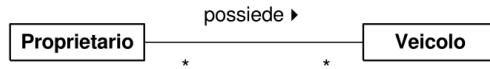




Quindi, quando l'associazione ha delle informazioni, o più raramente del comportamento, si introduce la classe d'associazione.

### 2.2.7.2 2 ° Esempio di Associazione

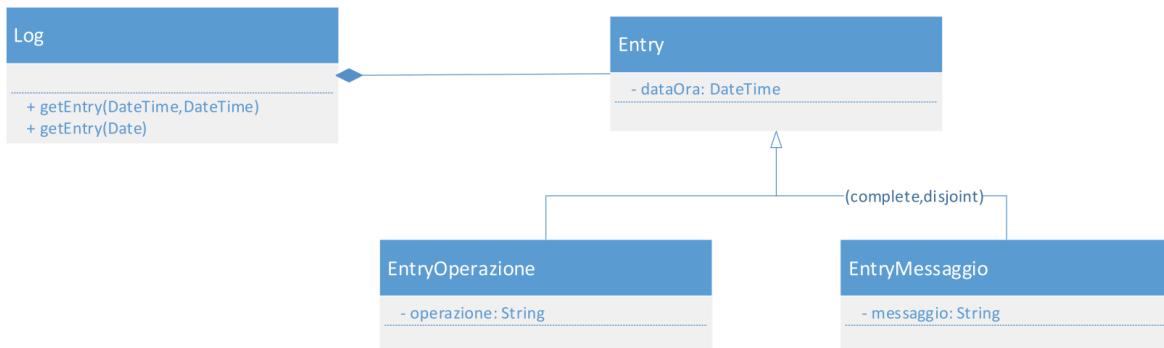
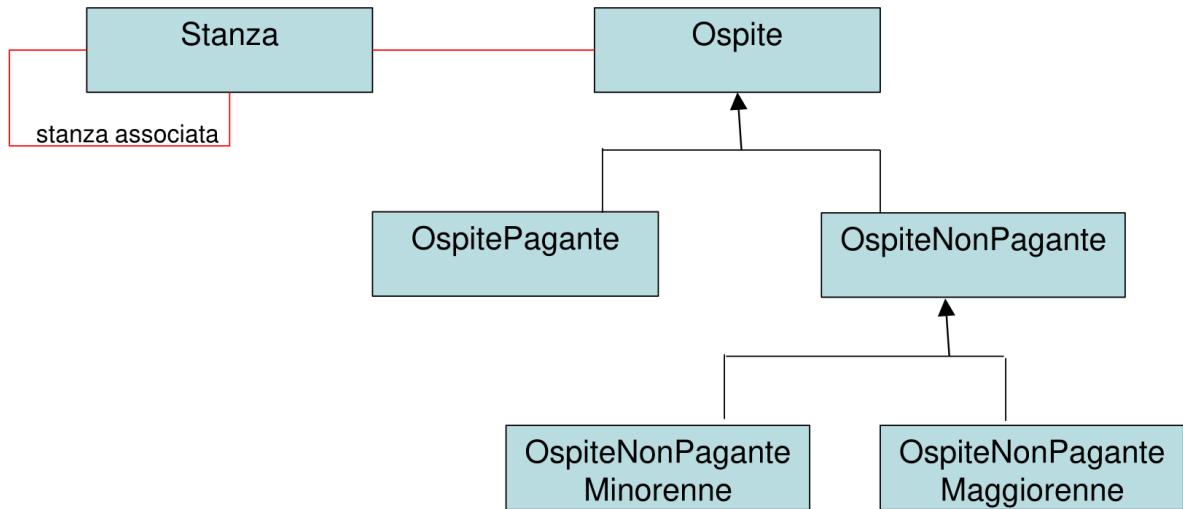
- Un proprietario può possedere molti veicoli
- Un veicolo può essere di molti proprietari
  - in tempi successivi
  - in comproprietà



Attenzione alle molteplicità: nel modello delle classi di analisi va bene, ma a livello di istanza si introduce il vincolo extra che ci può essere solo un'istanza della classe di associazione tra ogni coppia di oggetti associati

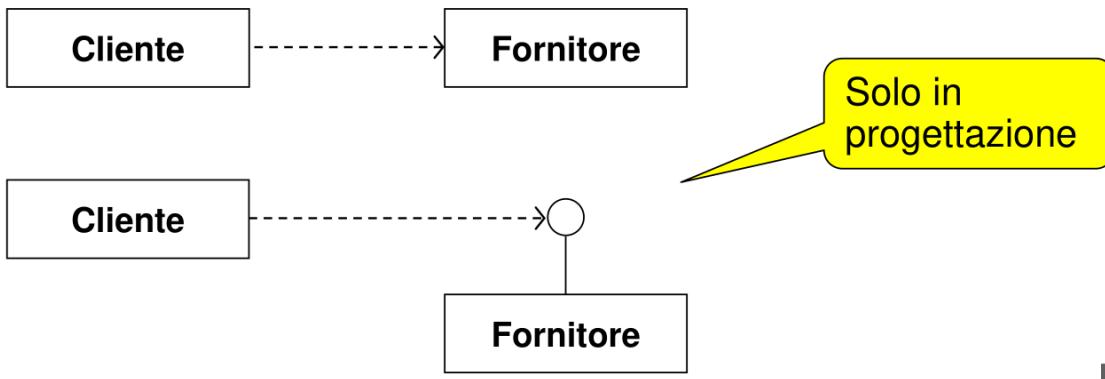
- Qui c'è un vincolo particolare, che è quello di tenere in conto del caso in cui una persona compra un veicolo da solo e, successivamente, vuole averlo in comproprietà
- Da notare che ci può essere solo un'istanza della classe di associazione per ogni coppia di oggetti associati

### 2.2.7.3 Associazioni: Villaggio Turistico



#### 2.2.7.4 Individuazione Collaborazioni

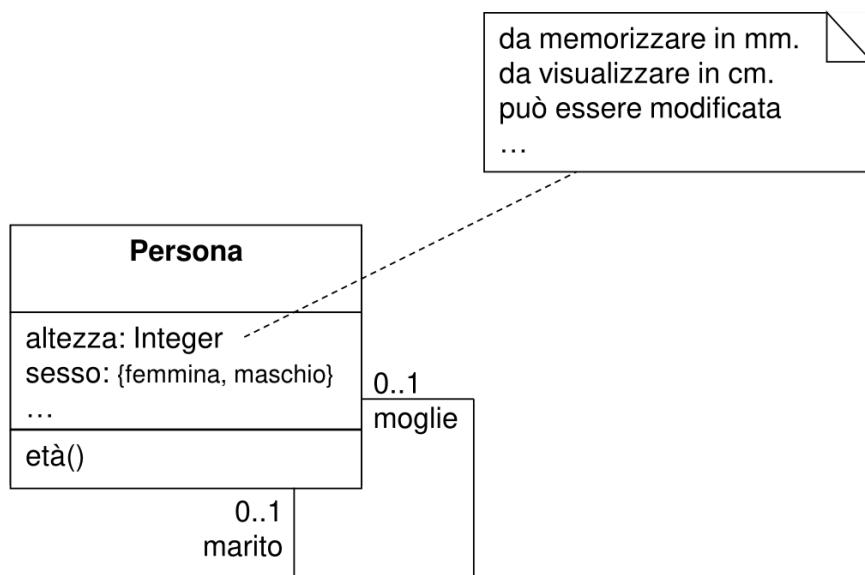
- Una classe A è in relazione USA con una classe B (A USA B) quando A ha bisogno della collaborazione di B per lo svolgimento di alcune funzionalità che A non è in grado di effettuare autonomamente
  - Un'operazione della classe A ha bisogno come argomento di un'istanza della classe B
    - `void fun1(B b) { ... usa b ... }`
  - Un'operazione della classe A restituisce un valore di tipo B
    - `B fun2(...) { B b; ... return b; }`
  - Un'operazione della classe A utilizza un'istanza della classe B (ma non esiste un'associazione tra le due classi)
    - `void fun3(...) { B b = new B(...); ... usa b ... }`
- La relazione non è simmetrica: A dipende da B, ma B non dipende da A
- Evitare situazioni in cui una classe, tramite una catena di relazioni USA, alla fine dipende da se stessa



### 2.2.7.5 Individuazione degli Attributi

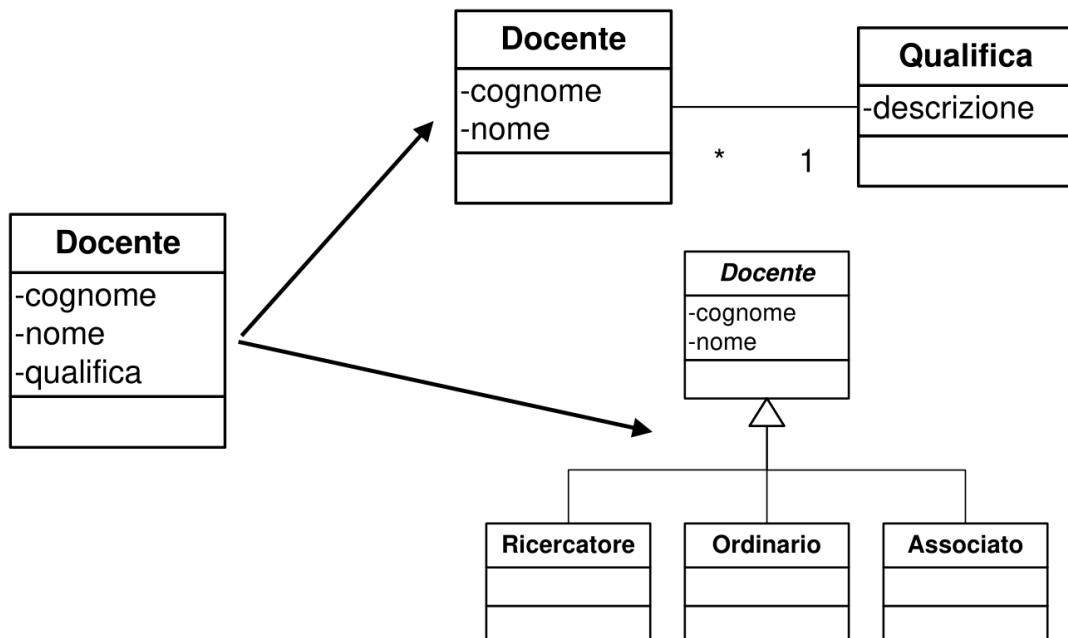
- Ogni attributo modella una proprietà atomica di una classe
  - Un valore singolo
    - Una descrizione, un importo, ..
  - Un gruppo di valori strettamente collegati tra loro
    - Un indirizzo, una data, ..
- Proprietà non atomiche di una classe devono essere modellate come associazioni
- A tempo di esecuzione, in un certo istante, ogni oggetto avrà un valore specifico per ogni attributo della classe di appartenenza: informazione di stato
- La base di partenza per la ricerca degli attributi sono le Tabelle di Informazione/ Flusso
- Il nome dell'attributo
  - deve essere un nome familiare
    - all'utente o all'esperto del dominio del problema
    - non allo sviluppatore!
  - non deve essere il nome di un valore ("qualifica" sì, "ricercatore" no)
  - deve iniziare con una lettera minuscola
- Esempi
  - cognome, dataDiNascita, annoDiImmatricolazione
- Esprimere tutti i vincoli applicabili all'attributo
  - Tipo (semplice o enumerativo)
  - Opzionalità
  - Valori ammessi
    - dominio, anti-dominio, univocità
  - Vincoli di creazione
    - valore iniziale di default, immodificabilità del valore (readonly), etc.
  - Vincoli dovuti ai valori di altri attributi
  - Unità di misura, precisione
- Esprimere tutti i vincoli applicabili all'attributo
  - **Visibilità** (opzionale in fase di analisi) Attenzione: gli attributi membro devono essere sempre privati!
  - Appartenenza alla classe (e non all'istanza) Attributi (e associazioni) possono essere di classe, cioè essere unici nella classe
    - numIstanze: int = 0

- I vincoli possono essere scritti
  - Utilizzando direttamente UML
  - Utilizzando Object Constraint Language (OCL)
  - Come testo in formato libero in un commento UML



- Attenzione: nel caso di attributi con valore booleano “vero o falso”, “sì o no”, il nome dell’attributo potrebbe essere uno dei valori di un’enumerazione
  - Ad esempio: tassabile (sì o no) potrebbe diventare `tassabile{ tassabile, esente, ridotto, ... }`
- Attenzione: attributi
  - con valore “non applicabile” o
  - con valore opzionale o
  - a valori multipli (enumerazioni)

possono nascondere ereditarietà o una nuova classe



- Attenzione: nel caso di attributi calcolabili (ad esempio, `età`), specificare sempre l’operazione di calcolo mai l’attributo

- se memorizzare oppure no un attributo calcolabile è una decisione progettuale, un compromesso tra tempo di calcolo e spazio di memoria
- Applicare l'ereditarietà:
  - Posizionare attributi e associazioni più generali più in alto possibile nella gerarchia
  - Posizionare attributi e associazioni specializzati più in basso

### 2.2.7.6 Individuazione degli Attributi: Villaggio Turistico

Informazione	Voce	Definizione
Nome Ospite	Stanza	Ambiente fisico in cui gli Ospiti dormono e tengono i loro bene. Ha un certo numero di posti disponibili
Cognome Ospite	StanzaCollegata	Ambiente fisico in cui gli Ospiti dormono e tengono i loro bene. È logicamente collegata ad un'altra stanza

Ospite

```

- nome: String
- cognome: String
- indirizzo: String
- telefono
- dataNascita: Date
- documentoID: String
- inizioSoggiorno: DateTime
- fineSoggiorno: DateTime
  
```

Informazione	Voce	Definizione
Nome Ospite	Stanza	Ambiente fisico in cui gli Ospiti dormono e tengono i loro bene. Ha un certo numero di posti disponibili
Cognome Ospite	StanzaCollegata	Ambiente fisico in cui gli Ospiti dormono e tengono i loro bene. È logicamente collegata ad un'altra stanza

Ospite

```

- nome: String
- cognome: String
- indirizzo: String
- telefono
- dataNascita: Date
- documentoID: String
- inizioSoggiorno: DateTime
- fineSoggiorno: DateTime
  
```

Stanza

```

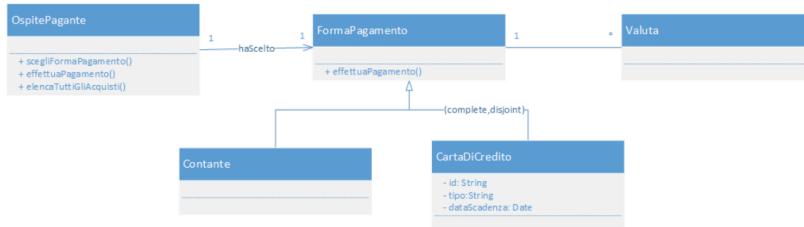
- numero: Integer
- tipo: String
- occupanti: Integer
  
```

- GuestCard una classe, l'identificativo sarà mappato con una associazione
- Saldo nasconde una nuova classe, associata alle funzionalità di Apertura e Chiusura Credito → L'ospite ha un conto aperto presso il Villaggio Turistico
- Movimento è relativo all'uso della GuestCard

Informazione
Nome Ospite
Cognome Ospite
DataNascita
Indirizzo di residenza
Numero di telefono
Estremi del documento di identificazione utilizzato
CartaCredito
Numeri della stanza
Data inizio soggiorno
Data fine soggiorno
Identificativo GuestCard
Movimento
Valuta
Saldo

Voce	Definizione
Stanza	Ambiente fisico in cui gli Ospiti dormono e tengono i loro bene. Ha un certo numero di posti disponibili
StanzaCollegata	Ambiente fisico in cui gli Ospiti dormono e tengono i loro bene. È logicamente collegata ad un'altra stanza

Carta di Credito e Valuta sono relative alle funzionalità di pagamento scelta e vanno modellate con delle classi



### 2.2.7.7 Individuazione delle Operazioni

- Il nome dell'operazione
  - deve appartenere al vocabolario standard del dominio del problema
  - potrebbe essere un verbo
    - all'imperativo (scrivi, esegui, ...) o
    - in terza persona (scrive, esegue, ...)
  - in modo consistente in tutto il sistema
- Operazioni standard
  - Operazioni che tutti gli oggetti hanno per il semplice fatto di esistere e di avere degli attributi e delle relazioni con altri oggetti (costruttore, accessori, ...)
  - Sono implicate e, di norma, non compaiono nel diagramma delle classi di analisi
- Altre operazioni
  - Devono essere determinate
    - servizi offerti agli altri oggetti
  - Compaiono nel diagramma delle classi di analisi
- Classi contenitori
  - Operazioni standard - aggiungi, rimuovi, conta, itera, ...
  - Altre operazioni - riguardano l'insieme degli oggetti, non il singolo oggetto
    - Calcoli da effettuare sugli oggetti contenuti Es: calcolaSulleParti(), totalizza()
    - Selezioni da fare sugli oggetti contenuti Es: trovaPartiSpecifiche()
    - Operazioni del tipo Es: eseguiUnAzioneSuTutteLeParti()
- Distribuire in modo bilanciato le operazioni nel sistema
- Mettere ogni operazione “vicino” ai dati a essa necessari
- Applicare l'ereditarietà
  - Posizionare le operazioni più generali più in alto possibile nella gerarchia
  - Posizionare le operazioni specializzate più in basso
- Descrivere tutti i vincoli applicabili all'operazione
  - Parametri formali, loro tipo e significato
  - Pre-condizioni, post-condizioni, invarianti di classe

- ▶ Eccezioni sollevate
- ▶ Eventi che attivano l'operazione
- ▶ Applicabilità dell'operazione
- ▶ ...
- PRE-CONDIZIONE Espressione logica riguardante le aspettative sullo stato del sistema prima che venga eseguita un'operazione
  - ▶ Esplicita in modo chiaro che è responsabilità della procedura chiamante controllare la correttezza degli argomenti passati
  - ▶ Ad esempio, per l'operazione **CalcolaRadiceQuadrata(valore)** , la pre-condizione potrebbe essere: “ **valore ≥ 0** ”
- POST-CONDIZIONE Espressione logica riguardante le aspettative sullo stato del sistema dopo l'esecuzione di un'operazione
  - ▶ Ad esempio, per l'operazione **CalcolaRadiceQuadrata(valore)** , la post-condizione potrebbe essere: **valore == risultato \* risultato**
- **INVARIANTE di classe**  
Vincolo di classe (espressione logica) che deve essere sempre verificato
  - ▶ sia all'inizio
  - ▶ sia alla fine

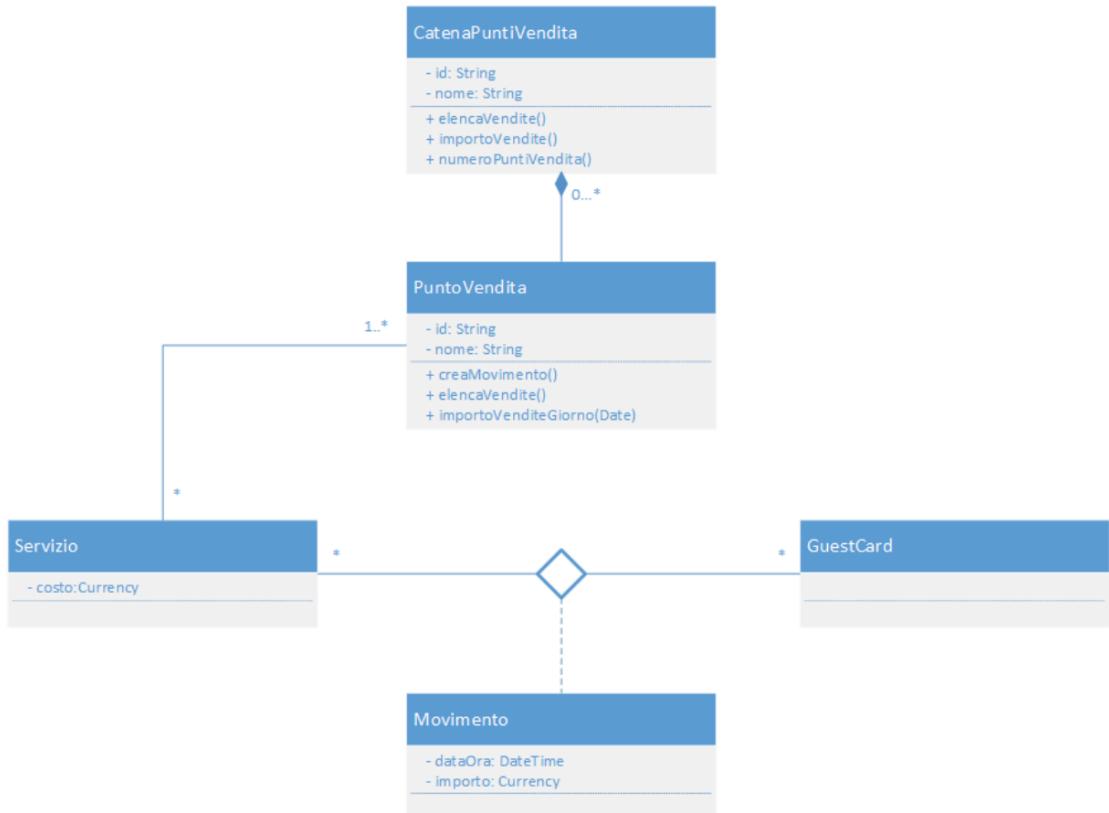
di tutte le operazioni pubbliche della classe Può non essere verificato solo durante l'esecuzione dell'operazione

#### 2.2.7.8 Individuazione delle Operazioni

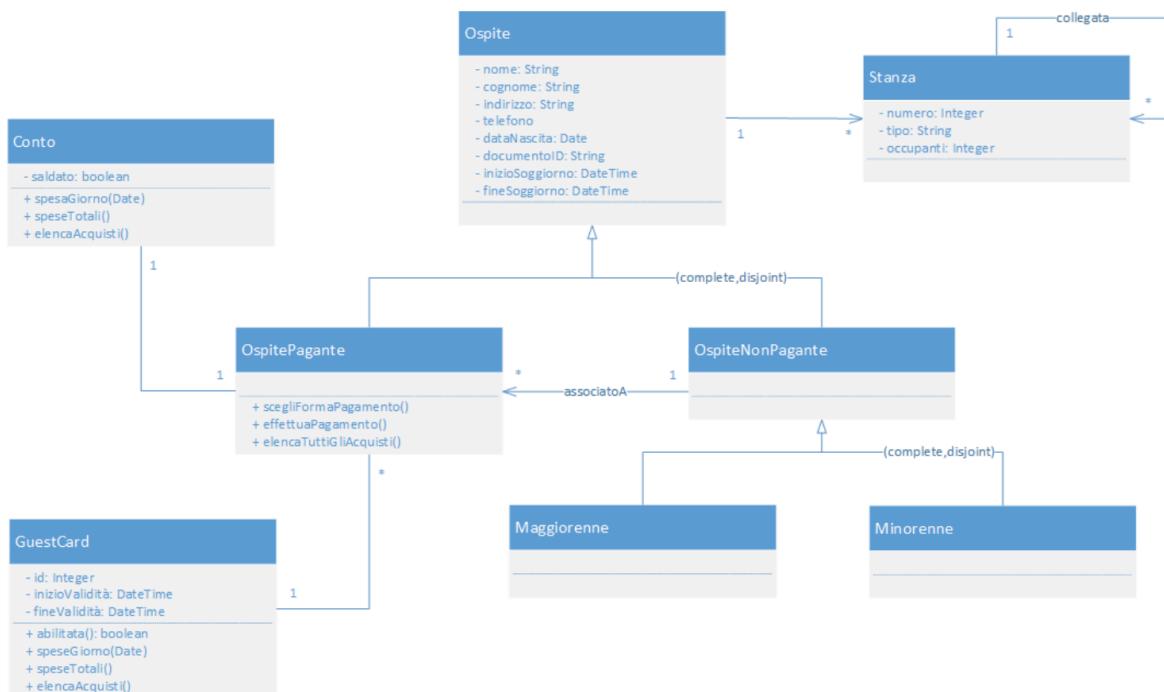
- ECCEZIONE Si verifica quando un'operazione
  - ▶ viene invocata nel rispetto delle sue pre-condizioni
  - ▶ ma non è in grado di terminare la propria esecuzione nel rispetto delle post-condizioni

#### 2.2.7.9 Esempio

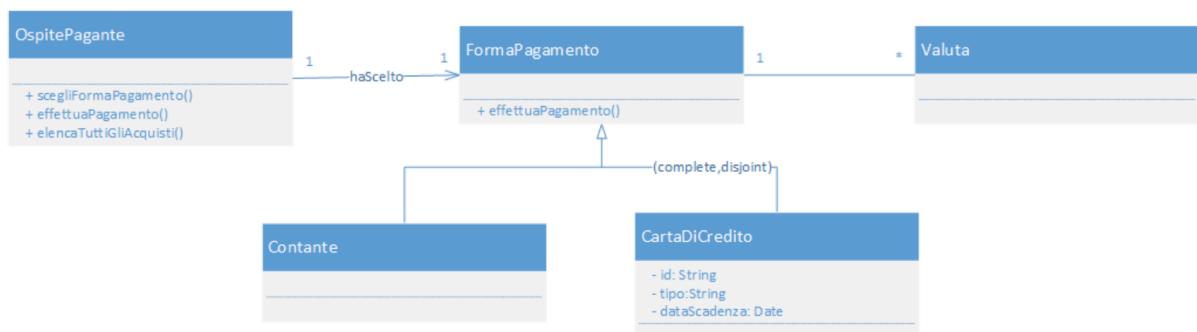
Modello del Dominio “Vendita Servizi” per il Villaggio Turistico



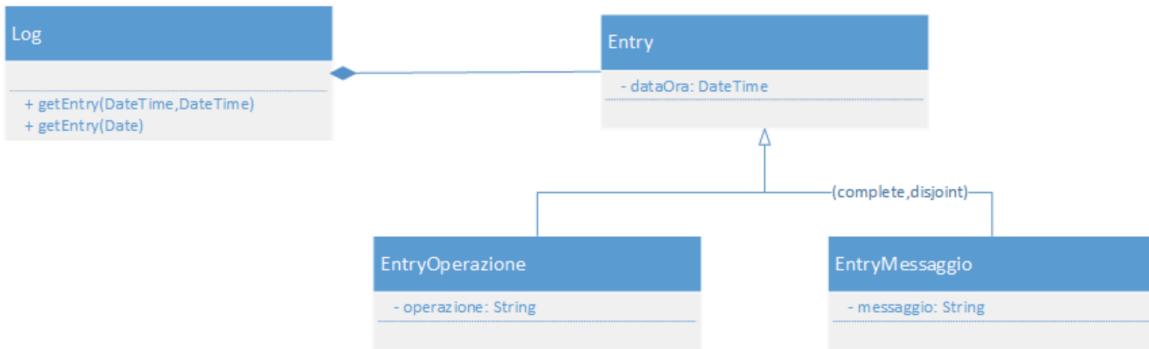
Modello del Dominio “Ospite” per il Villaggio Turistico



Modello del Dominio “Pagamenti” per il Villaggio Turistico



Modello del Dominio “Log” per il Villaggio Turistico



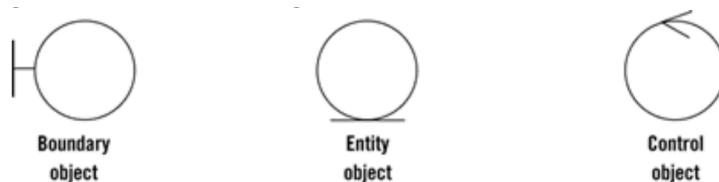
## 2.2.8 Architettura Logica: Struttura

**N.B.** Le parti “Struttura”, “Interazione” e “Comportamento” dell’architettura logica vanno sviluppate contestualmente (in contemporanea?).

- La parte strutturale dell’Architettura Logica dovrebbe essere composta di due tipi differenti di diagrammi UML
  - [Diagramma dei Package](#) che fornisce una visione di alto livello dell’architettura
  - [Diagramma delle classi](#) (uno o più diagrammi in base alla complessità) che fornisce una visione più dettagliata del contenuto dei singoli package
- Sarebbe opportuno organizzare sin da subito l’Architettura Logica usando un pattern architettonico chiamato **Boundary-Control-Entity** (BCE)

### 2.2.8.1 BCE

- BCE è un pattern architettonico che suggerisce di basare l’architettura di un sistema sulla partizione sistematica degli use case in oggetti di tre categorie:
  - **informazione**
  - **presentazione**
  - **controllo**



Noi useremo una convenzione più semplice → coloriamo package e classi in modo diverso

- A ciascuna di queste dimensioni corrisponde uno specifico insieme di classi
- Tale pattern è stato introdotto anche in RUP e sono state adottate icone ben particolari
- BCE è un pattern architettonico che suggerisce di basare l’architettura di un sistema sulla partizione sistematica degli use case in oggetti di tre categorie:
  - **informazione**
  - **presentazione**
  - **controllo**

- **Entity:** è la dimensione relativa alle entità cui corrisponde l'insieme delle classi che includono funzionalità relative alle informazioni che caratterizzano il problema; sostanzialmente ci dice quali sono le entità di interesse nel dominio del problema
  - costituiscono gran parte del modello del dominio
- **Boundary:** è la dimensione relativa alle funzionalità che dipendono dall'ambiente esterno cui corrisponde l'insieme delle classi che incapsulano l'interfaccia del sistema verso il mondo esterno
- **Control:** è la dimensione relativa agli enti che incapsulano il controllo
  - il loro compito è di fare da *collante* tra le interfacce e le entità
- Impostare l'architettura di un sistema software distinguendo tra *boundary*, *control* ed *entity* costituisce un solido punto di partenza per l'organizzazione dell'Architettura Logica di molte applicazioni
- L'analista tende ad affrontare la complessità dei problemi partizionando i problemi stessi in sotto-problemi
- È del tutto logico che un analista eviti di associare alle entità di un dominio
  - sia le funzionalità di una specifica applicazione
  - sia le funzionalità tipiche della interazione con l'utente
- La conseguenza è che l'architettura di un sistema software risulta quasi fisiologicamente articolata in una sequenza di livelli (*layer*) verticali che viene tipicamente mantenuta anche in fase di progetto e implementazione

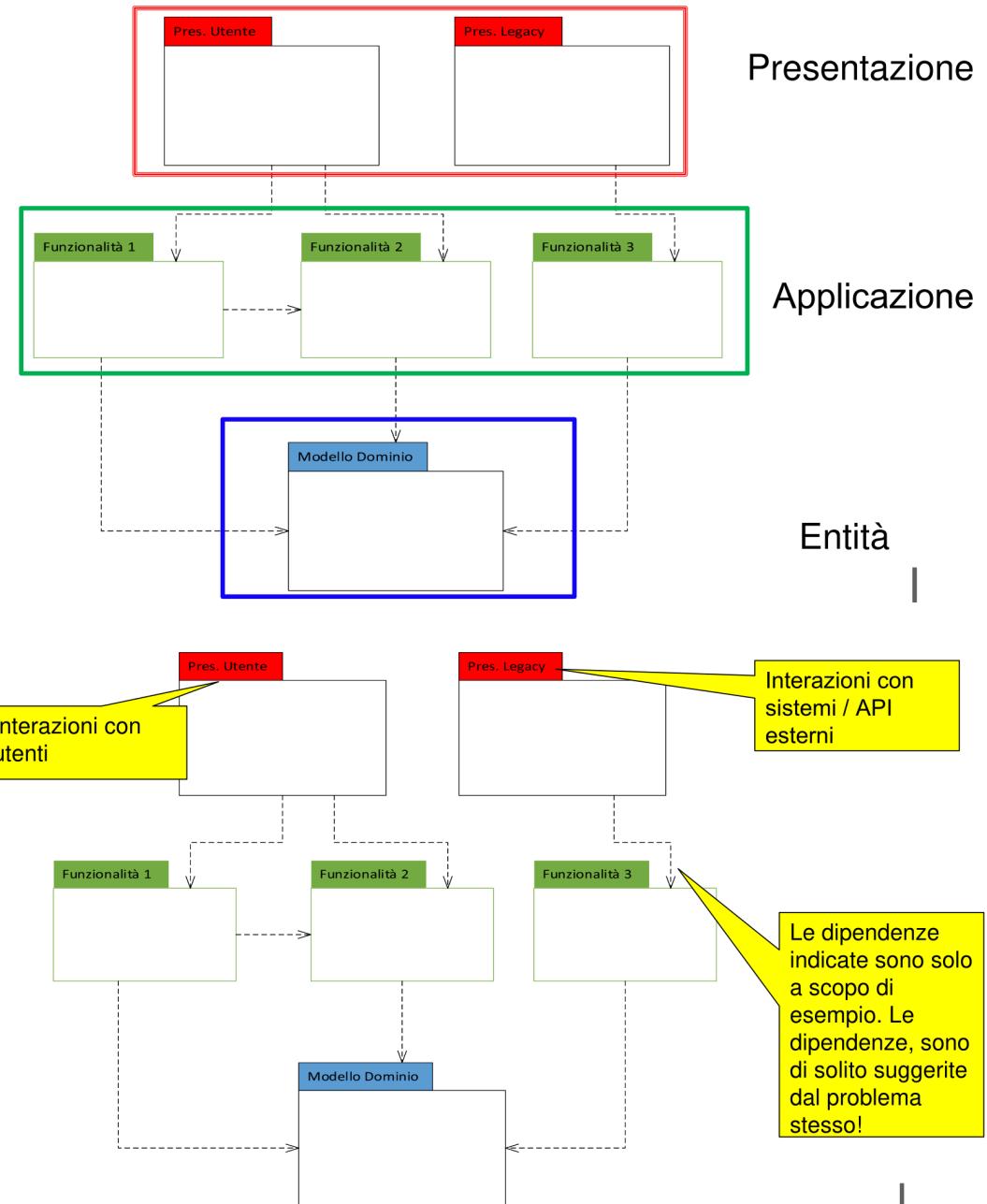
### 2.2.8.2 Layer

- È naturale separare la parte che realizza le entità dell'applicazione (dominio) dalla parte che realizza l'interazione con l'utente (logica di presentazione), introducendo una parte centrale di connessione (control)
- Nello specifico:
  - il livello di **presentazione** comprende le parti che realizzano l'interfaccia utente
    - Per aumentare la riusabilità delle parti, questo livello è progettato e costruito astraiendo quanto più possibile dai dettagli degli specifici dispositivi di I/O
  - il livello di **applicazione** comprende le parti che provvedono a elaborare l'informazione di ingresso, a produrre i risultati attesi e a presentare le informazioni in uscita
  - il livello delle **entità** forma il (modello del) dominio applicativo

### 2.2.8.3 Struttura: Package

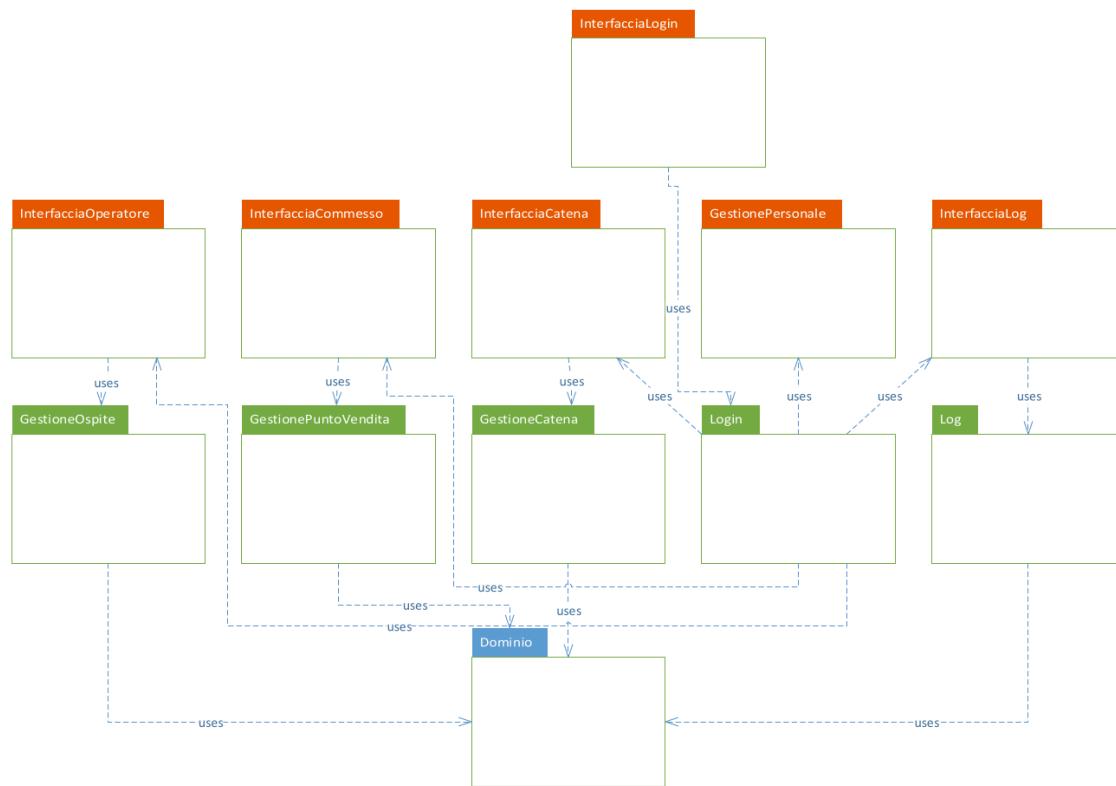
- Usando come base di partenza il lavoro di analisi svolto nelle fasi precedenti e tenendo in considerazione il pattern BCE si può iniziare la creazione del Diagramma dei Package che rappresenta la visione dal alto livello dell'Architettura Logica
- Il primo package che si può identificare è il package costituito dal **Modello del Dominio** creato nella fase precedente
  - tale Modello (se ben realizzato) costituisce la parte “**entity**” dell'architettura
- Poi è possibile creare un package per ognuna delle diverse funzionalità identificate nella Tabella delle Funzionalità (“control”); ciascuna delle righe della tabella diventa un package di controllo

- Vengono creati uno o più package per la parte di “boundary”
- Infine si identificano le *dipendenze logiche* tra i package



## 2.2.8.4 Esempio

Diagramma dei Package per il Villaggio Turistico

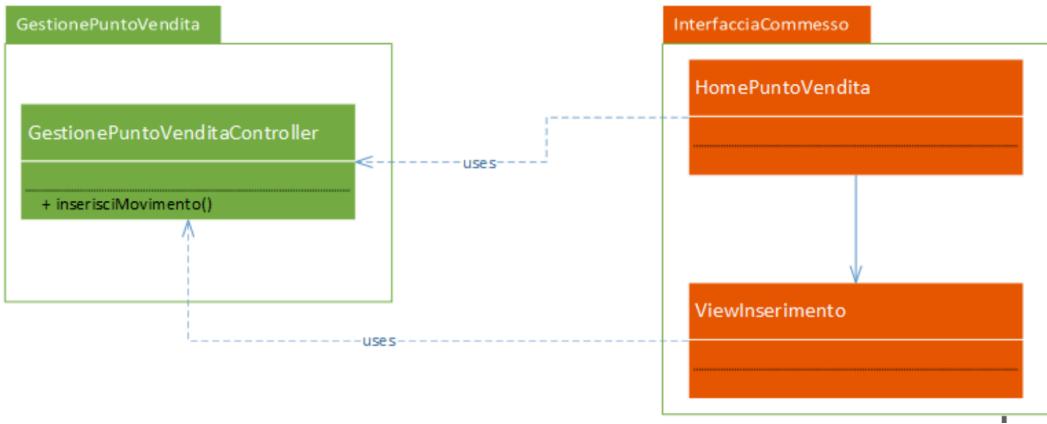


## 2.2.8.5 Struttura: Classi

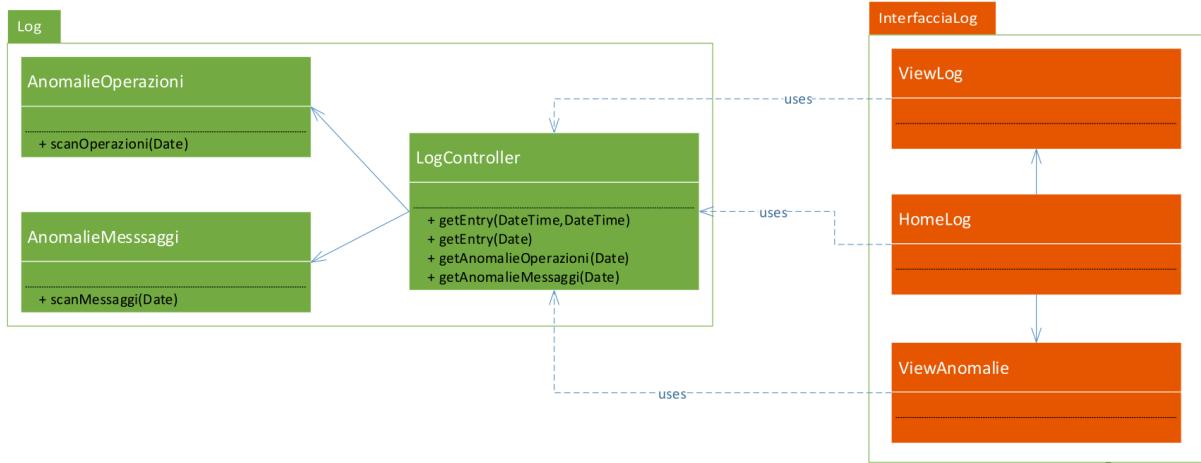
- Dopo aver stabilito la struttura di alto livello dell’Architettura Logica **si dettagliano le classi che compongono ogni package**
- Il Package del Dominio è già stato identificato nel Modello del Dominio
- Se le classi da specificare per ogni package sono poche si può realizzare un unico diagramma delle classi
- Altrimenti è possibile creare un diagramma delle classi separato per ogni package
- Attenzione a non introdurre scelte di progettazione in questa fase
- Indicare solo quelle classi che sono deducibili dal problema
  - in genere viene inserita almeno una classe che realizza le funzionalità indicate nelle specifiche
  - se nel diagramma dei package è stata indicata una funzionalità che nella fase precedente era stata poi scomposta si possono indicare le classi che realizzano le sotto-funzionalità
  - indicare le classi che rappresentano le maschere di interazione con l’utente identificate nelle fasi precedenti

## 2.2.8.6 Esempio

- Diagramma delle classi per il Villaggio Turistico
  - Diagramma delle classi: **InterfacciaCommesso & GestionePuntoVendita**



- Diagramma delle classi per il Villaggio Turistico
  - Diagramma delle classi: **Interfaccialog & Log**

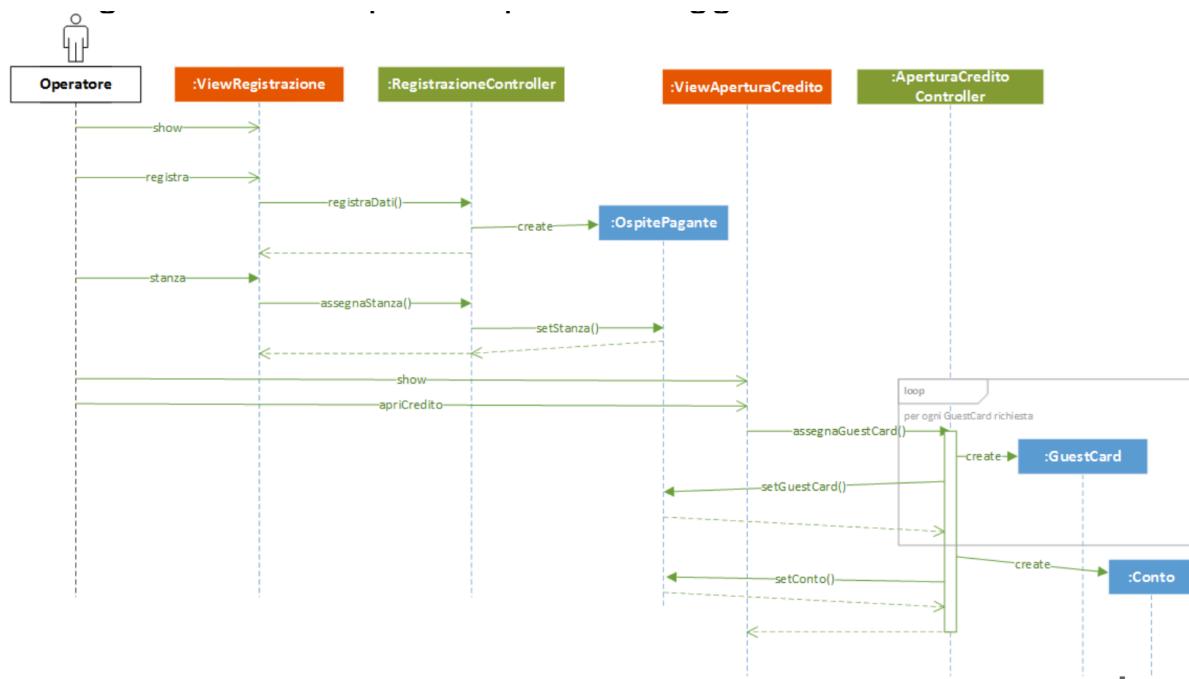


## 2.2.9 Architettura Logica: Interazione

- Descrivere le interazioni tra le entità identificate nella parte strutturale attraverso opportuni Diagrammi di Sequenza
- I **Diagrammi di sequenza** evidenziano
  - lo scambio di messaggi (le interazioni) tra gli oggetti
  - l'ordine in cui i messaggi vengono scambiati tra gli oggetti (sequenza di invocazioni delle operazioni)
- Non spingere la definizione dei diagrammi di sequenza sino ai minimi dettagli
- Utilizzare i diagrammi solo per descrivere il funzionamento del sistema
  - in risposta a sollecitazioni esterne
  - in fasi particolarmente significative
  - nei casi più critici
- Non serve mostrare tutti i diagrammi di interazione, ma magari fare vedere quelli più interessanti e/o complicati

### 2.2.9.1 Esempio

Diagramma di sequenza per il Villaggio Turistico



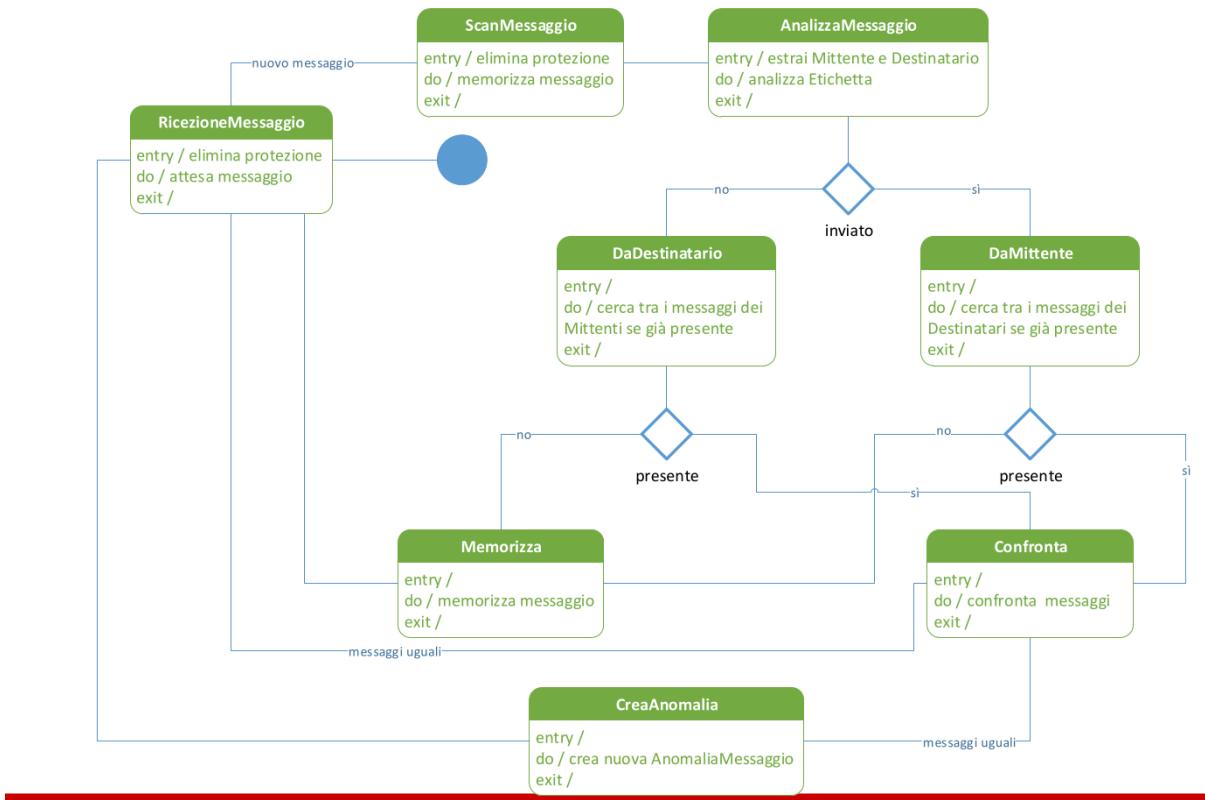
**N.B.** Mai fare vedere la password nel modello del dominio. L’interfaccia che gestisce il login avrà un auto-anello, quindi solo lei dovrà gestire il controllo password; come realizzare questa cosa si vedrà in progettazione.

### 2.2.10 Architettura Logica: Comportamento

- Descrivere il comportamento delle entità identificate nella parte strutturale attraverso opportuni Diagrammi di Stato o delle Attività
- Diagramma di Stato per mostrare come si comportano alcune entità complesse a seguito delle interazioni e degli eventi che avvengono nel sistema
- Diagramma delle Attività per dettagliare funzionamenti complessi delle entità
- Non spingere la definizione dei diagrammi sino ai minimi dettagli
- Lo stato di un oggetto è dato dal valore dei suoi attributi e delle sue associazioni
- In molti domini applicativi, esistono oggetti che, a seconda del proprio stato, rispondono in maniera diversa ai messaggi ricevuti
  - Dispositivi (spento, in attesa, operativo, guasto, ecc.)
  - Transazioni complesse (in definizione, in esecuzione, completata, fallita, ecc.)
- In questi casi, è opportuno disegnare un diagramma di stato per l’oggetto, mostrando i possibili stati e gli eventi che attivano transizioni da uno stato all’altro
- A un oggetto possono essere assegnate responsabilità che comportano un insieme di elaborazioni complesse che devono essere eseguite in un ordine particolare
- In questi casi, è opportuno disegnare un diagramma di attività per l’oggetto, mostrando le diverse elaborazioni che devono essere portate a termine e l’ordine di tali elaborazioni

#### 2.2.10.1 Esempio

Diagramma di stato per AnomalieMessaggi



## 2.2.11 Definizione del Piano di Lavoro

- Dopo la creazione dell'Architettura Logica è possibile iniziare a suddividere il lavoro
- In particolare è necessario:
  - suddividere le responsabilità ai diversi membri del team di progetto e sviluppo
  - stabilire le tempistiche per la progettazione di ciascuna parte
  - stabilire i tempi di sviluppo di ciascun sotto-sistema
  - programmare i test di integrazione tra le parti
  - identificare i tempi di rilascio delle diverse versioni del prototipo
  - identificare un piano per gli sviluppi futuri

### 2.2.11.1 Esempio

Package	Progetto	Sviluppo
Dominio	Team progettazione + Team DB	Team sviluppo A + Team DB
Gestione Ospite	Team progettazione	Team sviluppo A
GestionePuntoVendita	Team progettazione	Team sviluppo B
GestioneCatena	Team progettazione	Team sviluppo B
Login	Team sicurezza	Team sviluppo sicurezza
Log	Team sicurezza	Team sviluppo sicurezza
InterfacciaOperatore	Team progettazione + Team grafico	Team sviluppo A + Team Grafico
InterfacciaCommesso	Team progettazione + Team grafico	Team sviluppo B + Team Grafico
InterfacciaCatena	Team progettazione + Team grafico	Team sviluppo B + Team Grafico
InterfacciaLogin	Team sicurezza+ Team grafico	Team sicurezza+ Team grafico
InterfacciaLog	Team sicurezza+ Team grafico	Team sicurezza+ Team grafico
GestionePersonale	Team progettazione + Team DB	Team sviluppo A + Team DB

## 2.2.12 Definizione del Piano del Collaudo

- Al termine dell’Analisi del Problema, i modelli che definiscono il dominio e l’Architettura Logica dovrebbero dare sufficienti informazioni su *cosa* le varie parti del sistema debbano fare senza specificare ancora molti dettagli del loro comportamento
- Il “*cosa fare*” di una parte dovrà comprendere anche le forme di interazione con le altre parti
- Lo scopo del *piano del collaudo* è cercare di precisare il comportamento atteso da parte di una entità prima ancora di iniziare il progetto e la realizzazione
- Focalizzando l’attenzione sulle interfacce delle entità e sulle interazioni è possibile impostare scenari in cui specificare in modo già piuttosto dettagliato la “risposta” di una parte a uno “stimolo” di un’altra parte
- Lo sforzo di definire nel modo *più preciso possibile* un piano del collaudo di un sistema prima ancora di averne iniziato la fase di progettazione viene ricompensato da
  - una *miglior comprensione* dei requisiti
  - un approfondimento nella *comprensione dei problemi*
  - una più precisa definizione dell’insieme delle funzionalità (operazioni) che ciascuna parte deve fornire alle altre per una *effettiva integrazione* nel “tutto” che costituirà il sistema da costruire
  - comprendere il *significato delle entità* e specificarne nel modo più chiaro possibile il *comportamento atteso*

### 2.2.12.1 Definizione Piano del Collaudo

- Un piano del collaudo va concepito e impostato da un punto di **vista logico**, cercando di individuare categorie di comportamenti e punti critici
- In molti casi tuttavia può anche risultare possibile definire in modo precoce **piani di collaudo concretamente eseguibili**, avvalendosi di strumenti del tipo JUnit/JUnit che sono ormai diffusi in tutti gli ambienti di programmazione
- Lo sforzo di definire un piano di collaudo concretamente eseguibile promuove uno sviluppo **controllato, sicuro e consapevole** del codice poiché il progettista e lo sviluppatore possono verificare subito in modo concreto la correttezza di quanto sviluppato

### 2.2.12.2 JUnit

- JUnit (<https://junit.org/junit5/docs/current/user-guide/>) è un framework per unit-testing per il linguaggio Java
- Dovreste conoscerlo già bene da Fondamenti T-2

### 2.2.12.3 NUnit

- NUnit (<http://nunit.org/>) è un framework per unit-testing per tutti i linguaggi .Net
- NUnit è inizialmente derivato da JUnit, ma è stato totalmente riscritto dalla versione 3
- Troverete la documentazione e la guida all'installazione a <https://github.com/nunit/docs/wiki/Framework-Release-Notes>

## 3 Framework .NET

### 3.1 Introduzione

#### 3.1.1 Tecnologia COM - Component Object Model

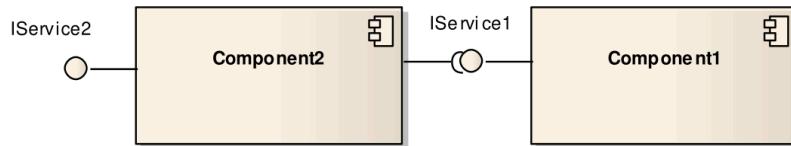
- Nasce nel 1993
- COM è un sistema platform-independent, distribuito, object-oriented per la creazione di componenti software binari
- COM non è un linguaggio object-oriented ma uno **standard**
- COM specifica un modello a oggetti e requisiti di programmazione che permettono agli oggetti COM di interagire con altri oggetti
- Ad esempio un programma Java può parlare con un programma C, utilizzando la Java Native Interface
- COM fa la stessa cosa del linker C quando abbiamo bisogno che più programmi parlino tra loro (link dinamico)

Esempio di oggetti COM che parlano tra di loro:

```
interface IUnknown
{
    virtual HRESULT QueryInterface(IID iid, void **ppvObject) = 0;
    virtual ULONG AddRef(void) = 0;
    virtual ULONG Release(void) = 0;
};
```

- QueryInterface è utilizzata per ottenere un puntatore a un'altra interfaccia (a tempo di esecuzione, non di compilazione), dato un GUID che identifica univocamente tale interfaccia (noto comunemente come interfaceID, o IID)
  - se l'oggetto COM non implementa tale interfaccia, viene restituito un errore E\_NOINTERFACE
- AddRef è utilizzato dai client per indicare che un oggetto COM viene referenziato (usato)
- Release è utilizzato dai client per indicare che hanno finito di utilizzare l'oggetto COM
- Le specifiche COM richiedono l'utilizzo di una tecnica chiamata **reference counting** per assicurare che i singoli oggetti rimangano “vivi” fintantoché esistono client che hanno ottenuto l’accesso a una o più delle loro interfacce e, di converso, che gli stessi oggetti vengano appropriatamente cancellati quando i clienti che li utilizzavano hanno finito di usarli e non ne hanno più bisogno; il contatore viene incrementato a ogni AddRef e decrementato a ogni Release
- Un oggetto COM è responsabile della liberazione della propria memoria una volta che il suo reference count arrivi a zero
- Il reference counting può causare **problemi se due o più oggetti hanno riferimenti circolari**
- Ereditarietà solo con composizione e delega

## interface IService2 : IService1



- La **posizione** di ciascun componente è memorizzata nel registro Windows
- Di un certo componente può esistere un'unica versione installata
- Questa limitazione può complicare seriamente il deployment di applicazioni basate su COM, a causa della possibilità che diversi programmi, o anche diverse versioni dello stesso programma, siano progettati per funzionare con versioni diverse dello stesso componente COM
- Questa situazione è nota anche come inferno delle DLL (DLL hell)

### 3.1.2 Cos’è il Framework .NET

- Ambiente di esecuzione (runtime environment) + Libreria di classi (standard + estensioni MS)
- Versione 1.0 del 2002 ► v. 4.8 (versione definitiva, 7/19)
- Semplifica lo sviluppo e il deployment
- Aumenta l'affidabilità del codice
- Unifica il modello di programmazione

- È completamente indipendente da COM
- È fortemente integrato con COM
- Ambiente object-oriented
  - Qualsiasi entità è un oggetto
  - Classi ed ereditarietà pienamente supportati
- Riduzione errori comuni di programmazione
  - **Garbage Collector**
  - Linguaggi fortemente tipizzati - **Type Checker**
  - Errori non gestiti ▷ generazione di eccezioni
- Libertà di scelta del linguaggio
  - Funzionalità del framework disponibili in tutti i linguaggi .NET
  - I componenti della stessa applicazione possono essere scritti in linguaggi diversi
  - Ereditarietà supportata anche tra linguaggi diversi
- Possibilità di estendere una qualsiasi classe .NET (non sealed) mediante ereditarietà
- Diversamente da COM:
  - si usa e si estende la classe stessa
  - non si deve utilizzare composizione e delega
- .NET è un'implementazione di CLI
  - Common Language Infrastructure
- CLI e il linguaggio C# sono standard ECMA
  - ECMA-334 (C#), ECMA-335 (CLI)
- Esistono altre implementazioni di CLI:
  - SSCLI (Shared Source CLI by Microsoft, per Windows, FreeBSD e Macintosh) - Rotor
  - Mono (per Linux)
  - DotGNU
  - Intel OCL (Open CLI Library)
  - ...

### 3.1.3 Standard ECMA-335

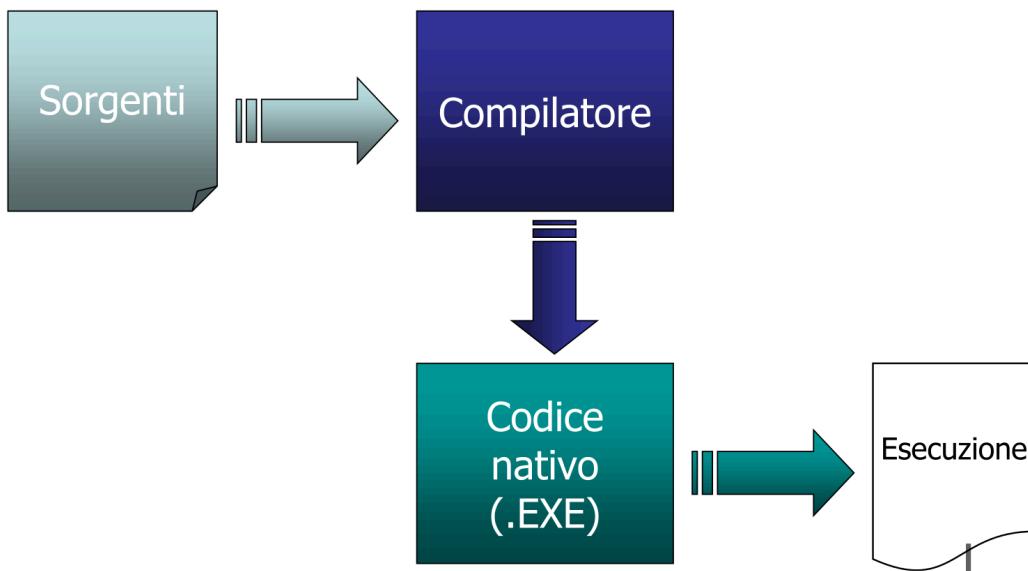
- Definisce la Common Language Infrastructure (CLI) nella quale applicazioni scritte in **diversi linguaggi di alto livello** possono essere eseguite in **diversi ambienti di sistema** senza la necessità di riscrivere l'applicazione per prendere in considerazione le caratteristiche peculiari di tali ambienti
- CLI è un **ambiente a tempo di esecuzione**, con:
  - un formato di file
  - un sistema di tipi comune
  - un sistema di metadati estensibile
  - un linguaggio intermedio
  - accesso alla piattaforma sottostante
  - una libreria di classi base
- Concetti chiave:
  - **(Microsoft) Intermediate Language** - (MS)IL

- ▶ **Common Language Runtime** - CLR
  - ambiente di esecuzione runtime per le applicazioni .NET
  - il codice che viene eseguito sotto il suo controllo si dice **codice gestito** (managed)
- ▶ **Common Type System** - CTS
  - tipi di dato supportati dal framework .NET
  - consente di fornire un modello di programmazione unificato
- ▶ **Common Language Specification** - CLS
  - regole che i linguaggi di programmazione devono seguire per essere interoperabili all'interno del framework .NET
  - sottoinsieme di CTS

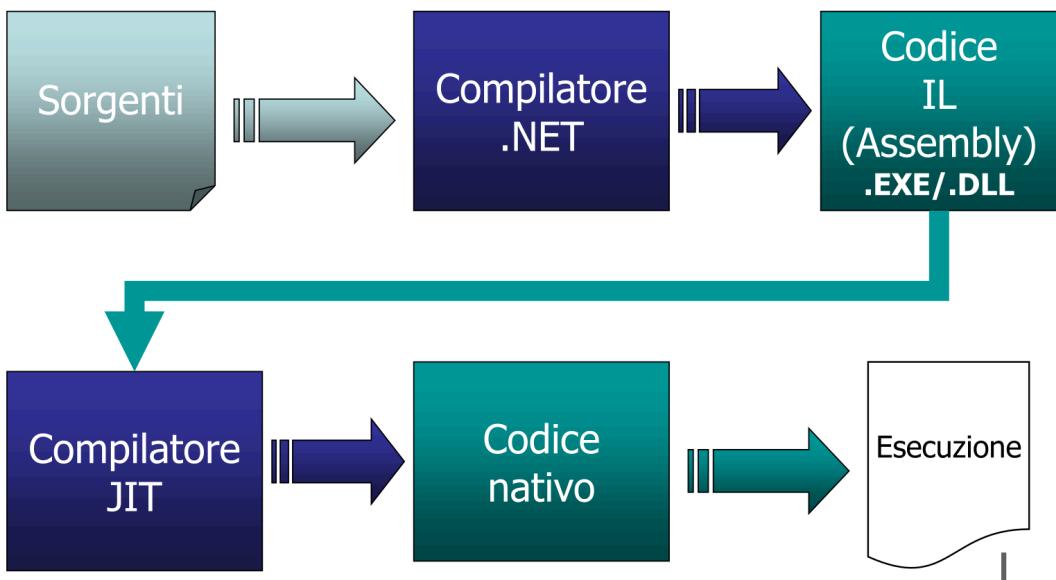
### 3.1.4 Codice interpretato



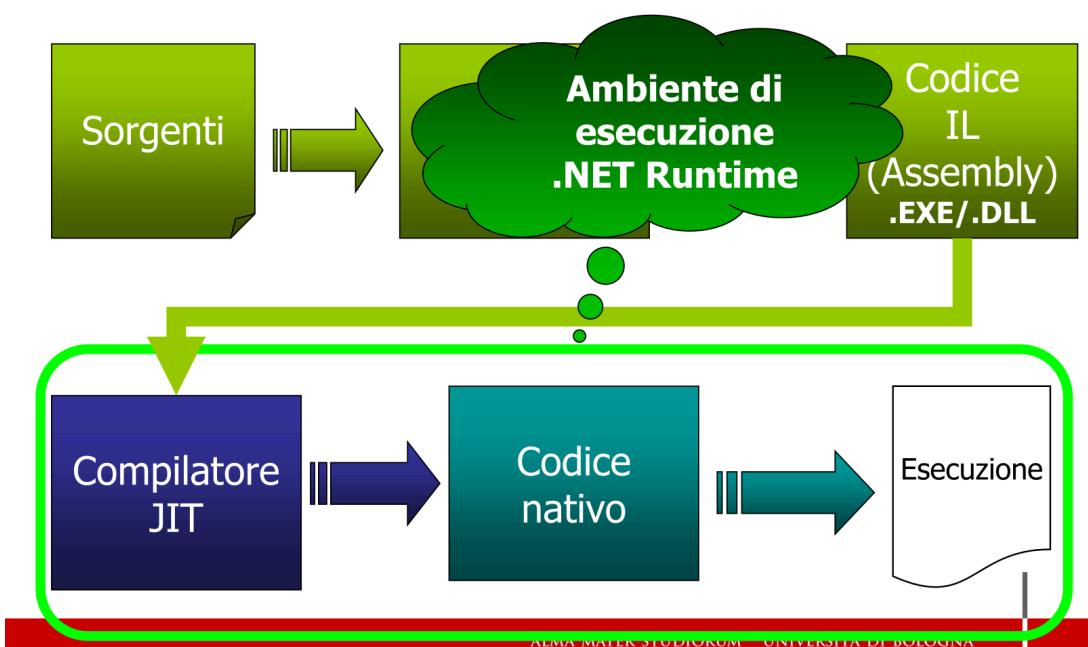
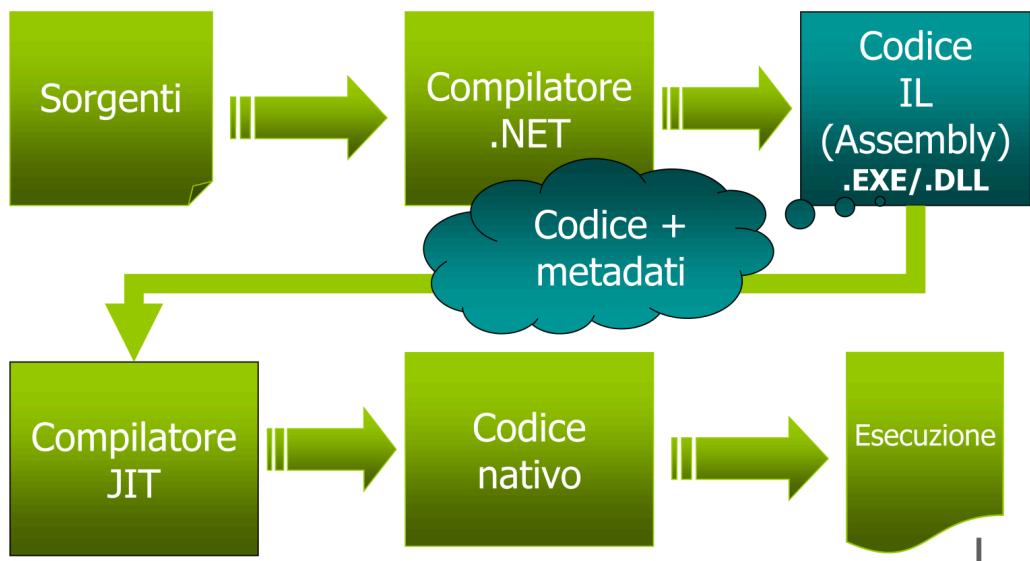
### 3.1.5 Codice nativo



### 3.1.6 Codice IL

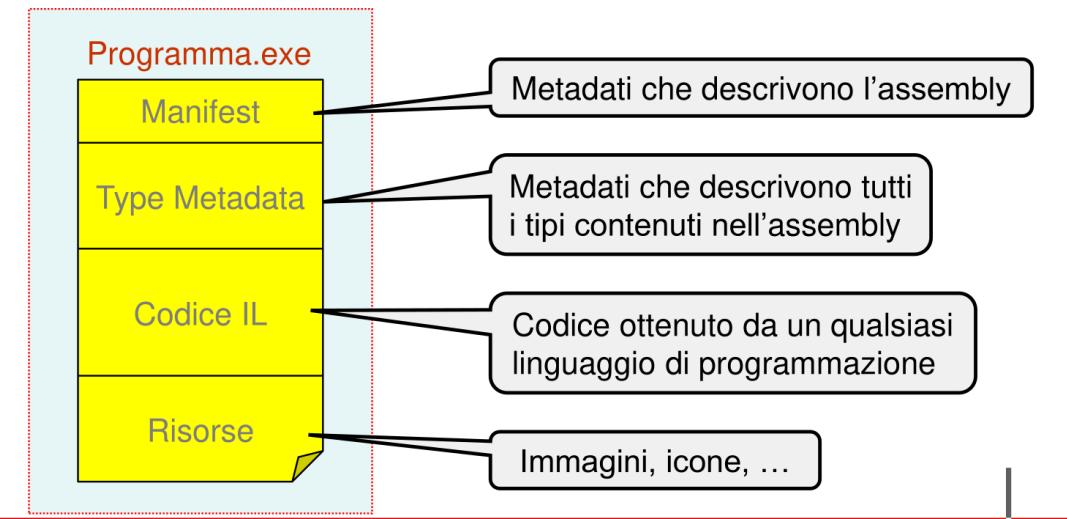


JIT: just in time

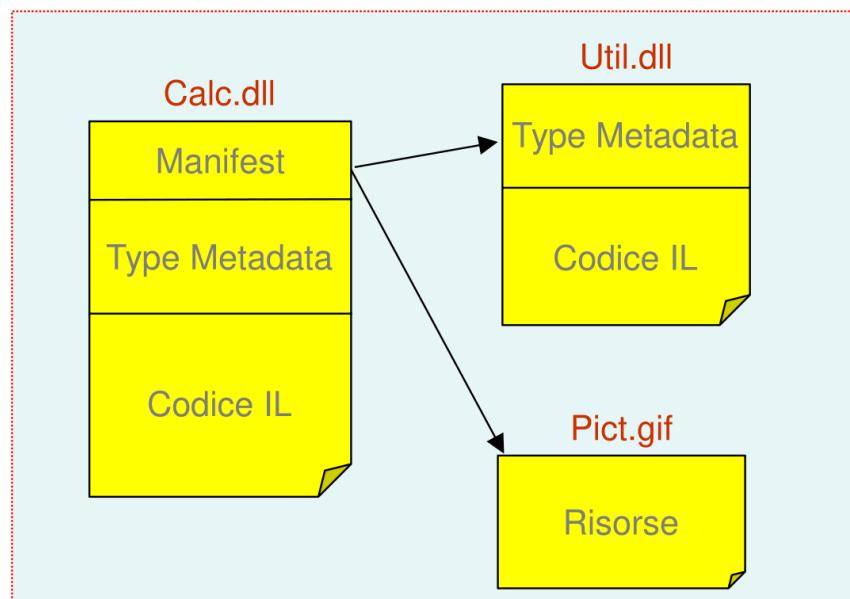


### 3.1.7 Assembly

- Unità minima per la distribuzione e il versioning
- Normalmente è composto da un solo file



- Ma può essere composto anche da più file



### 3.1.8 Metadati

- Descrizione dell'assembly - Manifest
  - Identità: nome, versione, cultura [, public key]
  - Lista dei file che compongono l'assembly
  - Riferimenti ad altri assembly da cui si dipende
  - Permessi necessari per l'esecuzione
  - ...
- Descrizione dei tipi contenuti nell'assembly
  - Nome, visibilità, classe base, interfacce
  - Campi (attributi membro), metodi, proprietà, eventi, ...
  - Attributi (caratteristiche aggiuntive)
    - definiti dal compilatore
    - definiti dal framework
    - definiti dall'utente

### 3.1.9 Chi usa i metadati?

- Compilatori
  - Compilazione condizionale
- Ambienti RAD (Rapid Application Development)
  - Informazioni sulle proprietà dei componenti

- Categoria
- Descrizione
- Editor specializzati per tipo di proprietà
- Tool di analisi dei tipi e del codice
  - Intellisense, ILDASM, Reflector, ...
- Sviluppatori - **Reflection** (introspezione)
  - Analisi del contenuto di un assembly, permetter agli altri di guardare dentro

### 3.1.10 Esempio Assembly

```
.assembly Hello { }
.assembly extern mscorel { }
.method public static void main()
{
    .entrypoint
    ldstr "Hello IL World!"
    call void [mscorel]System.Console::WriteLine
    (class System.String)
    ret
}

ilasm helloil.il
```

### 3.1.11 Dove trovare gli Assembly

- **Assembly privati**
  - Utilizzati da un'applicazione specifica
  - Directory applicazione (e sub-directory)
- **Assembly condivisi**
  - Utilizzati da più applicazioni
  - Global Assembly Cache (GAC)
  - c:\windows\assembly
- **Assembly scaricati da URL**
  - Download cache
  - c:\windows\assembly\download

### 3.1.12 Deployment semplificato

- Installazione senza effetti collaterali
  - Applicazioni e componenti possono essere
    - condivisi
    - privati
- Esecuzione side-by-side
  - Versioni diverse dello stesso componente possono coesistere, anche nello stesso processo

### 3.1.13 Common Language Runtime

VB

C++

C#

JScript

...

## Common Language Specification

Web  
Services

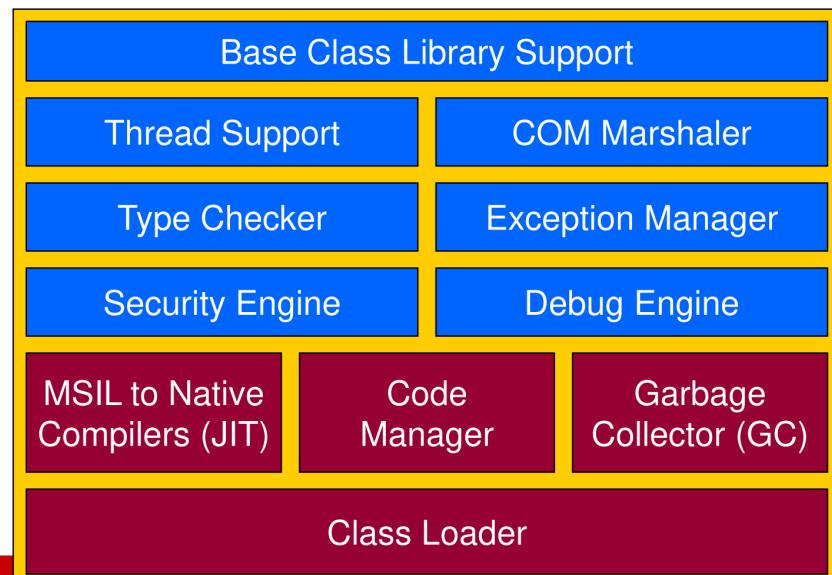
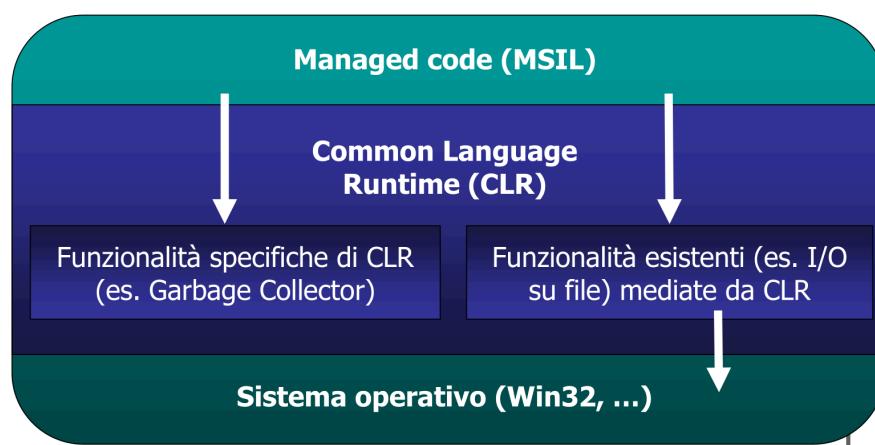
User  
Interface

Data and XML

Base Class Library

Common Language Runtime

- IL CLR offre vari servizi alle applicazioni



### 3.1.14 Garbage Collector

- Gestisce il ciclo di vita di tutti gli oggetti .NET
- Gli oggetti vengono distrutti automaticamente quando non sono più referenziati
- A differenza di COM, non si basa sul Reference Counting

- ▶ Maggiore velocità di allocazione
- ▶ Consentiti i riferimenti circolari
- ▶ Perdita della distruzione deterministica: non posso sapere in che momento un determinato oggetto verrà distrutto
- ▶ Inoltre, essendo più complicato del garbage collector di COM, l'esecuzione dello stesso è un po' più pesante

### 3.1.15 Gestione delle eccezioni

- Praticamente uguali a quelle di Java
- Un'eccezione è
  - ▶ una condizione di errore
  - ▶ un comportamento inaspettato

incontrato durante l'esecuzione del programma

- Un'eccezione può essere generata da
  - ▶ codice del programma in esecuzione
  - ▶ ambiente di runtime
- In CLR, un'eccezione è un oggetto che eredita dalla classe `System.Exception`
- Gestione uniforme, elimina
  - ▶ codici HRESULT di COM
  - ▶ codici di errore Win32
  - ▶ ...

### 3.1.16 Gestione delle eccezioni

- Concetti universali
  - ▶ Lanciare un'eccezione (`throw`)
  - ▶ Catturare un'eccezione (`catch`)
  - ▶ Eseguire codice di uscita da un blocco controllato (`finally`)
- Disponibile in tutti i linguaggi .NET con sintassi diverse

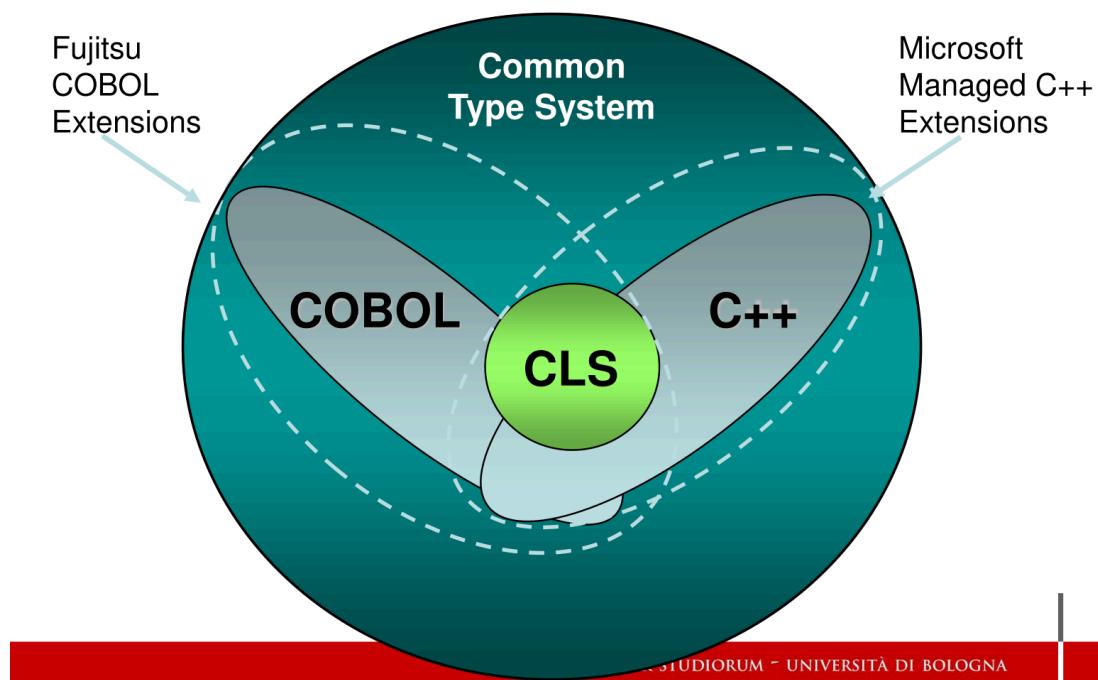
### 3.1.17 Common Type System

- Tipi di dato supportati dal framework .NET
  - ▶ Alla base di tutti i linguaggi .NET
- Fornisce un modello di programmazione unificato
- Progettato per linguaggi object-oriented, procedurali e funzionali
  - ▶ Esaminate caratteristiche di 20 linguaggi
  - ▶ Tutte le funzionalità disponibili con IL
  - ▶ Ogni linguaggio utilizza alcune caratteristiche
- Alla base di tutto ci sono i tipi: **classi, strutture, interfacce, enumerativi, delegati**
- Fortemente tipizzato (compile-time)
- Object-oriented
  - ▶ Campi, metodi, tipi annidati, proprietà, ...
- Overload di metodi (compile-time)
- Invocazione metodi virtuali risolta a run-time
- Ereditarietà singola di estensione

- Ereditarietà multipla di interfaccia

### 3.1.18 Common Language Specification

- Definisce le regole di compatibilità tra linguaggi (sottoinsieme di CTS)
  - Regole per gli identificatori
    - Unicode, case-sensitivity
    - Keyword
  - Regole per denominazione proprietà ed eventi
  - Regole per costruttori degli oggetti
  - Regole di overload più restrittive
  - Ammesse interfacce multiple con metodi con lo stesso nome
  - Non ammessi puntatori unmanaged
  - ...



#### 3.1.18.1 Tipi nativi

CTS	C#
System.Object	object
System.String	string
System.Boolean	bool
System.Char	char
System.Single	float
System.Double	double
System.Decimal	decimal
System.SByte	sbyte
System.Byte	byte
System.Int16	short
System.UInt16	ushort
System.Int32	int
System.UInt32	uint
System.Int64	long
System.UInt64	ulong

### 3.1.19 Common Type System

- Tutto è un oggetto

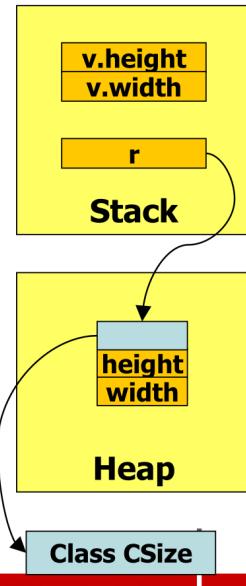
- ▶ System.Object è la classe radice
- Due categorie di tipi
  - ▶ Tipi riferimento
    - Riferimenti a oggetti allocati sull'**heap** gestito
    - Indirizzi di memoria
  - ▶ Tipi valore
    - Allocati sullo **stack** o parte di altri oggetti
    - Sequenza di byte
- Sono memorizzati come in Java

### 3.1.19.1 Tipi valore

- I tipi valore comprendono:
  - ▶ Tipi primitivi (built-in)
    - Int32, ...
    - Single, Double
    - Decimal
    - Boolean
    - Char
  - ▶ Tipi definiti dall'utente
    - Strutture (struct)
    - Enumerativi (enum)

### 3.1.19.2 Tipi valore vs tipi riferimento

```
public struct Size
{
    public int height;
    public int width;
}
public class CSize
{
    public int height;
    public int width;
}
public static void Main()
{
    Size v;           // v istanza di Size
    v.height = 100;   // ok
    CSize r;          // r è un reference
    r.height = 100;   // NO, r non assegnato
    r = new CSize();  // r fa riferimento a un CSize
    r.height = 100;   // ok, r inizializzata
}
```



ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

Appena dichiaro la variabile **r**, questa viene salvato nello Stack, poi, quando viene inizializzata, viene creato un riferimento della classe **Csize** nello Heap.

### 3.1.19.3 Common Type System

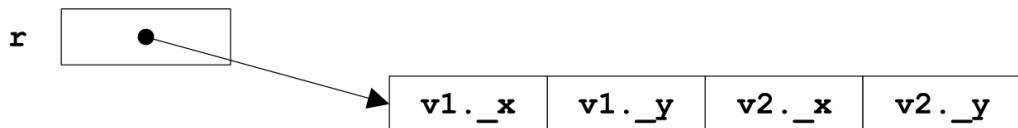
```
public struct Point
{
    private int _x, _y;
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }
    public int X
```

```

    {
        get { return _x; }
        set { _x = value; }
    }
    public int Y
    {
        get { return _y; }
        set { _y = value; }
    }
}

public class Rectangle
{
    Point v1;
    Point v2;
    ...
}
...
Rectangle r = new Rectangle();

```



```

...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
    points[i] = new Point(i, i);
...
• Alla fine, rimane un solo oggetto nell'heap (l'array di Point)

```

```

...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
{
    points[i].X = i;
    points[i].Y = i;
}
...

```

```

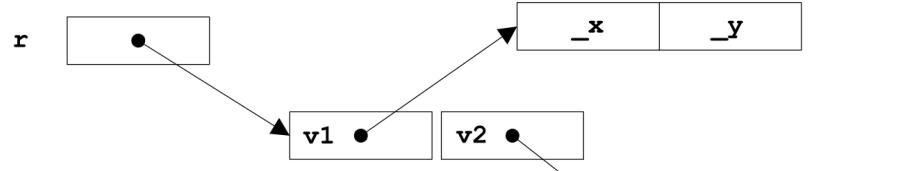
public class Point
{
    private int _x, _y;
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }
    public int X
    {
        get { return _x; }
        set { _x = value; }
    }
}

```

```

public int Y
{
    get { return _y; }
    set { _y = value; }
}

```



```

public class Rectangle
{
    Point v1;
    Point v2;
    ...
}

...
Rectangle r = new Rectangle();

```

```

...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
points[i] = new Point(i, i);
...

```

- Alla fine, rimangono 101 oggetti nell'heap (1 array di Point + 100 Point )

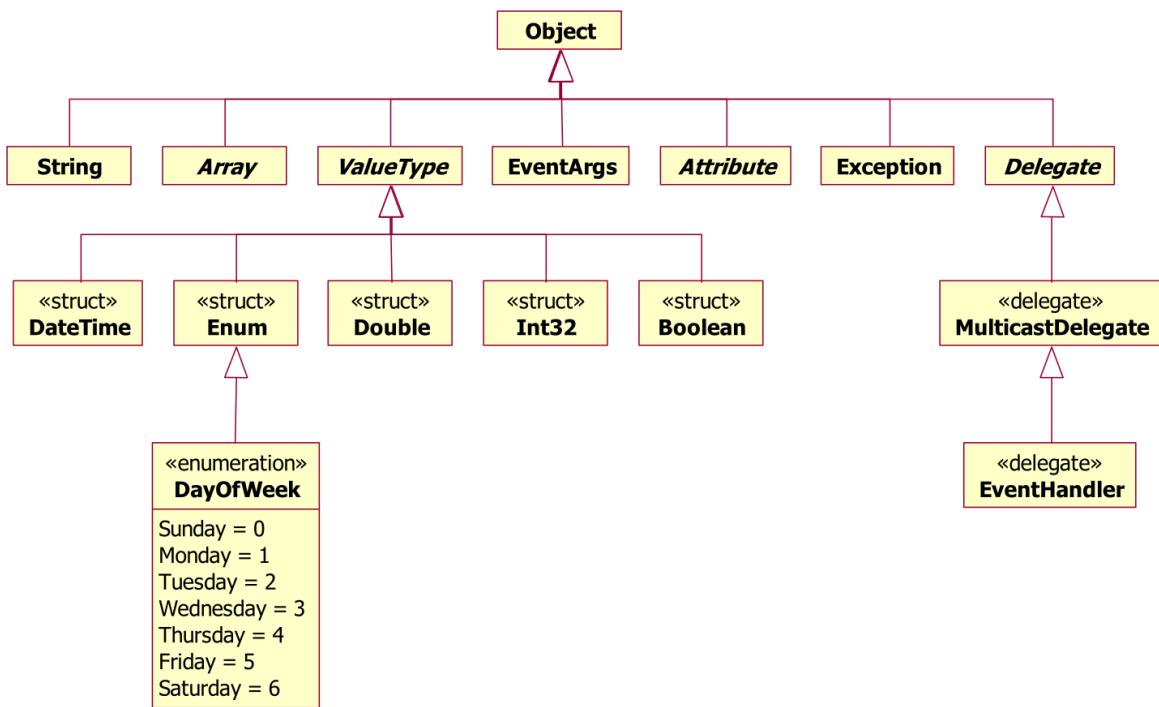
```

...
Point[] points = new Point[100];
for (int i = 0; i < 100; i++)
{
    points[i].X = i;
    points[i].Y = i;
}
...

```

NO!

### 3.1.19.4 Tipi valore e tipi riferimento



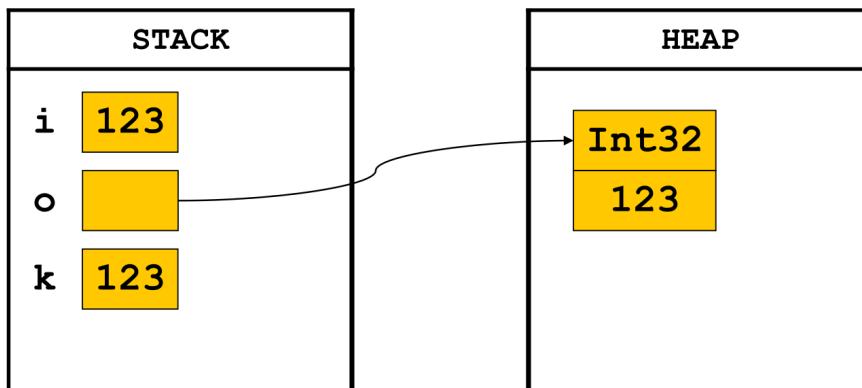
### 3.1.20 Boxing / Unboxing

- Un qualsiasi tipo valore può essere automaticamente convertito in un tipo riferimento (**boxing**) mediante un up cast implicito a `System.Object`

```
int i = 123;
object o = i;
```
- Un tipo valore “boxed” può tornare a essere un tipo valore standard (**unboxing**) mediante un down cast esplicito
 

```
int k = (int) o;
```
- Un tipo valore “boxed” è un **clone indipendente**, quindi se, dopo aver eseguito il boxing, cambio il valore di `i`, il valore di `k` non viene modificato

```
int i = 123;
object o = i;
int k = (int) o;
```



### 3.1.21 Bibliografia

Libri di base:

- D. S. Platt, *Introducing Microsoft® .NET*, Second Edition
- J. Sharp, J. Jagger, *Microsoft® Visual C#™ .NET Step by Step*
- T. Archer, A. Whitechapel, *Inside C#, Second Edition*

- M. J. Young, XML Stepby Step, Second Edition
- R. M. Riordan, Microsoft® ADO.NET Stepby Step

Libri avanzati:

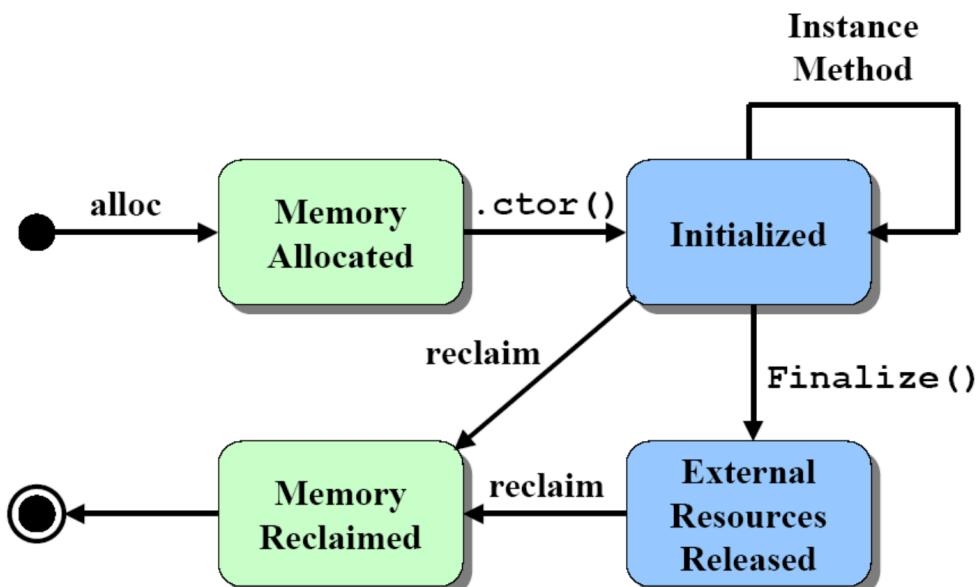
- J. Richter, AppliedMicrosoft® .NET Framework Programming
- C. Petzold, Programming Microsoft® Windows® with C#
- S. Lidin, Inside Microsoft® .NET IL Assembler

## 3.2 Garbage Collection

### 3.2.1 Utilizzo di un oggetto

- In un ambiente object-oriented, ogni oggetto che deve essere utilizzato dal programma
  - È descritto da un tipo
  - Ha bisogno di un'area di memoria dove memorizzare il suo stato
  - Una volta che si conosce il tipo dell'oggetto sappiamo quanta memoria allocare per lo stesso
- Passi per utilizzare un oggetto di tipo riferimento:
  - **Allocare memoria** per l'oggetto
  - **Inizializzare la memoria** per rendere utilizzabile l'oggetto
  - **Usare l'oggetto**
  - **Eseguire un clean up** dello stato dell'oggetto, se necessario; se, ad esempio, all'interno di quell'oggetto c'era un riferimento a un puntatore a file, quel file deve essere chiuso
  - **Liberare la memoria**

### 3.2.2 Ciclo di vita di un oggetto



### 3.2.3 Allocazione della memoria

- In C:
  - `malloc` ( `calloc` , `realloc` )
- In C++:
  - `malloc` ( `calloc` , `realloc` )
  - `new`
- In Java:

- ▶ new
- In IL:
  - ▶ newobj
- In C#:
  - ▶ new

### 3.2.4 Inizializzazione della memoria

- Definite Assignment: a ogni variabile deve essere sempre assegnato un valore prima che essa venga utilizzata
- il compilatore deve assicurarsi che ciò sia sempre verificato
- Data-flow analysis del codice
- valori di default
- usati in generale per tipi valore
- ad esempio, in Java le variabili di classe, locali e i componenti di un array sono inizializzati al valore di default, non le variabili di istanza, perché in quel caso è il costruttore che si deve occupare di inizializzarle
- costruttore
- usato per i tipi classe (Java, C++, C#)

### 3.2.5 Definite Assignment

```
int k;
if (v > 0 && (k = System.in.read()) >= 0)
    System.out.println(k);
```

Questo è corretto

```
int k;
while (n < 4) {
    k = n;
    if (k >= 5) break;
    n = 6;
}
System.out.println(k);
```

Questo è sbagliato, perché il compilatore non sa il valore di n

```
int k;
while (true) {
    k = n;
    if (k >= 5) break;
    n = 6;
}
System.out.println(k);
```

Questa è corretta

```

int k;
int n=5;
if (n>2) k=3;
System.out.println(k);

```

Questo non è corretto perché, a differenza dell'esempio sopra, l'espressione all'interno dell'`if` non è costante; se pensiamo a `n` come variabile condivisa, il valore di `n` potrebbe cambiare tra l'assegnamento e la valutazione.

### 3.2.6 Clean up dello stato

- In C++/C#:
  - distruttore (più propriamente, finalizzatore): `~{nome della classe}`
  - unico, non ereditabile, no overload, senza parametri e modificatori
  - invocato automaticamente alla distruzione dell'oggetto (non può essere invocato)
- In java:
  - `finalize()`
  - metodo di `Object`
  - invocato automaticamente alla distruzione dell'oggetto (non può essere invocato)
  - il momento in cui viene invocato un finalizzatore dipende dalla JVM

### 3.2.7 Liberazione della memoria

- In C:
  - `free()`
- In C++:
  - `free()`
  - `delete`
- In java/C#: garbage collector (GC)

### 3.2.8 Garbage Collection

- Modalità automatica di rilascio delle risorse utilizzate da un oggetto
- Migliora la stabilità dei programmi
  - Evita errori connessi alla necessità, da parte del programmatore, di manipolare direttamente i puntatori alle aree di memoria
- Pro:
  - dangling pointer: puntatore che fa riferimento a un'area di memoria non più valida
  - doppia de-allocazione
  - memory leak
- Contro:
  - aumentata richiesta risorse di calcolo
  - incertezza del momento in cui viene effettuata la GC
  - rilascio della memoria non deterministico; quindi non posso distruggere un oggetto quando voglio io
- Strategie disponibili:
  - Tracing
    - determinare quali oggetti sono (potenzialmente) raggiungibili
    - eliminare gli oggetti non raggiungibili
  - Reference counting

- ogni oggetto contiene un contatore che indica il numero di riferimenti a esso
- la memoria può essere liberata quando il contatore raggiunge lo 0
- Escape analysis
  - si spostano oggetti dallo heap allo stack
  - l'analisi viene effettuata a compile-time in modo da stabilire se un oggetto, allocato all'interno di una subroutine, non è accessibile al di fuori di essa
  - riduce il lavoro del GC

### 3.2.9 GC: Reference counting

- Svantaggi:
  - Cicli di riferimenti
    - se due oggetti si referenziano a vicenda, il loro contatore non raggiungerà mai 0
  - Aumento dell'occupazione di memoria
  - Riduzione della velocità delle operazioni sui riferimenti
    - ogni operazione su un riferimento deve anche incrementare/decrementare i contatori
  - Atomicità dell'operazione
    - ogni modifica a un contatore deve essere resa operazione atomica in ambienti multi-threaded
  - Assenza di comportamento real-time
    - ogni operazione sui riferimenti può (potenzialmente) causare la de-allocazione di diversi oggetti
    - il numero di tali oggetti è limitato solamente dalla memoria allocata

### 3.2.10 GC: Tracing

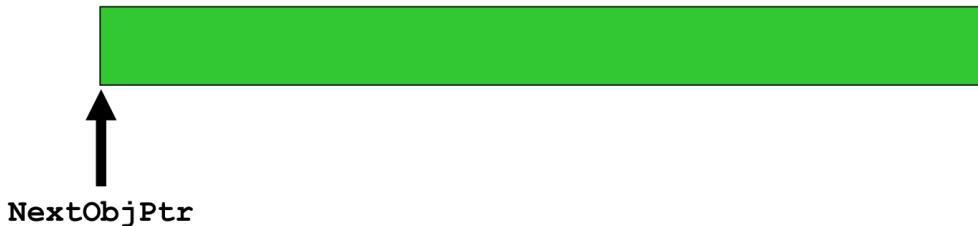
- Siano *p* e *q* due oggetti
- Sia *q* un oggetto raggiungibile
- Diremo che *p* è raggiungibile in maniera ricorsiva se e solo se:
  - esiste un riferimento a *p* tramite *q*
  - ovvero *p* è raggiungibile attraverso un oggetto, a sua volta raggiungibile
- Un oggetto è pertanto raggiungibile in due soli casi:
  - è un oggetto radice
    - creato all'avvio del programma (oggetto globale)
    - creato da una sub-routine (oggetto scope, riferito da variabile sullo stack)
  - è referenziato da un oggetto raggiungibile
    - la raggiungibilità è una chiusura transitiva
- La definizione di garbage tramite la raggiungibilità non è ottimale
  - può accadere che un programma utilizzi per l'ultima volta un certo oggetto molto prima che questo diventi irraggiungibile
- Distinzione:
  - garbage **sintattico**  
(oggetti che il programma non può raggiungere)

- ▶ **garbage semantico**  
 (oggetti che il programma non vuole più usare)
  - problema solo parzialmente decidibile, quindi non possono essere creati algoritmi per risolvere questo problema

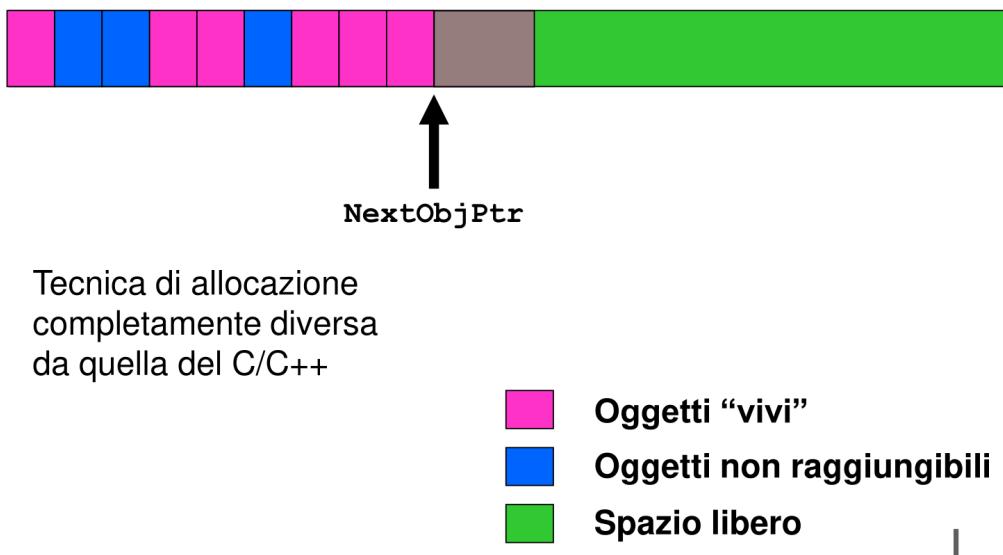
```
Object x = new Foo();
Object y = new Bar();
x = new Quux(); // qui l'oggetto Foo è garbage sintattico
if(x.check_something())
x.do_something(y); // qui y *potrebbe* essere garbage
semantico
```

### 3.2.11 Allocazione della memoria

- In fase di inizializzazione di un processo, il CLR
  - ▶ Riserva una regione contigua di spazio di indirizzamento managedheap
  - ▶ Memorizza in un puntatore (NextObjPtr) l'indirizzo di partenza della regione



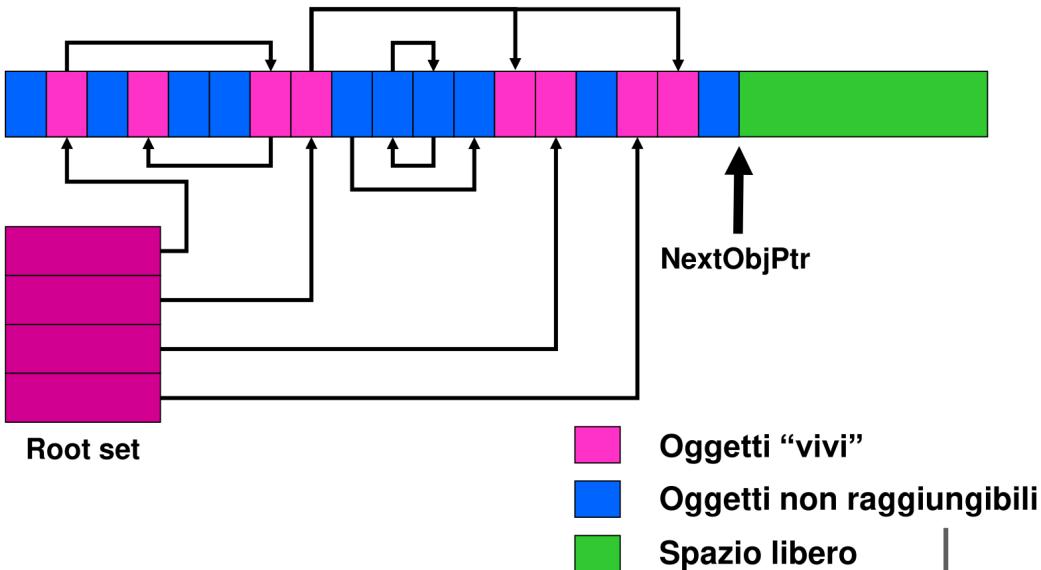
- Quando deve eseguire una newobj, il CLR
  - ▶ Calcola la dimensione in byte dell'oggetto e aggiunge all'oggetto due campi di 32 (o 64) bit
    - Un puntatore alla tabella dei metodi
    - Un campo SyncBlockIndex
  - ▶ Controlla che ci sia spazio sufficiente a partire da NextObjPtr
    - in caso di spazio insufficiente:
      - garbage collection
      - OutOfMemoryException
  - ▶ thisObjPtr= NextObjPtr;
  - ▶ NextObjPtr += sizeof(oggetto);
  - ▶ Invoca il costruttore dell'oggetto ( this ≡ thisObjPtr )
  - ▶ Restituisce il riferimento all'oggetto



### 3.2.12 Garbage Collector

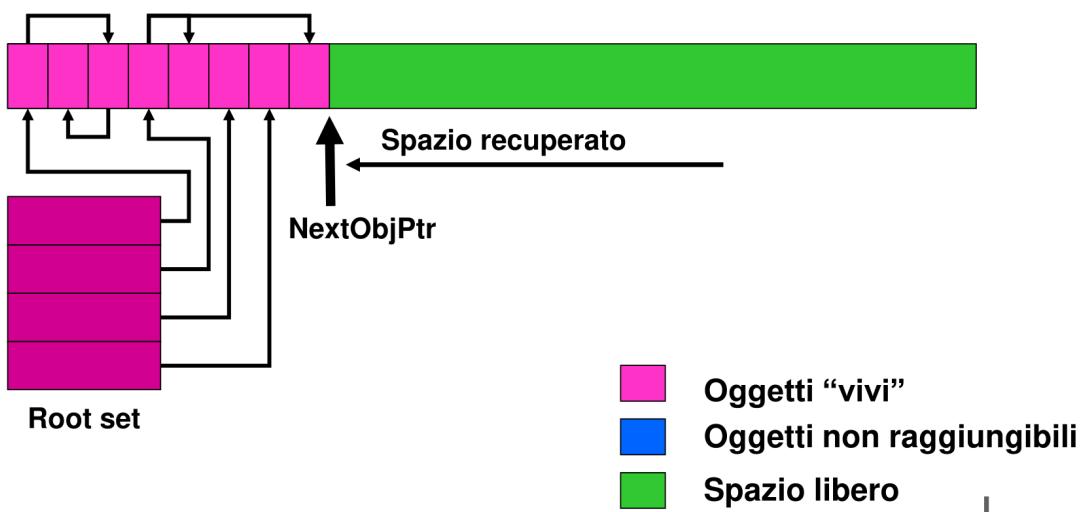
- Verifica se nell'heap esistono oggetti non più utilizzati dall'applicazione
- Ogni applicazione ha un insieme di radici (*root*)
- Ogni radice è un puntatore che contiene l'indirizzo di un oggetto di tipo riferimento oppure vale 'null'
- Le radici sono:
  - Variabili globali e field statici di tipo riferimento
  - Variabili locali o argomenti attuali di tipo riferimento sugli stack dei vari thread
  - Registri della CPU che contengono l'indirizzo di un oggetto di tipo riferimento
- **Gli oggetti "vivi"** sono quelli **raggiungibili** direttamente o indirettamente dalle radici
- **Gli oggetti garbage** sono quelli **NON raggiungibili** direttamente o indirettamente dalle radici
- Quando parte, il GC ipotizza che tutti gli oggetti siano garbage
- Quindi, scandisce le radici e per ogni radice **marca**
  - l'eventuale oggetto referenziato e
  - Tutti gli oggetti a loro volta raggiungibili a partire da tale oggetto
- Se durante la scansione incontra un oggetto già marcato in precedenza, lo salta
  - sia per motivi di prestazioni
  - sia per gestire correttamente riferimenti ciclici
- Una volta terminata la scansione delle radici, tutti gli oggetti NON marcati sono non raggiungibili e quindi garbage

#### 3.2.12.1 Fase 1: Mark



- Rilascia la memoria usata dagli oggetti non raggiungibili
- Compatta la memoria ancora in uso, modificando nello stesso tempo tutti i riferimenti agli oggetti spostati!
- Unifica la memoria disponibile, aggiornando il valore di `NextObjPtr`
- Tutte le operazioni che il GC effettua sono possibili in quanto
  - Il tipo di un oggetto è sempre noto
  - È possibile utilizzare i metadati per determinare quali field dell’oggetto fanno riferimento ad altri oggetti

### 3.2.12.2 Fase 2: Compact



### 3.2.13 Finalization

- Non è responsabilità del GC, ma del programmatore
- Se un oggetto contiene esclusivamente
  - tipi valore e/o
  - riferimenti a oggetti managed
- (maggior parte dei casi), non è necessario eseguire alcun codice particolare
- Se un oggetto contiene almeno un riferimento a un oggetto *unmanaged* (in genere, una risorsa del S.O.)

- file, connessione a database, socket, mutex, bitmap, ...
 

è necessario eseguire del codice per rilasciare la risorsa, prima della deallocazione dell'oggetto
- Ad esempio, un oggetto di tipo `System.IO.FileStream`
  - ▶ Prima deve aprire un file e memorizzare in un suo field l'handle del file (una risorsa di S.O. unmanaged)
  - ▶ Quindi usa tale handle nei metodi **Read** e **Write**
  - ▶ Infine, deve rilasciare l'handle nel metodo **Finalize**
- In C#
  - ▶ NON è possibile definire il metodo **Finalize**
  - ▶ È necessario definire un **distruttore** (sintassi C++)

```
public class OSHandle
{
    // Field contenente l'handle della risorsa unmanaged
    private readonly IntPtr handle;

    public IntPtr Handle
    { get { return _handle; } }

    public OSHandle(IntPtr handle)
    { _handle = handle; }

    ~OSHandle()
    { CloseHandle(_handle); }

    [System.Runtime.InteropServices.DllImport("Kernel32")]
    private extern static bool CloseHandle(IntPtr handle);
}
```

- Il compilatore C# trasforma il codice del distruttore

```
~OSHandle()
{ CloseHandle(_handle); }
```

nel seguente codice (ovviamente in IL):

```
protected override void Finalize()
{
    try
    { CloseHandle(_handle); }
    finally
    { base.Finalize(); }
}
```

- L'invocazione del metodo **Finalize** non avviene in modo deterministico
- Inoltre, non essendo un metodo pubblico, il metodo **Finalize** non può essere invocato direttamente
- Nel caso di utilizzo di risorse che devono essere rilasciate appena termina il loro uso, questa situazione è problematica
- Si pensi a file aperti o a connessioni a database che vengono chiusi solo quando il GC invoca il corrispondente metodo **Finalize**

- In questi casi, è di fondamentale importanza rilasciare (Dispose) o chiudere (Close) la risorsa in modo deterministico

### 3.2.14 Rilascio deterministico senza gestione delle eccezioni

```
...
Byte[] bytesToWrite = new Byte[] {1,2,3,4,5};
FileStream fs;
fs = new FileStream("Temp.dat", FileMode.Create);
fs.Write(bytesToWrite, 0, bytesToWrite.Length);
fs.Close();
...
```

### 3.2.15 Rilascio deterministico con gestione delle eccezioni

```
...
Byte[] bytesToWrite = new Byte[] {1,2,3,4,5};
FileStream fs = null;
try
{
    fs = new FileStream("Temp.dat", FileMode.Create);
    fs.Write(bytesToWrite, 0, bytesToWrite.Length);
}
finally
{
    if(fs != null) fs.Close();
}
...
...
```

### 3.2.16 Il pattern Dispose

- Se un tipo T vuole offrire ai suoi utilizzatori un servizio di clean up esplicito, deve implementare l'interfaccia IDisposable

```
public interface IDisposable
{
    void Dispose();
}
```

- I clienti di T possono utilizzare l'istruzione using

```
using (T tx = ...)
{
    utilizzo di tx... #Invocazione automatica di tx.Dispose()
}
```

### 3.2.17 Rilascio deterministico con using

```
...
Byte[] bytesToWrite = new Byte[] {1,2,3,4,5};
using (FileStream fs = new FileStream("Temp.dat", FileMode.Create))
{
    fs.Write(bytesToWrite, 0, bytesToWrite.Length);
}
...
...
```

- Il tipo della variabile definita nella parte iniziale di ‘using’ deve implementare l'interfaccia IDisposable
- All'uscita del blocco ‘using’ , viene sempre invocato

automaticamente il metodo Dispose

### 3.2.18 Il pattern Dispose (altro esempio di utilizzo)

```
public class CursorReplacer : IDisposable
{
    private readonly Cursor _previous;
    public CursorReplacer()
    {
        _previous = Cursor.Current;
        Cursor.Current = Cursors.WaitCursor;
    }
    public void Dispose()
    {
        Cursor.Current = _previous;
    }
}

List<DbTableWrapper> tableWrappers = new List<DbTableWrapper>();
// Recupero di tutte le tabelle selezionate
using (CursorReplacer cursorReplacer = new CursorReplacer())
{
    foreach (DbServerWrapper serverWrapper in SelectedDbServerWrappers)
        foreach (DbCatalogWrapper catalogWrapper in
serverWrapper.SelectedDbCatalogWrappers)
            foreach (DbTableWrapper tableWrapper in
catalogWrapper.SelectedDbTableWrappers)
                {
                    tableWrappers.Add(tableWrapper);
                }
}
```