UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

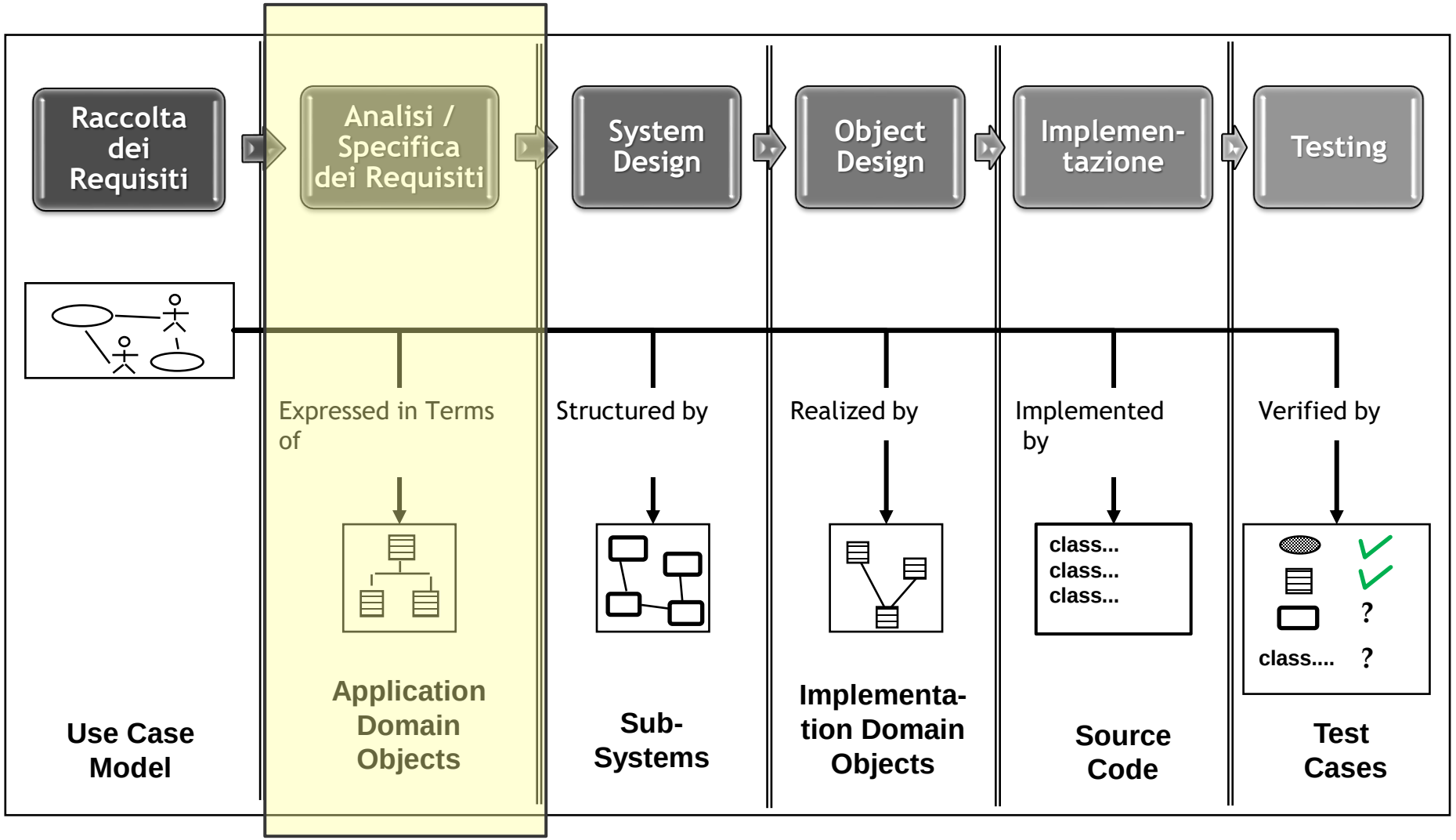
Ingegneria del Software – I Modelli di Dominio

Prof. Sergio Di Martino

Verso la progettazione…

- Una delle maggiori difficoltà è “tradurre” i requisiti software in un progetto di sistema implementabile
- Obiettivo della fase successiva all’analisi dei requisiti è di iniziare ad individuare le classi, il loro comportamento dinamico, e le relazioni che vi intercorrono.
- Obiettivo della lezione: Definire i Modelli di Dominio
  - Identificazione degli oggetti
  - Definizione del comportamento degli oggetti
  - Definizione delle relazioni tra gli oggetti
  - Classificazione degli oggetti
  - Organizzazione degli oggetti

Ciclo di Vita del Software

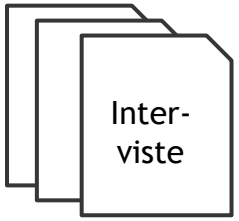


Documento dei requisiti SW (da Lezione precedente)

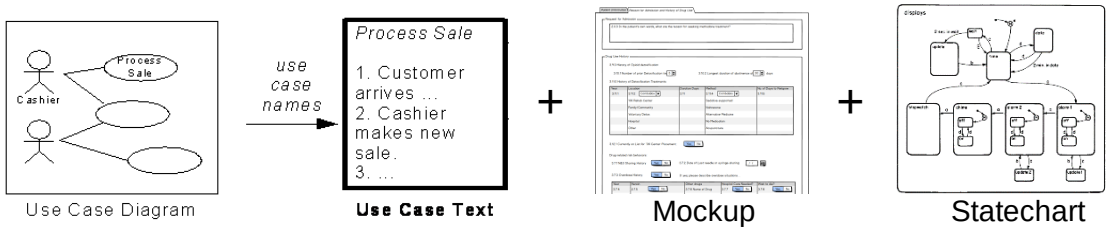
- Si focalizza sulla descrizione del sistema da sviluppare
- Cliente, utenti e sviluppatori contribuiscono alla stesura del documento di specifica dei requisiti
  - Può essere usato come contratto tra cliente e sviluppatori
- Il documento prevede vari livelli di raffinamento, dal linguaggio naturale, al linguaggio strutturato, ai modelli UML.
  - Tutti i documenti rappresentano la stessa informazione ma sono scritti usando linguaggi diversi, per utenti diversi

Requirement Engineering

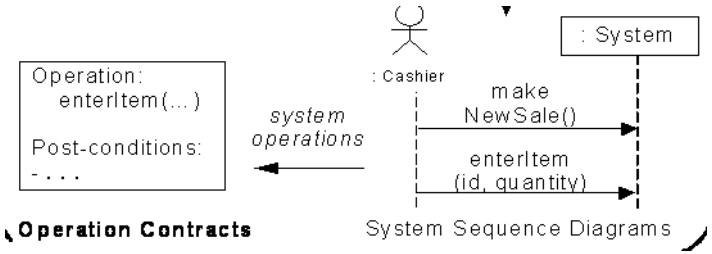
1. **Requirement Elicitation**

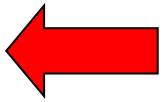


2. **Requirement Analysis**



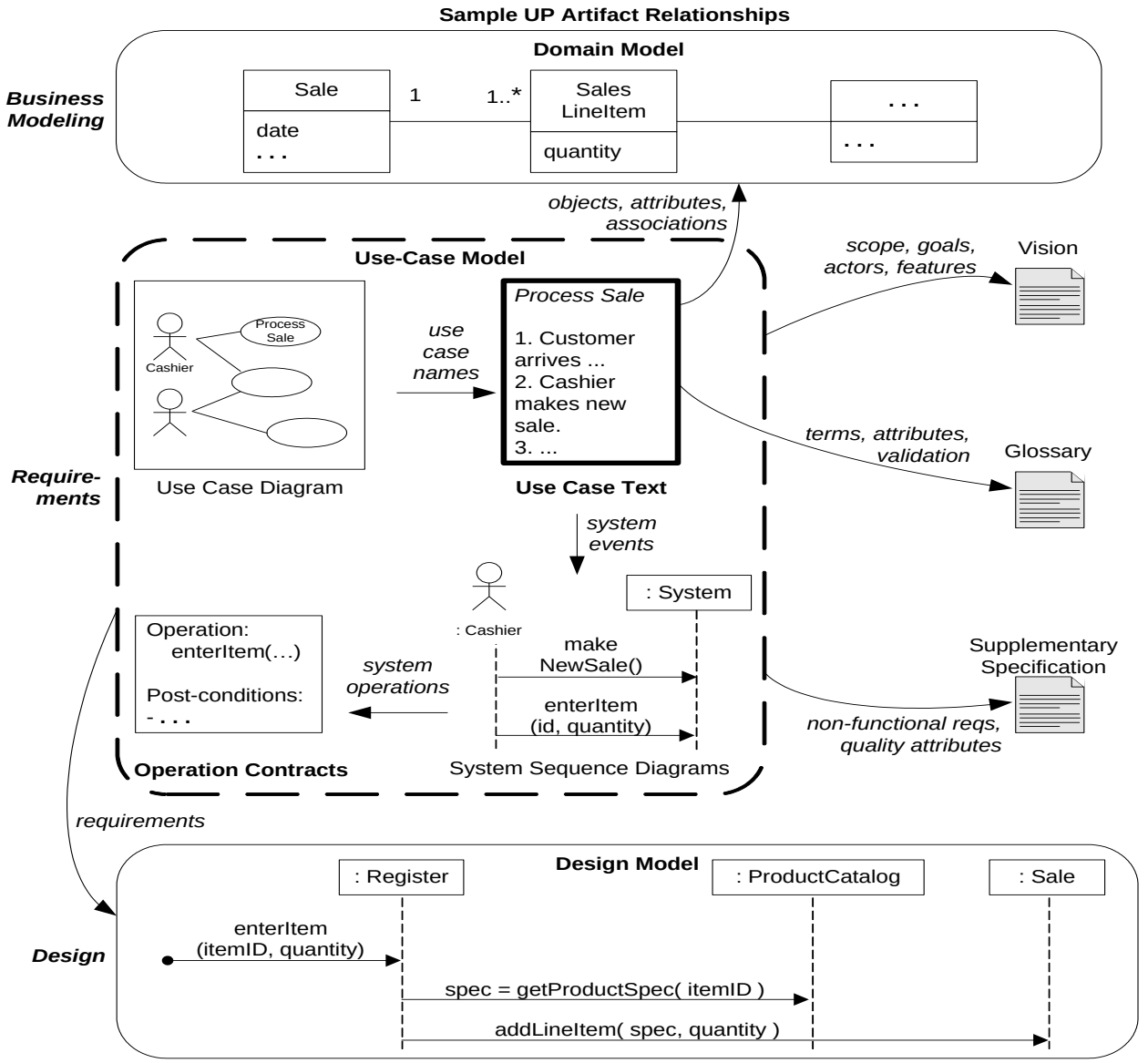
3. **Requirement Specification**





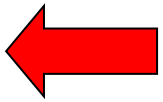
4. **Requirement Validation**

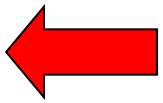
Artefatti coinvolti



Il Documento di Specifica dei Requisiti

- E’ composto da tre modelli:
  - **Modello Funzionale**, rappresentato da use cases
    - Possibile contratto col cliente, è scritto per i “profani”
  - **Modello di Dominio**, rappresentato dal diagramma delle classi
  - **Modello Dinamico**, rappresentato da sequence diagrams (opzionali statechart e activity diagrams)
- Gli use case prodotti nella fase di raccolta dei requisiti vengono raffinati per derivare il Modello di Dominio e il Modello Dinamico, primo passo per la successiva Progettazione del sistema





Il modello di Dominio

- E’ il più importante modello della fase di Specifica dei Requisiti
- E’ una rappresentazione visuale di classi concettuali, relative al dominio del problema
  - Si esprime attraverso un class diagram, con classi, attributi e operazioni
- Si focalizza sui concetti che sono manipolati dal sistema, le loro proprietà e le relazioni

Il modello di Dominio (2)

- E’ un **dizionario visuale** dei concetti principali relativi al dominio
- E’ detto anche **Modello a Oggetti di Analisi**
- E’ una rappresentazione di oggetti del mondo reale in uno specifico dominio, NON di oggetti software
  - NB: sia il modello dinamico che il modello ad oggetti rappresentano concetti a livello utente, non a livello di componenti e classi software
  - Le classi di analisi rappresentano astrazioni che saranno dettagliate e/o raffinate successivamente

Il modello dinamico

- Si focalizza sul comportamento del sistema
  - I sequence diagram rappresentano le interazioni tra un insieme di oggetti durante un singolo use case
  - Gli statechart rappresentano il comportamento di un singolo oggetto o di alcuni oggetti strettamente accoppiati
- Consente di assegnare le responsabilità alle classi e quindi individuare nuove classi che sono aggiunte al modello ad oggetti di analisi, in un procedimento iterativo

I Class Diagram in Requirement Engineering

UML Class Diagram

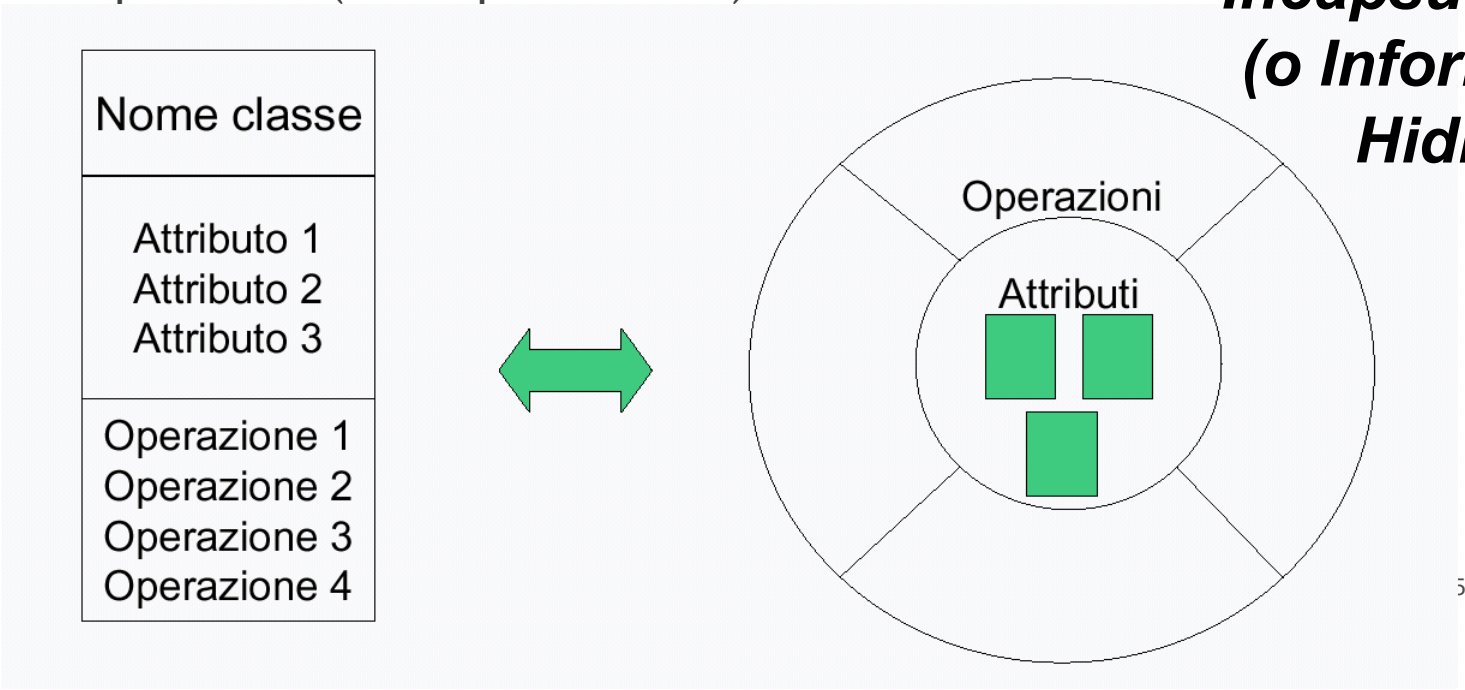
- Descrive:
  - Classi
  - Relazioni tra classi
    - associazione (uso)
    - generalizzazione (ereditarietà)
    - aggregazione (contenimento)
- Definisce la visione **statica** del sistema
- È il modello principe di UML, perché definisce gli elementi base del sistema sw da sviluppare.

Livello di Astrazione

- Un class diagram può essere definito/utilizzato a livello:
  - concettuale (o di Requisiti)
    - studia i concetti propri del dominio sotto studio, senza preoccuparsi della loro successiva implementazione
    - E’ una sorta di Dizionario Visuale del dominio
  - di Specifica Implementativa
    - studia il software da un punto di vista implementativo, specificando come va sviluppato il sistema
    - E’ un raffinamento del precedente

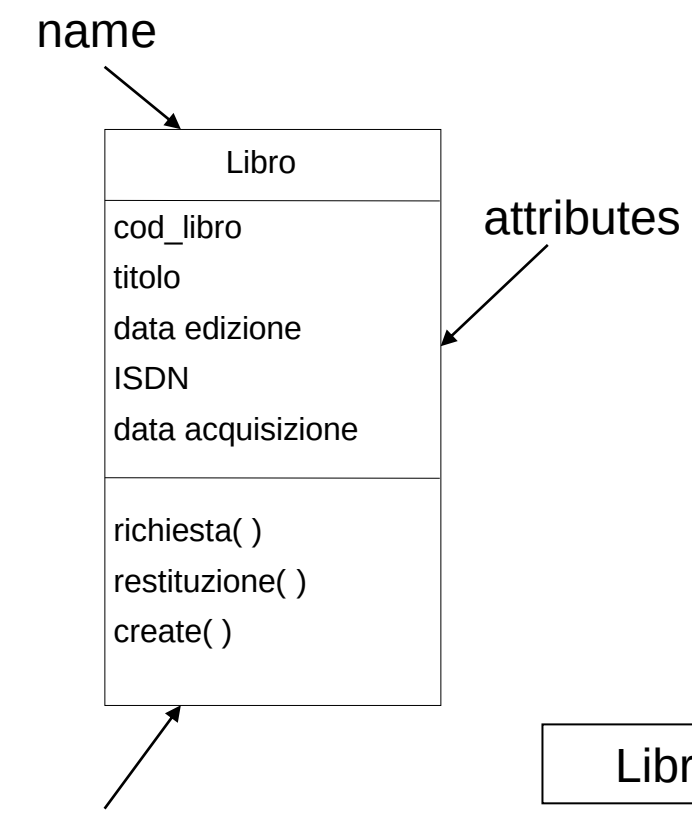
Classi in UML

- In UML una classe è composta da tre parti
  - nome
  - attributi (lo stato)
  - metodi o operazioni (il comportamento)

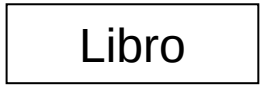


Incapsulamento (o Information Hiding)

Nomi di Classi

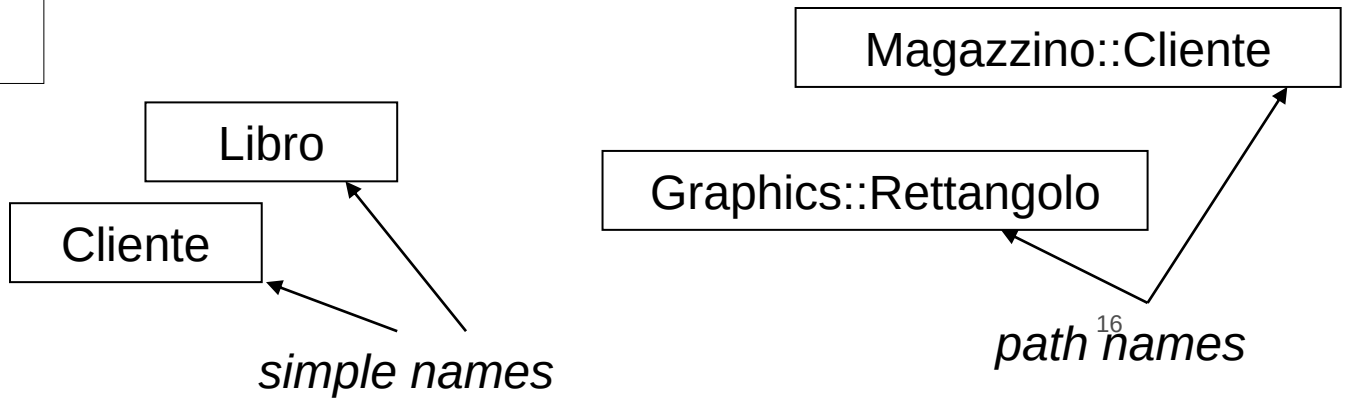


Una classe può essere rappresentata anche usando solo la sezione del nome



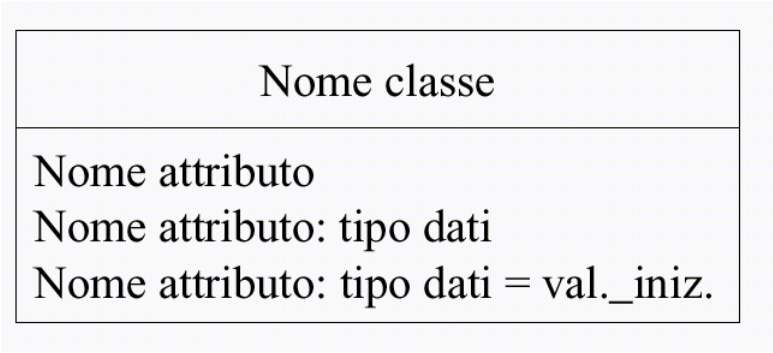
Un nome può essere un:

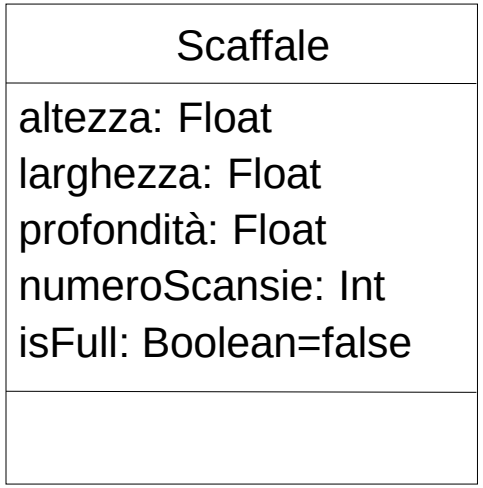
- *simple name*, il solo nome della classe
- *path name*, il nome della classe preceduto dal nome del package in cui la classe è posta



Attributi

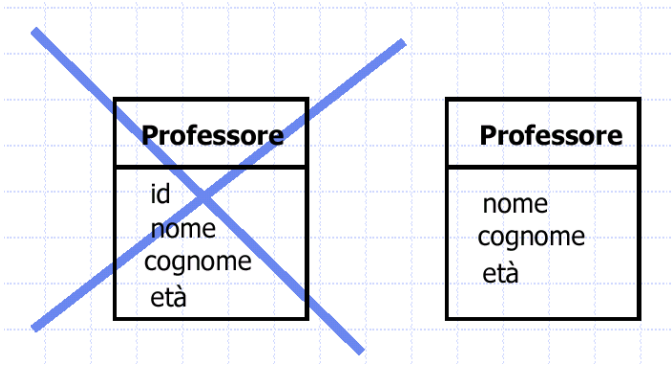
- In UML, per ciascun attributo si può specificare il tipo, la visibilità ed un valore iniziale
- Tipicamente il nome di un attributo è composto da una o più parole
  - usare il maiuscolo per la prima lettera di ciascuna parola, lasciando minuscola la lettera iniziale del nome (Camel Case)





Individuazione degli attributi

- Gli oggetti hanno implicitamente una loro identità, non bisogna aggiungerla



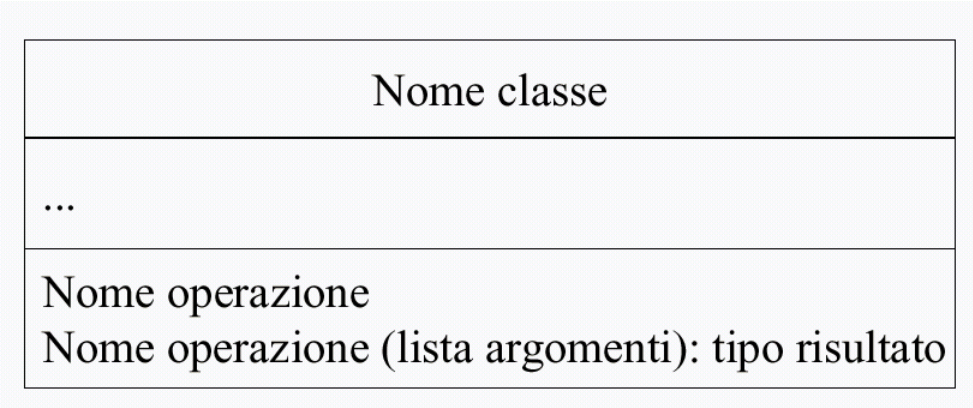
- Candidati per attributi
  - Sostantivi che non sono diventati classi
- Conoscenza del dominio applicativo
  - Persona (ambito bancario)
    - nome, cognome, codiceFiscale, numeroConto
  - Persona (ambito medico)
    - nome, cognome, allergie, peso, altezza

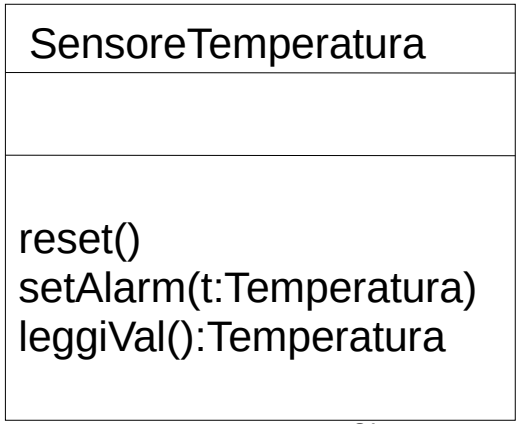
Operazioni

- Un’operazione è un’azione che un oggetto esegue su un altro oggetto e che determina una reazione
  - Le operazioni operano sui dati incapsulati dell’oggetto
- Tipi di operazione
  - Selettore (query): accedono allo stato dell’oggetto senza alterarlo (es. “lunghezza” della classe coda)
  - Modificatore: alterano lo stato di un oggetto (es. “append” della classe coda)
- Operazioni di base per una classe di oggetti (realizzate con modalità diverse a seconda dei linguaggi)
  - Costruttore: crea un nuovo oggetto e/o inizializza il suo stato
  - Distruttore: distrugge un oggetto e/o libera il suo stato

Operazioni

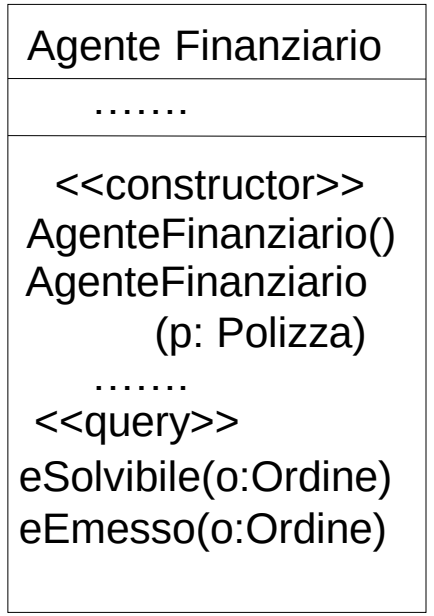
- Per ciascuna operation si può specificare il solo nome o la sua signature, indicando il nome, il tipo, parametri e, in caso di funzione, il tipo ritornato
- Stesse convenzioni dette per gli attributi
(Camel Case)





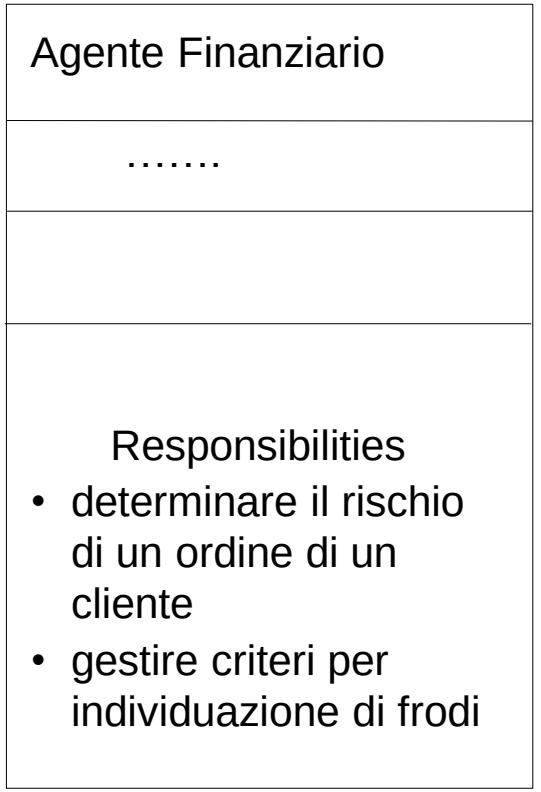
Attributi e Operazioni

- Attributi e Operazioni di una classe non devono obbligatoriamente essere descritti tutti subito
  - Attributi ed operazioni possono essere mostrati solo parzialmente, elidendo la classe
  - Per indicare che esistono più attributi/operazioni di quelli mostrati si usano i punti sospensivi “ …….”
- Per meglio organizzare lunghe liste di attributes/operations raggrupparli insieme in categorie usando stereotipi quali <<constructor>>, <<query>>, <<update>>



Responsibility

- Una responsibility è un contratto o una obbligazione di una classe
  - Questa è definita dallo stato e comportamento della classe
  - Una classe può avere un qualsiasi numero di responsabiltà, ma una classe ben strutturata ha una o poche responsabiltità
- Le responsibilità possono essere indicate, in maniera testuale, in una ulteriore sezione, sul fondo della icona della class
  - Possono anche essere specificate in linguaggi formali, come OCL (Object Constraint Language)



Vincoli aggiuntivi

- Talvolta è necessario o conveniente esplicitare dei vincoli aggiuntivi in modo da rendere più comprensibile il diagramma delle classi.
  - {breve descrizione in linguaggio naturale o in linguaggi formali (come OCL)}

Studente Nuovo Ordinam.
Nome Cognome Matricola Corso di Laurea
if Corso di Laurea = "Informatica" then Matricola = N86

Visibilità di features

- E’ possibile specificare la visibilità di attributi e operazioni
- In UML è possibile specificare tre livelli di visibilità:
  - **+ (public):** qualsiasi altra classe con visibilità alla classe data può usare l’attributo/operazione (di default se nessun simbolo è indicato)
  - **# (protected):** qualsiasi classe discendente della classe data può usare l’attributo/operazione
  - **- (private):** solo la classe data può usare l’attributo/operazione

Stringhe di Proprietà

- E’ possibile definire una “stringa di proprietà” per specificare particolari caratteristiche delle features di una classe
- Si indica tra parentesi graffe affianco alla definizione della feature
- Per gli attributi, property-string può assumere uno dei seguenti 3 valori:
  - **changeable**: nessuna limitazione per la modifica del valore dell’attributo (default)
  - **addOnly**: per attributi con molteplicità maggiore di 1 possono essere aggiunti ulteriori valori, ma una volta creato un valore non può essere né rimosso né modificato
  - **readonly**: il valore non può essere modificato.
    - **frozen** (deprecato in UML2)

Proprietà di features

- Per i metodi, property-string può assumere uno dei seguenti valori:
  - **isQuery**: l'esecuzione dell'operazione lascia lo stato del sistema immutato
  - **sequential**: i chiamanti devono coordinare l'oggetto dall'esterno in modo che vi sia un sol flusso per volta verso l'oggetto; in presenza di più flussi di controllo, non è garantita la semantica e l'integrità dell'oggetto
  - **guarded**: la semantica e l'integrità dell'oggetto è garantita in presenza di flussi di controllo multipli dalla sequenzializzazione di tutte le chiamate a tutte le operation guarded dell'oggetto;
  - **concurrent**: la semantica e l'integrità dell'oggetto è garantita in presenza di flussi di controllo multipli trattando la operation come atomica. Chiamate multiple da flussi di controllo concorrente possono presentarsi contemporaneamente ad un oggetto o ad una sua operation concurrent ed essere eseguite correttamente con corretta semantica
- Le ultime tre proprietà riguardano la concorrenza di una operation e sono rilevanti solo in presenza di più processi o threads

Sintassi finale di features

- Attributi:
  - visibilità nome: tipo molteplicità = default {property-string}
  - Es: - titolo : String [1] = “Sw Engineering” {ReadOnly}

- Metodi:
  - visibilità nome (elenco parametri): tipo di ritorno{property-string}
  - Es: + getSaldo(data: Date): double {isQuery}

Relazioni tra classi

Relazioni tra classi

- In UML, le interconnessini tra oggetti/classi sono rappresentate attraverso “relazioni”

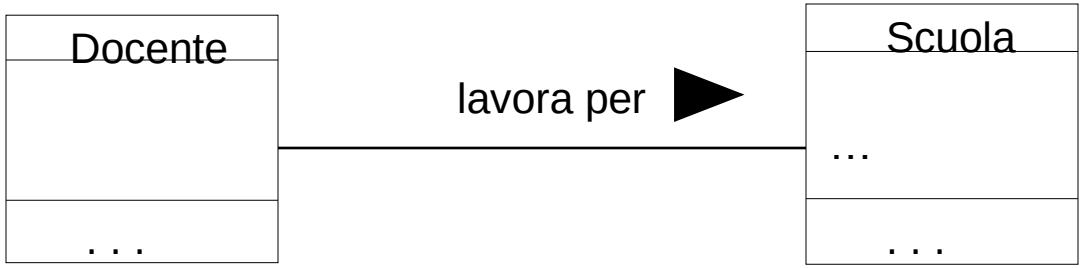
- UML definisce 3 tipi di relazioni:
  - Associazioni
  - Generalizzazioni/Specializzazioni
  - Dipendenze (raro in Analisi)

Associazioni

- Un sistema O-O è costituito da classi che collaborano tra loro scambiandosi messaggi
- Quando è in esecuzione un sistema O-O è popolato da istanze di classi
- Quando un'istanza di classe passa messaggi a un'altra classe si sottintende l'esistenza di un'associazione tra le due classi

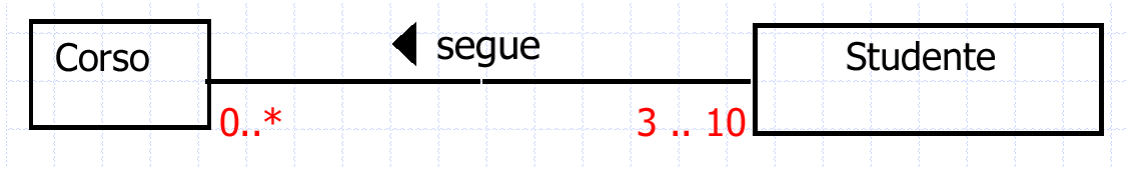
Associazioni

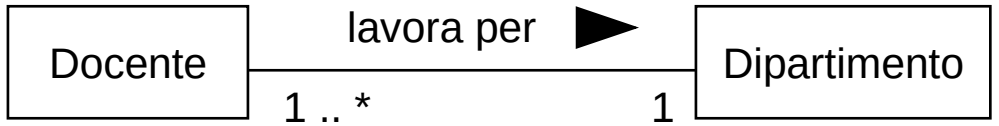
- Indicano relazioni tra classi
- Possono essere Riflessive
- Adornments:
  - Nome: non obbligatorio; può avere un verso di lettura
  - Ruolo: per ciascuna classe coinvolta
  - Cardinalità: come negli E-R
- Aggregazioni: un tipo particolare di associazione che specifica un rapporto aggregato-componente
- Composizioni: aggregazione in cui ogni componente partecipa ad un solo aggregato



Molteplicità delle associazioni

- La molteplicità dice
  - Se l'associazione è obbligatoria oppure no
  - Il numero minimo e massimo di oggetti che possono essere relazionati ad un altro oggetto





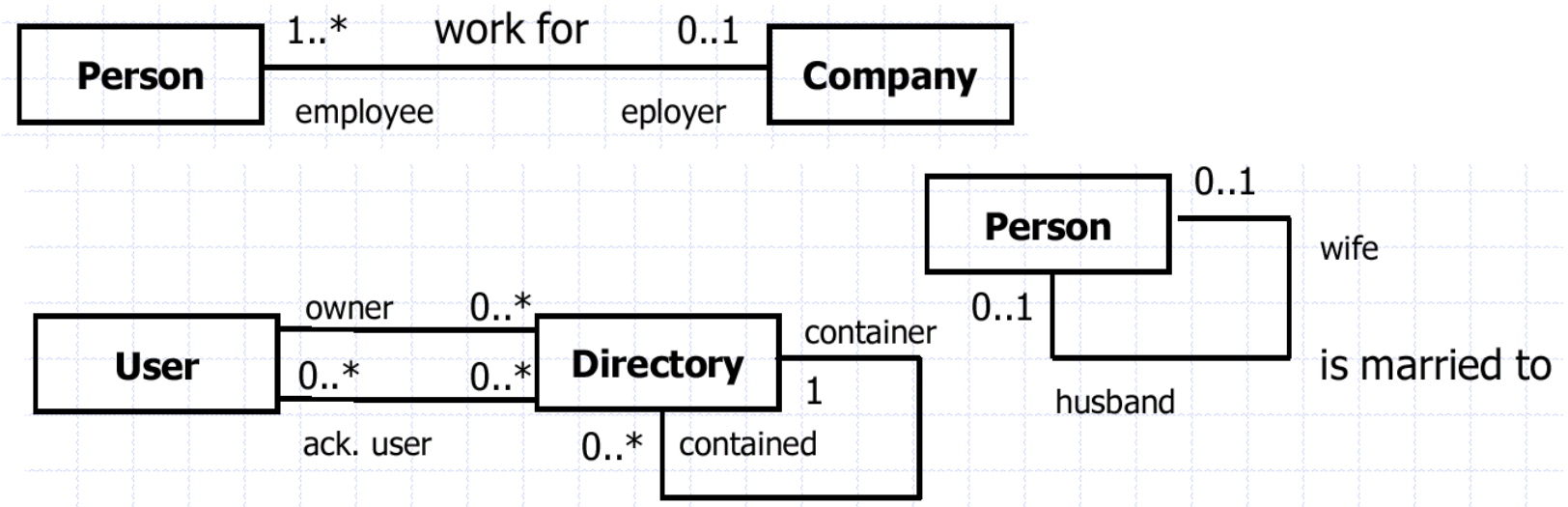
Molteplicità delle associazioni

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

Nota: Uml 1.x permetteva anche m..n, con m e n > 1

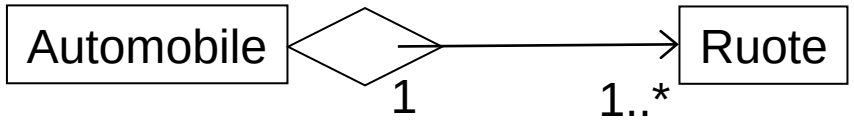
Ruoli

- I ruoli forniscono una modalità per attraversare relazioni da una classe ad un'altra
  - Chiariscono il ruolo giocato da una classe all'interno di un'associazione
  - I nomi di ruolo possono essere usati in alternativa ai nomi delle associazioni
- I ruoli sono spesso usati per relazioni tra oggetti della stessa classe (associazioni riflessive)



Aggregazioni

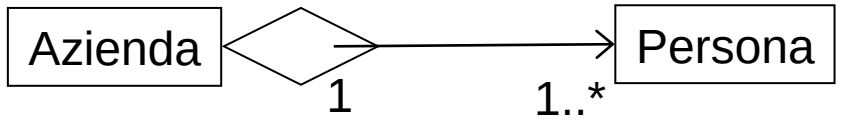
- La relazione di aggregazione è un'associazione speciale che aggrega gli oggetti di una classe componente in un unico oggetto della classe
  - la si puo' leggere come "è fatto da" in un verso e "è parte di" nell'altro verso



- Proprietà
  - transitività: se A è parte di B e B è parte di C allora A è parte di C
  - antisimmetria: se A è parte di B allora B non è parte di A
  - Indipendenza: un oggetto contenuto può sopravvivere senza l'oggetto contenente

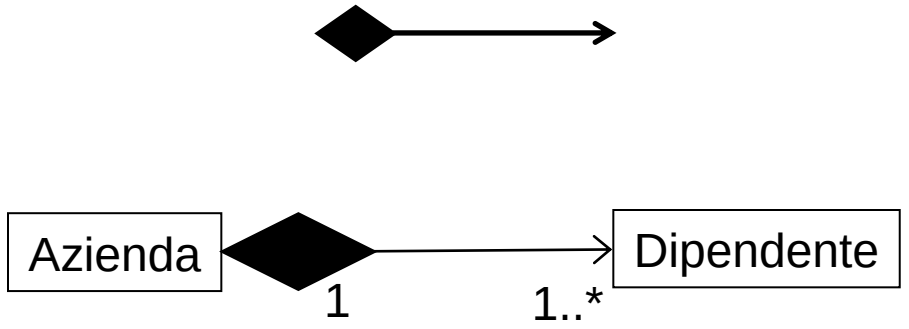
Aggregazione

- Aggregazione: è la relazione “parte di”
  - Es: un’azienda ha delle persone che vi lavorano. I vari oggetti “persona” continuano ad avere dignità ed esistenza propria anche al di là dell’oggetto azienda.
- La distruzione dell’oggetto “azienda” non comporta automaticamente quella delle sue parti



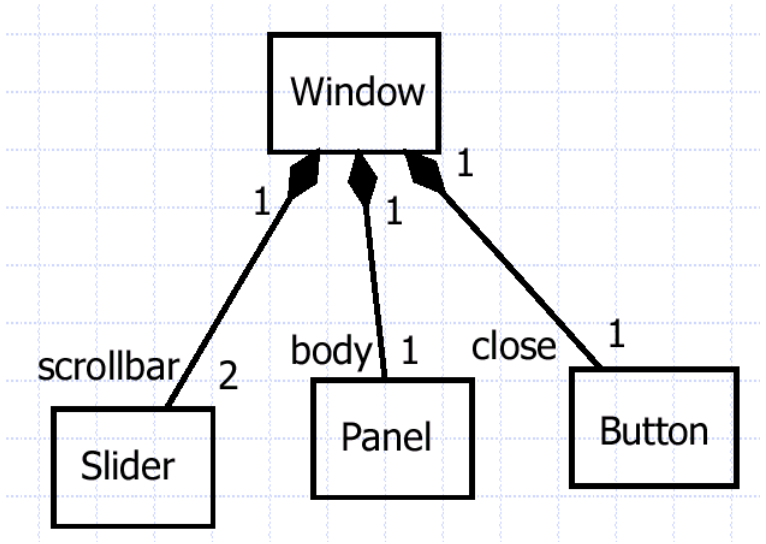
Composizione

- Composizione: è una relazione più forte, l'oggetto parte appartiene ad un solo tutto e le parti hanno lo stesso ciclo di vita dell'insieme.
  - All'atto della distruzione dell'oggetto principale si ha la propagazione della distruzione agli oggetti parte.
  - Es: un'azienda ha dei dipendenti che vi lavorano. In tal caso, a differenza dell'esempio precedente, non ha senso che i vari oggetti "dipendente" abbiano vita propria senza un oggetto "azienda" associato

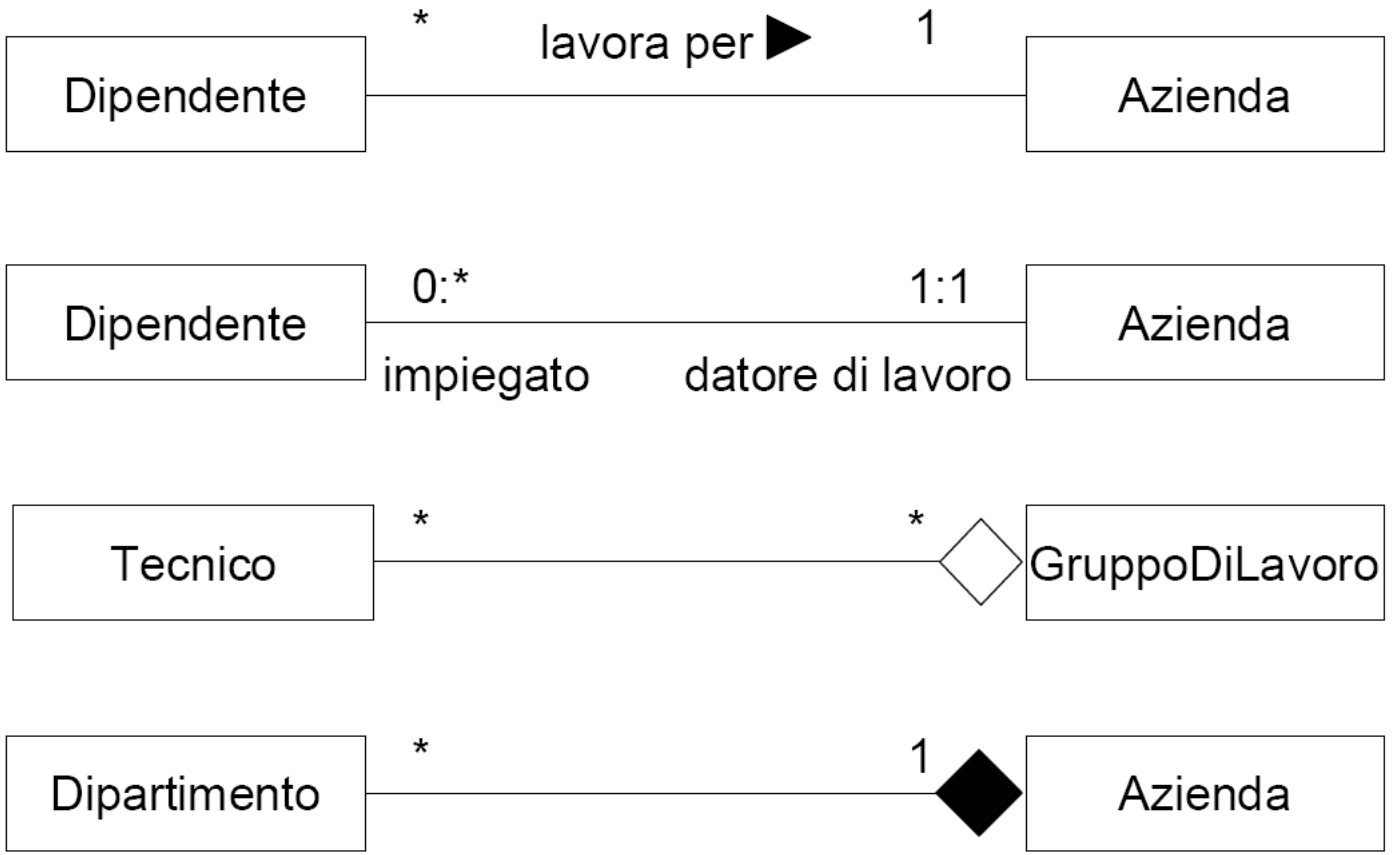


Composizione

- Una relazione di composizione è un'aggregazione forte
  - Le parti componenti non esistono senza il contenitore
  - Ciascuna parte componente ha la stessa durata di vita del contenitore
  - Una parte può appartenere ad un solo tutto per volta

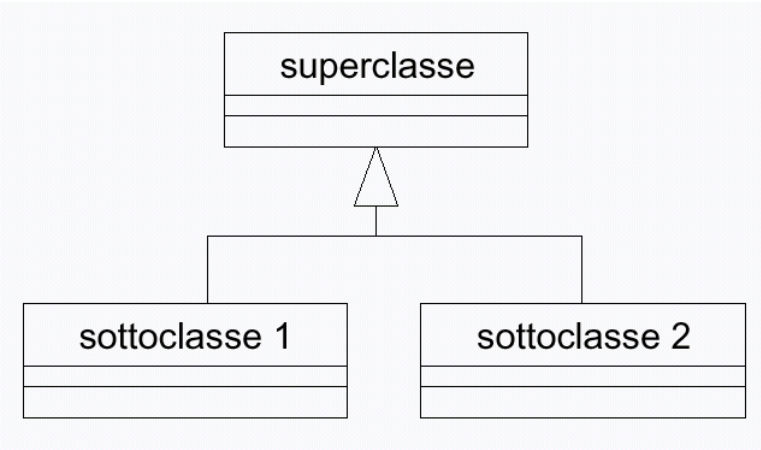


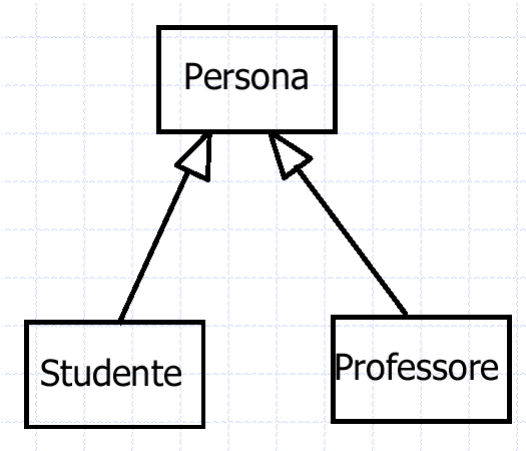
Riassunto Associazioni



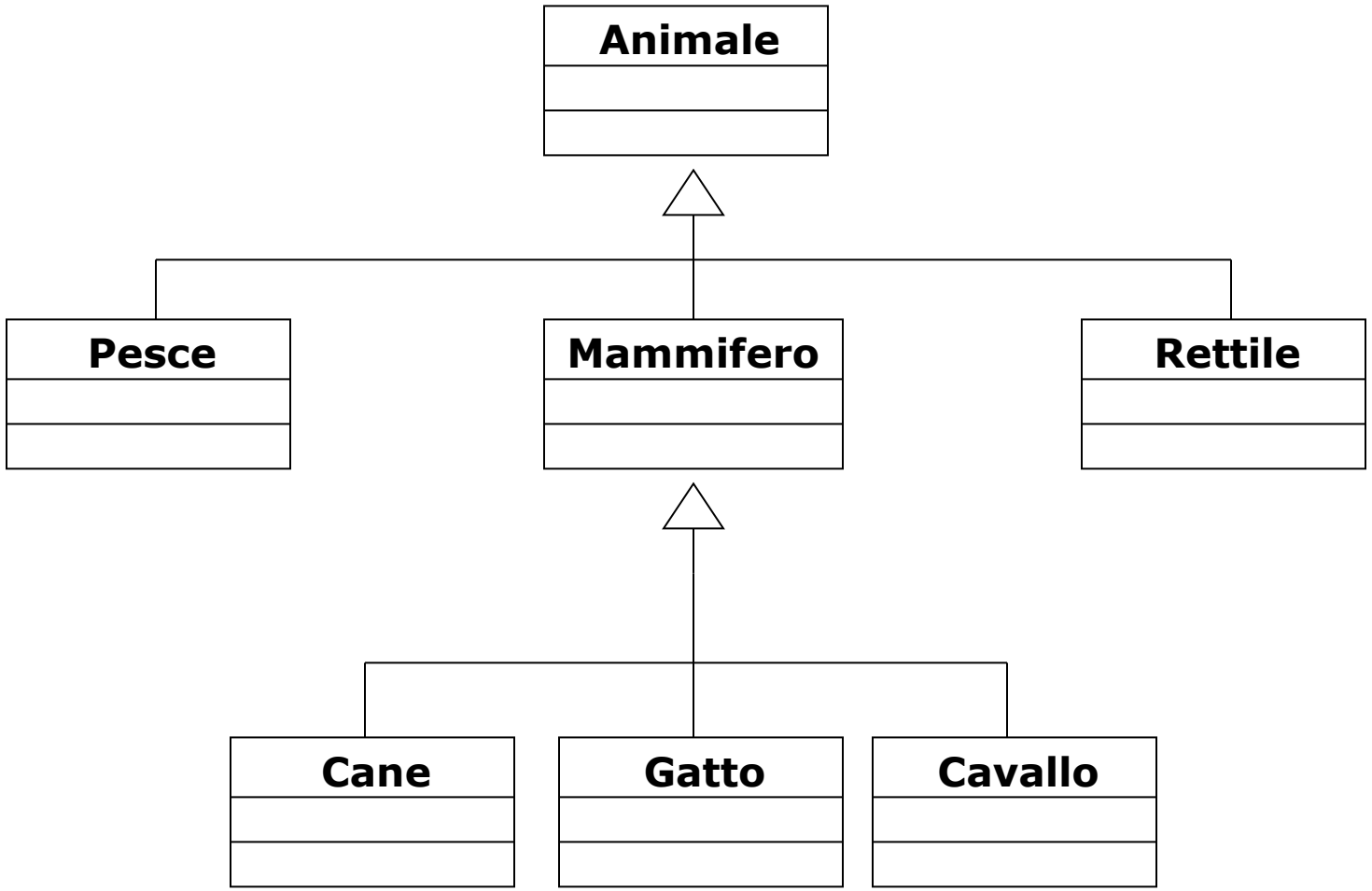
Ereditarietà

- La relazione di generalizzazione rappresenta una tassonomia delle classi
- Può essere letta come
  - “e’ un tipo di” (verso di generalizzazione)
  - “puo’ essere un” (verso di specializzazione)
- Ogni oggetto di una sottoclasse è anche un oggetto della sua superclasse





Generalizzazione fra classi (es.)



Identificazione degli oggetti

Identificare gli Oggetti

- La fase fondamentale dello sviluppo del software Object Oriented è quella dell'analisi (OOA Object Oriented Analysis) e della progettazione (OOD Object Oriented Design), perché sono queste fasi che garantiscono il successo e il raggiungimento degli obiettivi dell'OOP
- Per individuare gli oggetti partecipanti si esamina la descrizione di ogni use case e si individuano le classi candidate
- E’ un procedimento altamente creativo e soggettivo
- Può essere guidato da 2 Euristiche:
  - Three-Object-Type → Classificazione Oggetti in Entity, Boundary e Control
    - Abbott → Identificazione oggetti Entity

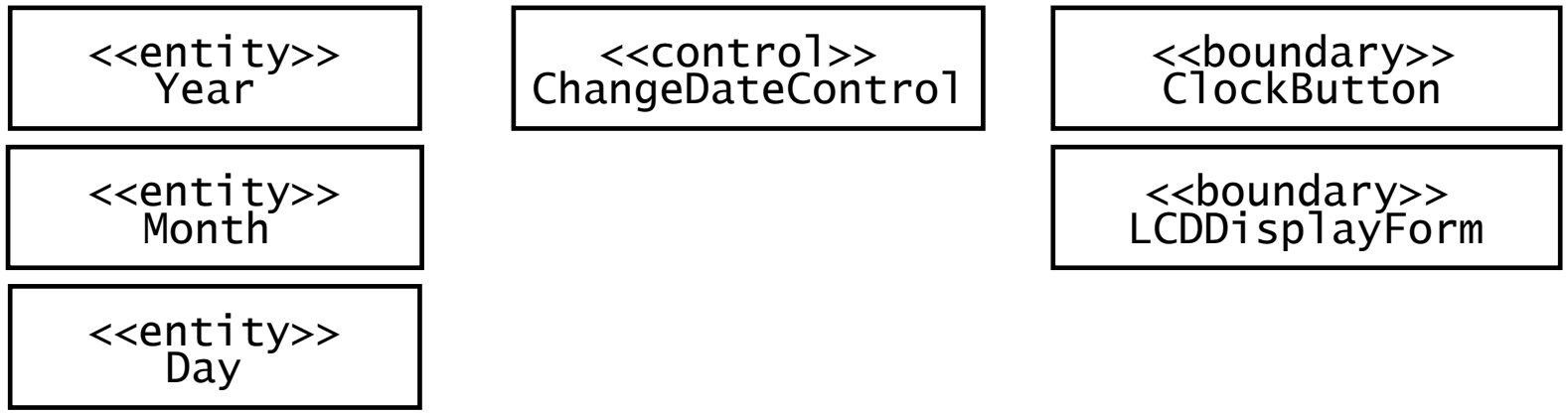
Euristica three-object-type

- Secondo tale euristica, il modello ad oggetti di analisi raggruppa gli oggetti in *Entity*, *Boundary* e *Control*:
  - Gli oggetti **Entity** modellano l'informazione persistente
  - Gli oggetti **Boundary** modellano le interazioni tra gli attori e il sistema
  - Gli oggetti **Control** modellano la logica che si occupa di realizzare gli use case

- L'approccio three-object-type porta a modelli che sono più flessibili e facili da modificare:
  - L'interfaccia al sistema (rappresentata da oggetti boundary) è più soggetta a cambiamenti rispetto alle funzionalità (rappresentate da oggetti entity e control)

Regole di Naming

- UML fornisce il meccanismo degli stereotipi per consentire di aggiungere tale meta-informazione agli elementi di modellazione
- E' opportuno usare convenzioni sui nomi:
  - Gli oggetti control possono avere il suffisso Control
  - Gli oggetti boundary dovrebbero avere nomi che ricordano aspetti dell'interfaccia (es. suffisso Form, Button, ecc)



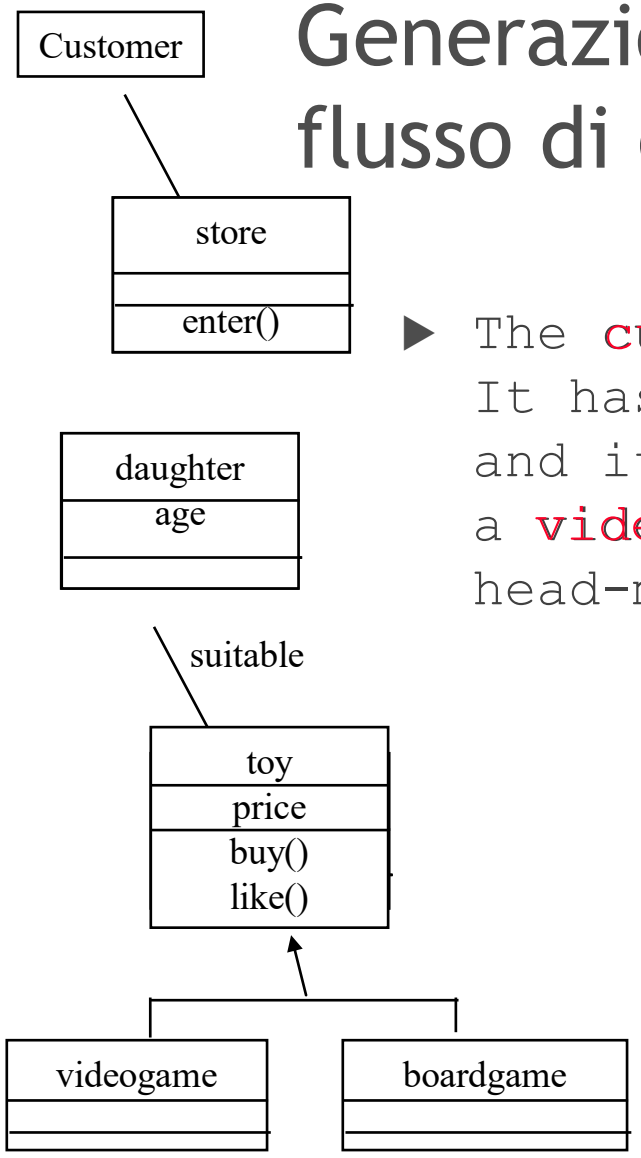
Identificare gli Oggetti Entity e l’euristica di Abbott

- Gli oggetti Entity rappresentano i concetti del dominio.
- La loro individuazione può essere facilitata dall’uso dell’Euristica di Abbott
- L’euristica di Abbott si basa sull’analisi linguistica per identificare oggetti, attributi, associazioni dai requisiti di sistema
  - mappano parti delle parole (nomi, verbo avere, verso essere, aggettivi) per modellare componenti (oggetti, operazioni, relazioni di ereditarietà, classi)

Attività di Analisi: Euristica di Abbott

Parti del parlato	Componente del modello	esempio
Nome proprio	Istanza	Alice
Nome comune	Classe	Agente di Polizia (FieldOfficer)
Verbo fare	Operazione	Crea, Submit, Select
Verbo essere	gerarchia	È un tipo di, è uno di
Verbo avere	aggregazione	Ha, consiste di, include
Verbo modale	vincoli	Deve essere
Aggettivo	attributo	Descrizione dell'incidente

Generazione di un class diagram da un flusso di eventi



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

Euristica di Abbott

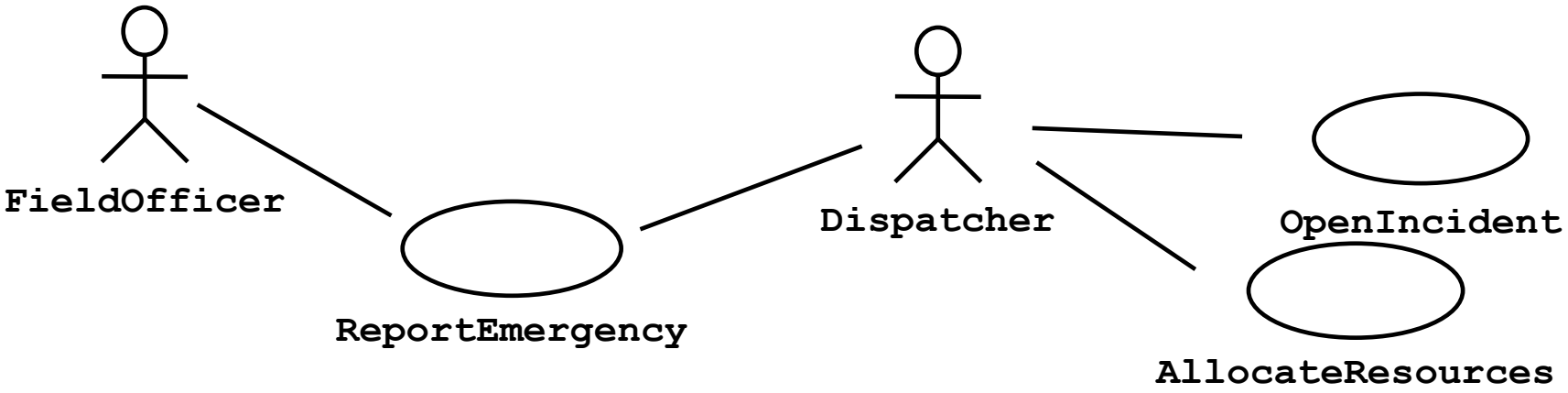
- Vantaggi:
  - Ci si focalizza sui termini dell'utente
- Svantaggi:
  - Il linguaggio naturale è impreciso, anche il modello ad oggetti derivato rischia di essere impreciso
  - La qualità del modello dipende fortemente dallo stile di scrittura dell'analista
  - Ci possono essere molti più sostantivi delle classi rilevanti, corrispondenti a sinonimi o attributi

Identificare gli Oggetti Boundary

- Gli oggetti Boundary rappresentano l'interfaccia del sistema con gli attori
  - In ogni use case, ogni attore (sia esso umano o sw già esistente) interagisce almeno con un oggetto Boundary
  - L'oggetto Boundary colleziona informazioni dall'attore e le traduce in una forma che può essere usata sia dagli oggetti Control che dagli oggetti Entity
  - Uno o più oggetti Boundary effettuano anche il processo inverso.

- Gli oggetti Boundary modellano l'interfaccia senza descriverne gli aspetti visuali!
  - Non ha senso parlare di "item di menu" o "scroll bar"
  - Lo sviluppo dell'interfaccia è solitamente di tipo prototipale, e iterativo

Modello di Caso d’Uso per la gestione di Incidenti



Esempio: ReportEmergency

Nome Use Case	ReportEmergency
Partecipanti	Inizializzato dal *FieldOfficer* Comunica con il *Dispatcher*

Flusso degli eventi

Attori	Sistema
Il FieldOfficer attiva la funzione “ReportEmergency” dal suo terminale	
	Il sistema risponde presentando un form al FieldOfficer
Il FieldOfficer completa il form selezionando il livello di emergenza, il tipo, la località, e una breve descrizione della situazione. Il FieldOfficer descrive anche possibili risposte alla situazione di emergenza. Quando il form è completo, il FieldOfficer sottomette il form	
	Il sistema riceve il form e notifica il Dispatcher
Il Dispatcher rivede le informazioni sottomesse e crea un Incident invocando lo use case OpenIncident. Il Dispatcher seleziona una risposta e comunica il report	
	Il sistema visualizza il report e la risposta selezionata per il FieldOfficer

Identificazione degli Oggetti Entity

- Dall'esame dello use case ReportEmergency, dalla conoscenza del dominio e dalle interviste al cliente è possibile identificare i seguenti oggetti
  - Dispatcher
  - FieldOfficer
  - Incident
  - EmergencyReport

- Si noti che EmergencyReport non è nominato esplicitamente nello use case:
  - nel passo 4 si nomina "informazione sottomessa dal FieldOfficer "
  - e dal colloquio con il cliente si deduce che è proprio ciò che solitamente è detto "emergency report"

Euristiche per Identificare gli Oggetti Boundary

- Identificare i controlli della UI di cui l'utente ha bisogno per iniziare lo use case (ReportEmergencyButton)
- Identificare form di cui l'utente ha bisogno per inserire dati nel sistema (ReportEmergencyForm)
- Identificare avvisi e messaggi che il sistema usa per rispondere all'utente (AcknowlegmentNotice)
- Non modellare aspetti visuali della UI con oggetti Boundary (meglio usare i mock-up per questo)
- Usare sempre i termini dell'utente finale per descrivere l'interfaccia, non usare termini del dominio di implementazione

Esempio: ReportEmergency

Nome Use Case	ReportEmergency
Partecipanti	Inizializzato dal *FieldOfficer* Comunica con il *Dispatcher*

Flusso degli eventi

Attore	Sistema
Il FieldOfficer attiva la funzione “ReportEmergency” dal suo terminale	
	Il sistema risponde presentando un form al FieldOfficer
Il FieldOfficer completa il form selezionando il livello di emergenza, il tipo, la località, e una breve descrizione della situazione. Il FieldOfficer descrive anche possibili risposte alla situazione di emergenza. Quando il form è completo, il FieldOfficer sottomette il form	
	Il sistema riceve il form e notifica il Dispatcher
Il Dispatcher rivede le informazioni sottomesse e crea un Incident invocando lo use case OpenIncident. Il Dispatcher seleziona una risposta e comunica il report	
	Il sistema visualizza il report e la risposta selezionata per il FieldOfficer

Identificazione degli Oggetti Entity dallo use case ReportEmergency

Dispatcher	Agente di polizia che gestisce *Incidenti*. Un *Dispatcher* apre, documenta e chiude *Incident* in risposta al Report di Emergenza e ad altre comunicazioni con *FieldOfficers*. I *Dispatcher* sono identificati dal numero del badge
EmergencyReport	Report iniziale su un *Incident* inviato da un *FieldOfficer* a un *Dispatcher*. Un *EmergencyReport* solitamente determina la creazione di un *Incident* da parte di un *Dispatcher*. Un *EmergencyReport* è composto da un livello di emergenza, un tipo (fuoco, stradale, ..), un luogo e una descrizione
FieldOfficer	Un agente di polizia o dei vigili del fuoco in servizio. Un *FieldOfficer* può essere allocato al più ad un *Incident* alla volta. I *FieldOfficer* sono identificati da badge
Incident	Situazione che richiede l'attenzione di un *FieldOfficer*. Un *Incident* può essere riportato nel sistema da un *FieldOfficer* o da qualcuno anche esterno al sistema. Un *Incident* è composto da una descrizione, una risposta, uno status (aperto, chiuso, documentato), una locazione, e un numero di *FieldOfficer*

Identificazione degli Oggetti Boundary dallo use case ReportEmergency

ReportEmergencyButton	Bottone usato dal *FieldOfficer* per iniziare lo use case *ReportEmergency*
EmergencyReportForm	Form usato per l'input del *ReportEmergency*. Questa form è presentata sul *FieldOfficerStation* quando viene selezionata "Report Emergency". *EmergencyReportForm* contiene campi per specificare tutti gli attributi di un report di emergenza e un bottone (o altro controllo) per sottomettere la form completata.
FieldOfficerStation	Computer usato dal *FieldOfficer*
IncidentForm	Form usata per la creazione di *Incident*. Questa form è presentata sul *DispatcherStation* quando è ricevuto *l'EmergencyReport*. Il Dispatcher usa anche questa form per allocare le risorse e notificare il report del *FieldOfficer*
AcknowledgmentNotice	Avviso usato per mostrare l'acknowlegment del *Dispatcher* al *FieldOfficer*

Identificare gli Oggetti Control

- Gli oggetti Control sono responsabili del coordinamento degli oggetti Boundary e Entity
  - Si preoccupano di collezionare informazioni dagli oggetti Boundary e inviarle agli oggetti Entity
- Di solito non hanno una controparte nel mondo reale: modellano la logica di funzionamento di un caso d'uso.
- Esiste una stretta relazione tra oggetti Control e use case:
  - Un oggetto Control è creato all'inizio dello use case e cessa di esistere alla fine
- Linee Guida
  - Identificare almeno un oggetto Control per ogni use case
  - La vita di un oggetto Control dovrebbe corrispondere alla durata di uno use case o di una sessione utente. Se è difficile identificare l'inizio e la fine dell'attivazione di un oggetto Control, il corrispondente use case probabilmente non ha delle entry e exit condition ben definite

Identificare gli Oggetti Control dallo use case ReportEmergency

- Il flusso di controllo dello use case ReportEmergency viene modellato con due oggetti Control
  - ReportEmergencyControl per FieldOfficer
  - ManageEmergencyControl per Dispatcher
- Tale decisione deriva dalla consapevolezza che FieldOfficerStation e DispatcherStation sono due sottosistemi che comunicano su un link asincrono
  - Questa decisione potrebbe essere rimandata all'attività di design, comunque renderla visibile in fase di analisi consente di focalizzare l'attenzione su comportamenti eccezionali, come la perdita di comunicazione tra due stazioni
- Nel modellare lo use case ReportEmergency sono state modellate le stesse funzionalità usando oggetti Boundary, Entity e Control
  - Si è passati da una prospettica "flusso di eventi" ad una "strutturale"
  - È aumentato il livello di dettaglio della descrizione
  - Sono stati selezionati termini standard per riferirci alle entità principali del dominio di applicazione e del sistema

Oggetti Control dallo use case ReportEmergency

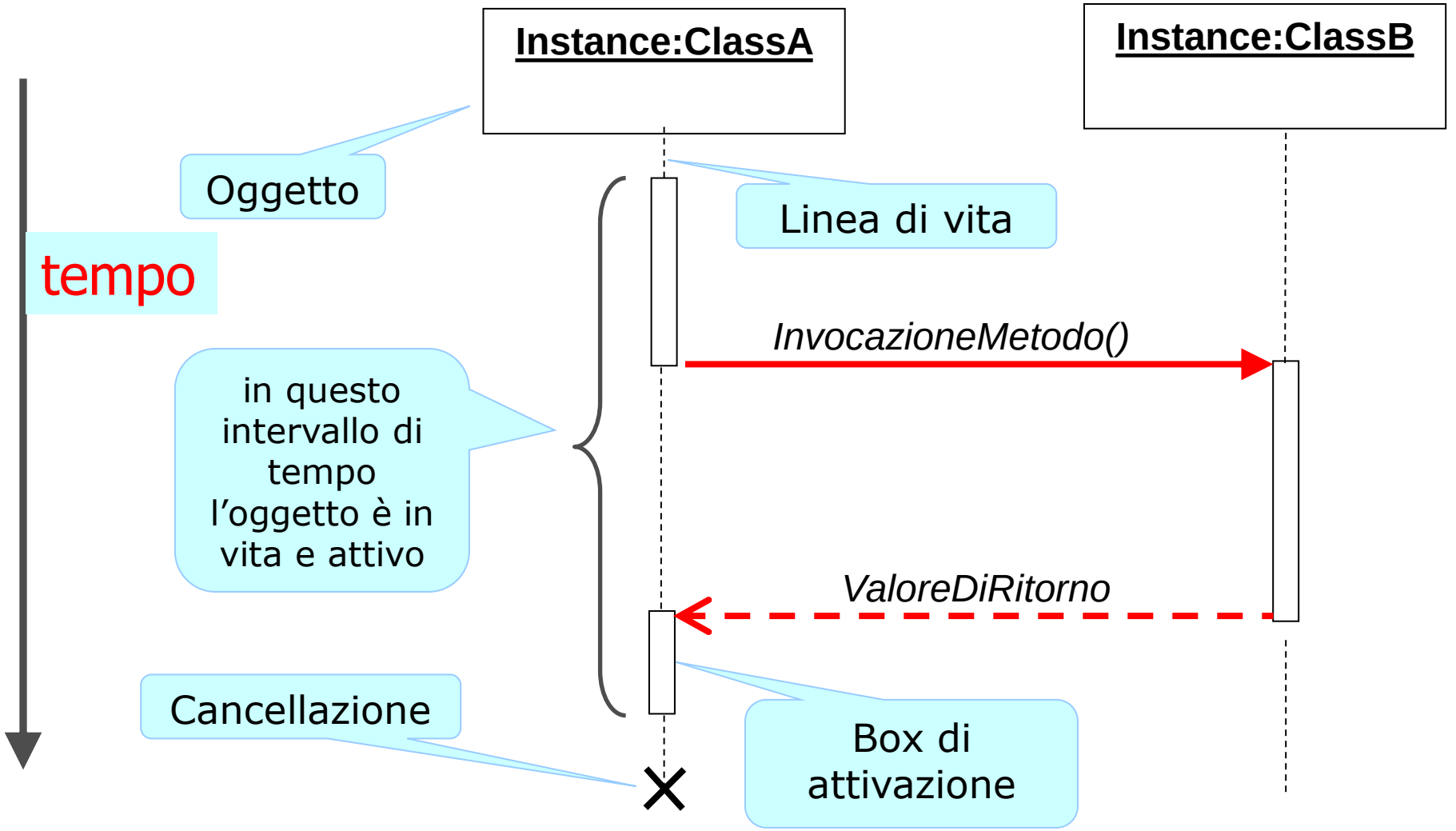
ReportEmergencyControl	Gestisce la funzione *ReportEmergency* sulla *FieldOfficerStation*. Questo oggetto è creato quando il *FieldOfficer* seleziona il bottone “Report Emergency”. Crea un *EmergencyReportForm* e lo presenta al *FieldOfficer*. Dopo la sottomissione della form, questo oggetto colleziona l’informazione dalla form, crea un *EmergencyReport*, e lo inoltra al *Dispatcher*. L’oggetto Control quindi aspetta una notifica dal *DispatcherStation*. Quando riceve la notifica, l’oggetto *ReportEmergencyControl* crea un *AcknowlegmentNotice* e lo mostra al *FieldOfficer*
ManageEmergencyControl	Gestisce la funzione *ReportEmergency* sulla *DispatcherStation*. Questo oggetto è creato quando viene ricevuto un *EmergencyReport*. Quindi crea un *IncidentForm* e lo presenta al *Dispatcher*. Quando il *Dispatcher* ha creato un *Incident*, allocato *Resources,* e sottomesso una notifica, *ManageEmergencyControl* inoltra la notifica a *FieldOfficerStation*

I Sequence Diagrams

Sequence Diagram

- Mostra la sequenza temporale dei messaggi che gli oggetti si scambiano per portare a termine una funzionalità.
- E’ un diagramma di interazione: evidenzia come una funzionalità è realizzata tramite la collaborazione di un insieme di oggetti
- E’ uno dei principali input per l’implementazione dello scenario
  - E’ utilizzato in analisi e poi ad un maggior livello di dettaglio in design

Sequence Diagram



Sequence Diagram

- La ricezione di un messaggio determina l'attivazione di un metodo
  - L'attivazione è rappresentata da un rettangolo sulla linea della vita, da cui altri messaggi possono prendere origine
  - La lunghezza del rettangolo rappresenta il tempo durante il quale l'operazione è attiva
- La vita degli oggetti
  - Il tempo procede verticalmente dal top al bottom
  - Al top del diagramma si trovano gli oggetti che esistono prima del 1° messaggio inviato
  - Oggetti creati durante l'interazione sono illustrati con il messaggio <<create>>
  - Oggetti distrutti durante l'interazione sono evidenziati con una croce
  - La linea tratteggiata indica il tempo in cui l'oggetto può ricevere messaggi

Messaggi nei Sequence

- In generale un messaggio rappresenta il trasferimento del controllo da un oggetto ad un altro

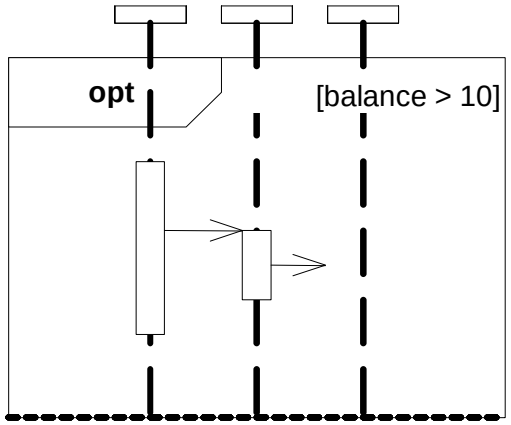
  - Se l'oggetto che invia il messaggio rimane in attesa che l'oggetto ricevente ritorni, si ha un messaggio **sincrono**

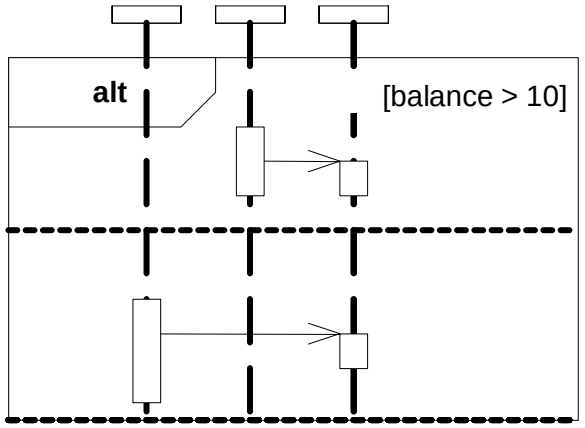
  - Se l'oggetto che invia il messaggio prosegue la propria elaborazione in parallelo all'oggetto chiamato, siamo in presenza di un messaggio **asincrono**.

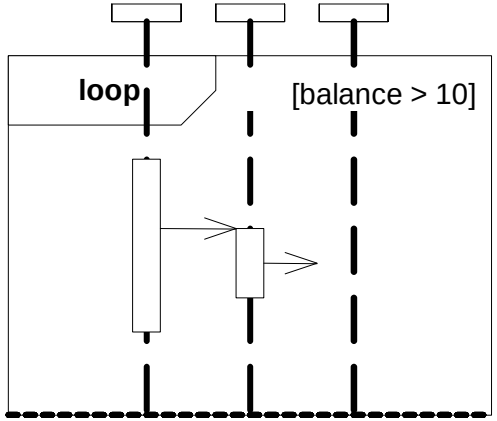
  - Il valore restituito all'oggetto chiamante si indica con un **messaggio di ritorno**

Opzioni e Cicli

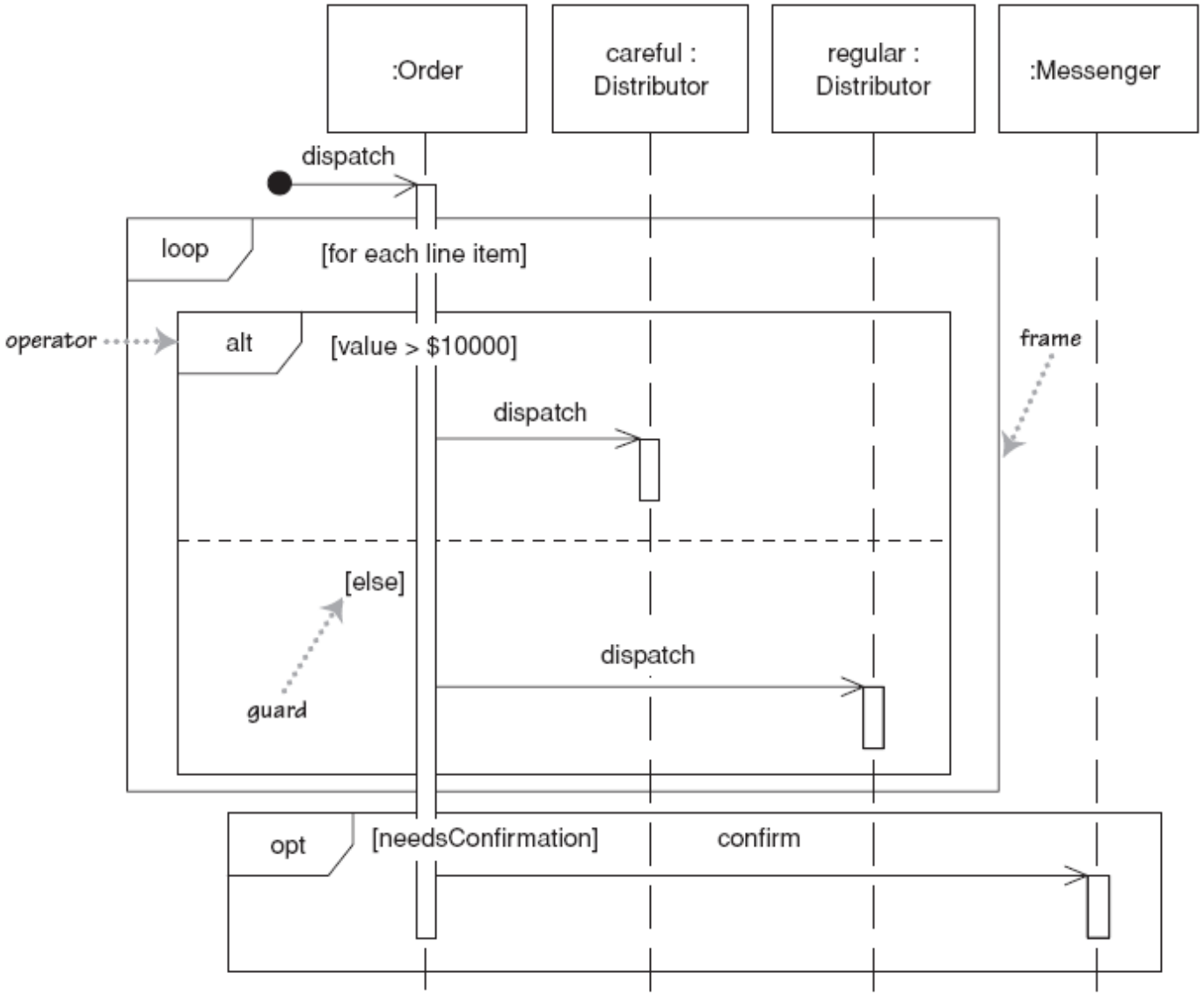
- Si indicano con una cornice (frame) intorno ad una parte del Sequence, per indicare che quella sezione è opzionale o viene ripetuta.
- Rappresentazioni:
  - if -> OPT [condition]
  - if/else -> ALT [condition], separati da linea orizzontale tratteggiata
  - cicli -> LOOP [condition o items su cui ciclare]





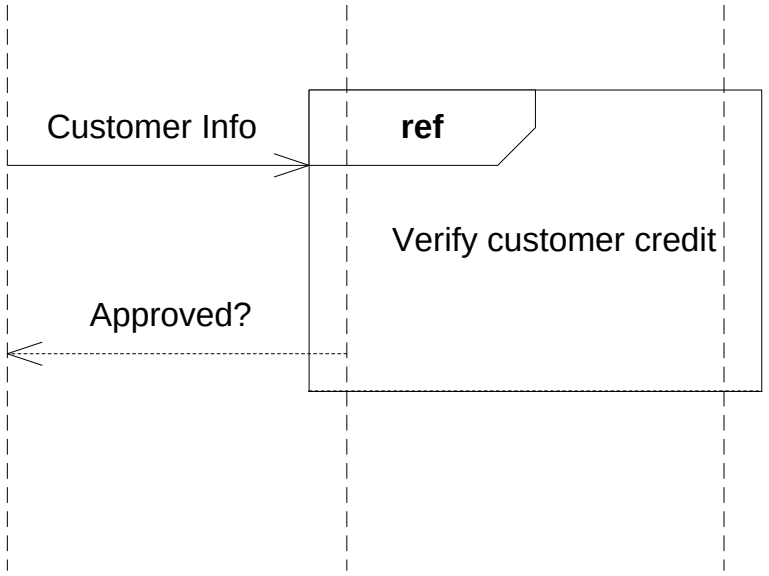


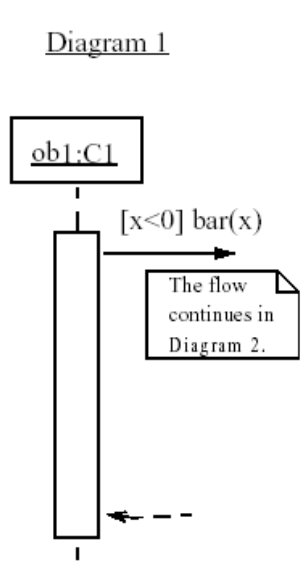
Esempio

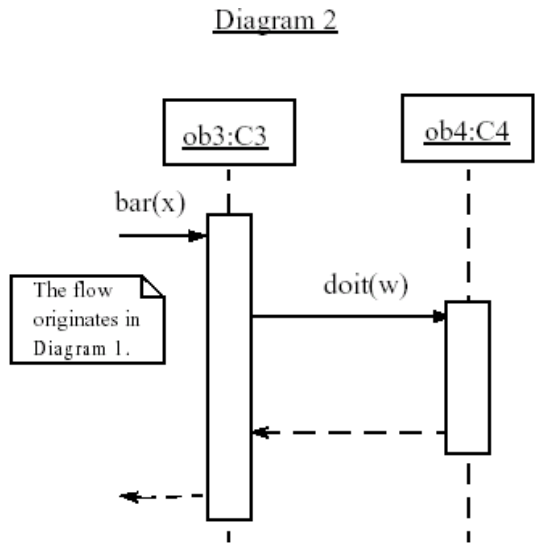

:Order
careful : Distributor
regular : Distributor
:Messenger
dispatch
loop
[for each line item]
operator
alt
[value > $10000]
frame
dispatch
[else]
guard
dispatch
opt
[needsConfirmation]
confirm

Linking sequence diagrams

- Se un Sequence è troppo complesso o fa riferimento ad un altro Sequence, si può utilizzare il REF, con:
  - Un rettangolo con label REF, col nome dell'altro diagramma
  - Una freccia che punta a tale rettangolo
  - Una eventuale condizione per specificare quando si fa il riferimento







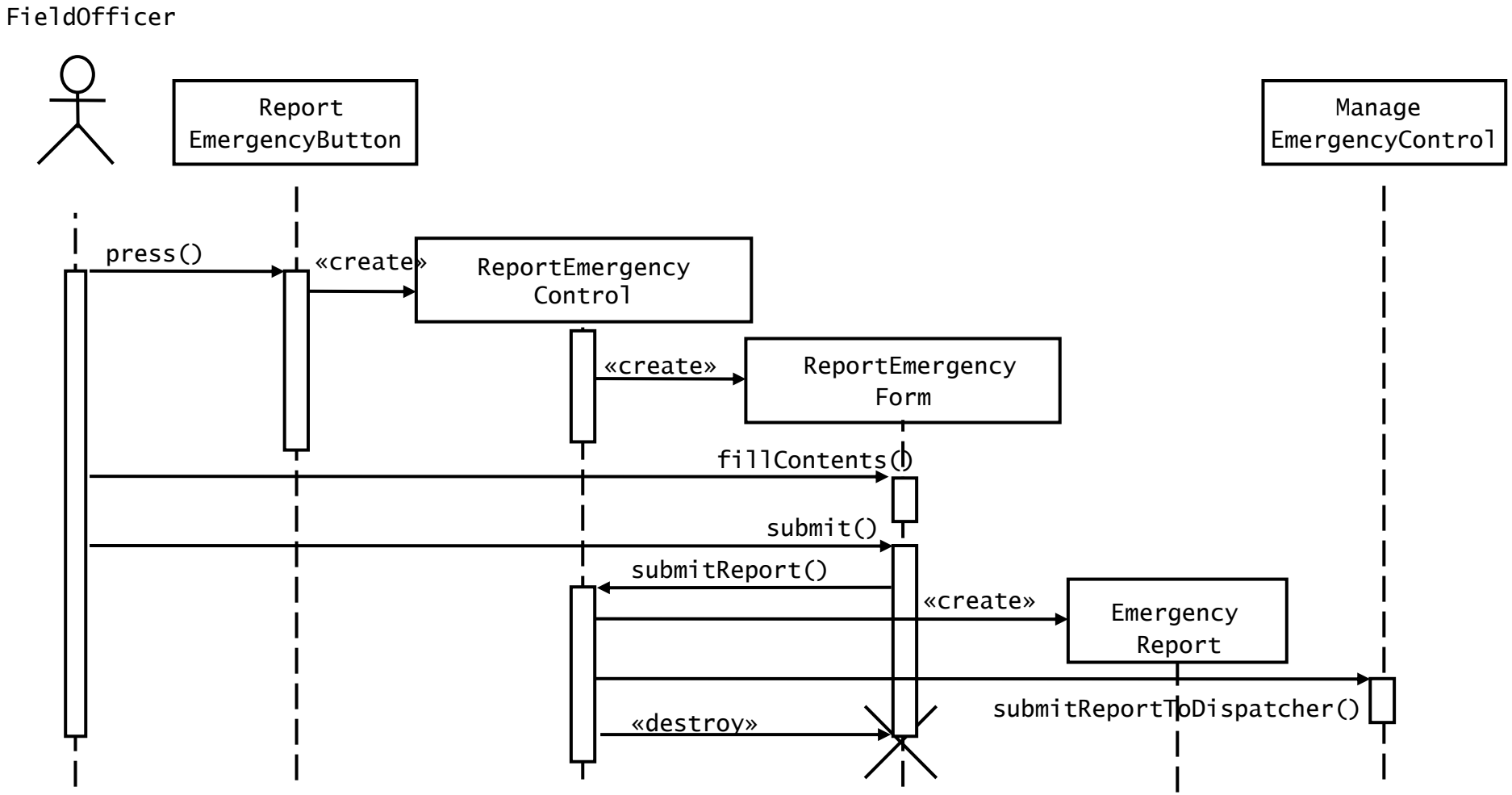
Mappare Use case in Oggetti con Sequence Diagram

- Un Sequence Diagram lega use case con oggetti.
  - mostra come il comportamento di uno use case (o scenario) è distribuito tra i suoi oggetti partecipanti
    - Vengono assegnate responsabilità a ogni oggetto in termini di un insieme di operazioni
  - Illustra la sequenza di interazioni tra gli oggetti necessaria per realizzare uno use case
    - non ci occupiamo di questioni di implementazioni, come l'efficienza!
- Non è adatto alla comunicazione con il cliente
  - Solo per i clienti esperti è più intuitivo e preciso degli use case
- Fornisce una prospettiva diversa che consente di individuare classi mancanti e aree non chiare nelle specifiche

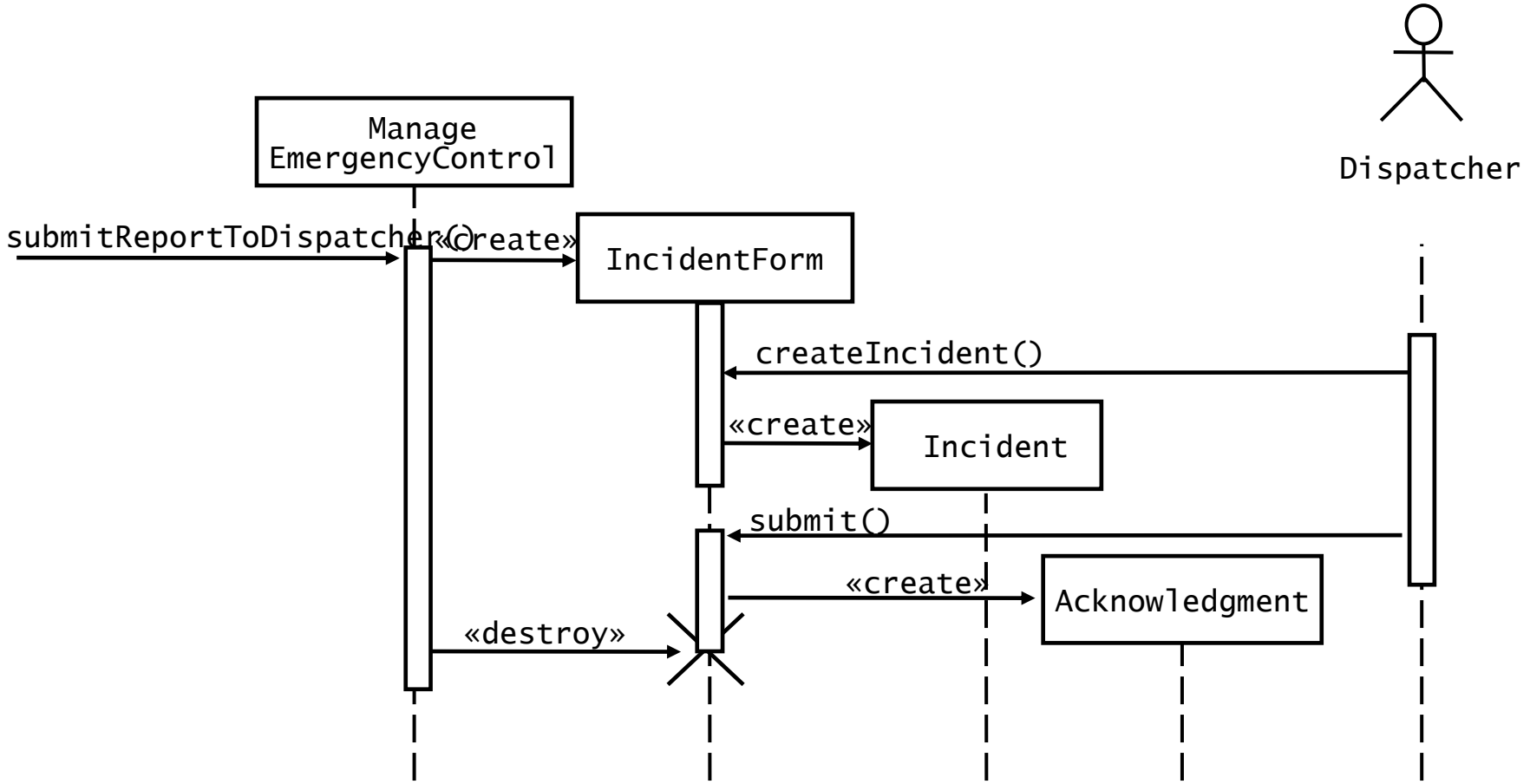
Sequence Diagram in fase di analisi

- Convenzioni per utilizzare i Sequence Diagrams con l'euristica Three-Object-Type in fase di analisi:
  - La colonna più a sinistra rappresenta l'attore che inizia lo use case
  - La seconda colonna -> oggetto Boundary con cui l'attore interagisce per iniziare lo use case
  - La terza colonna -> oggetto Control che gestisce il resto dello use case
  - Le altre colonne possono rappresentare qualunque oggetto che interviene nel caso d'uso.
    - Gli oggetti Control creano altri oggetti Boundary/Entity e possono interagire con altri oggetti
  - Oggetti Control accedono ad altri oggetti Entity e Boundary
  - Gli oggetti Entity non accedono mai agli oggetti Control e Boundary: ciò rende più facile condividere oggetti Entity tra più use case

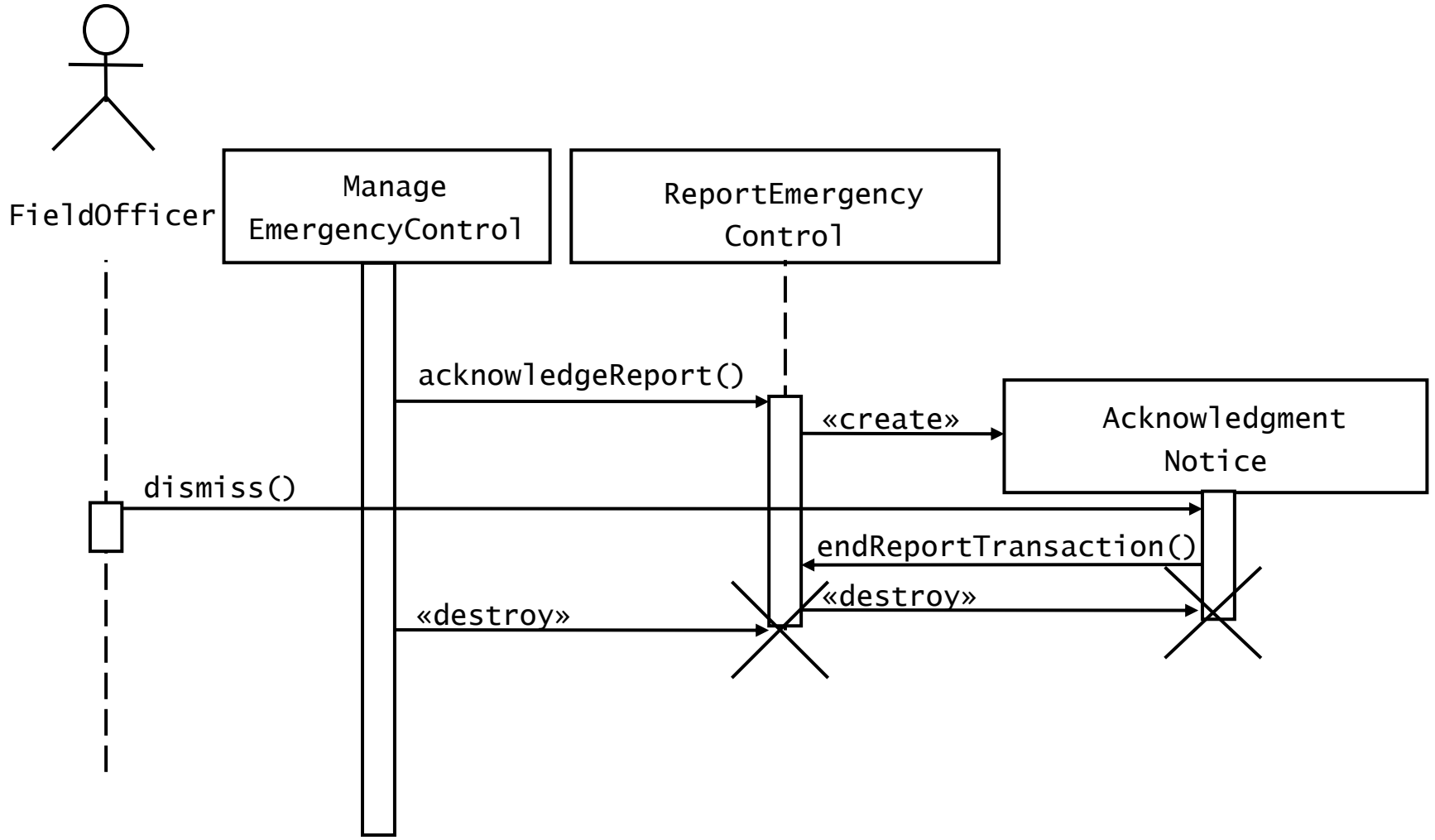
Sequence diagram for the ReportEmergency use case.



Sequence diagram for the ReportEmergency use case (continued).



Sequence diagram for the ReportEmergency use case (continued).



Identificazione di nuovo oggetto

- Lo use case ReportEmergency è incompleto: menziona l'esistenza di una notifica ma non descrive l'informazione ad essa associata
- C'è necessità di chiarire con il cliente→ l'oggetto Acknowledgment è aggiunto al modello di analisi e lo use case è raffinato
- L'oggetto Acknowlegment è creato prima dell'oggetto Boundary AcknowldegmentNotice

Identificazione di nuovo oggetto

Acknowledgment	Risposta di un Dispatcher a un EmergencyReport di un FieldOfficer. Inviando un Acknowledgment, il Dispatcher comunica al FieldOfficer che ha ricevuto l'EmergencyReport, crea un Incident, e assegna risorse. L'Acknowledgment contiene le risorse assegnate e il tempo stimato del loro arrivo

Analisi e Sequence Diagram

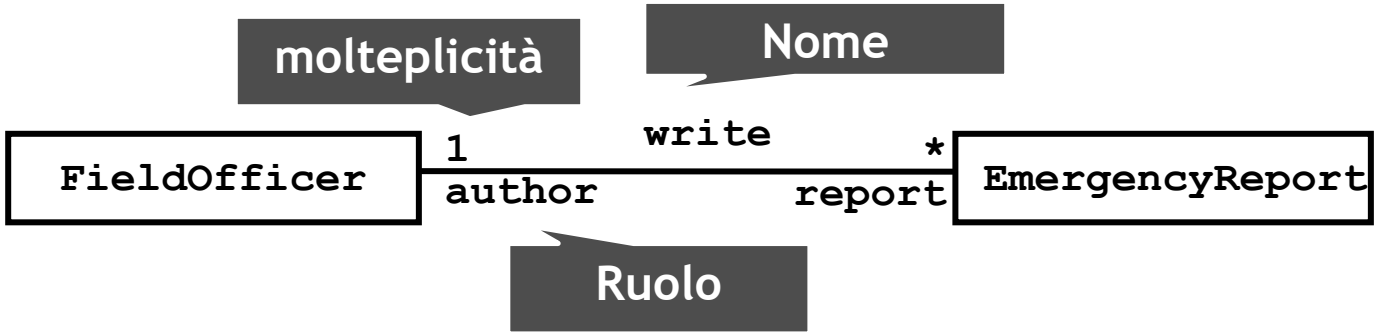
- Durante l’analisi i Sequence Diagram sono usati per individuare
  - nuovi oggetti
  - comportamenti mancanti

- Disegnare Sequence Diagram è un’attività laboriosa, quindi:
  - Occorre dare priorità a quelle funzionalità problematiche o non ben specificate
  - Per le parti ben definite può essere utile solo per evitare di posticipare alcune decisioni chiavi

Relazioni tra oggetti

Identificare le Associazioni

- I sequence diagram permettono di descrivere le relazioni tra gli oggetti
- I class diagram permettono di descrivere le dipendenze tra gli oggetti

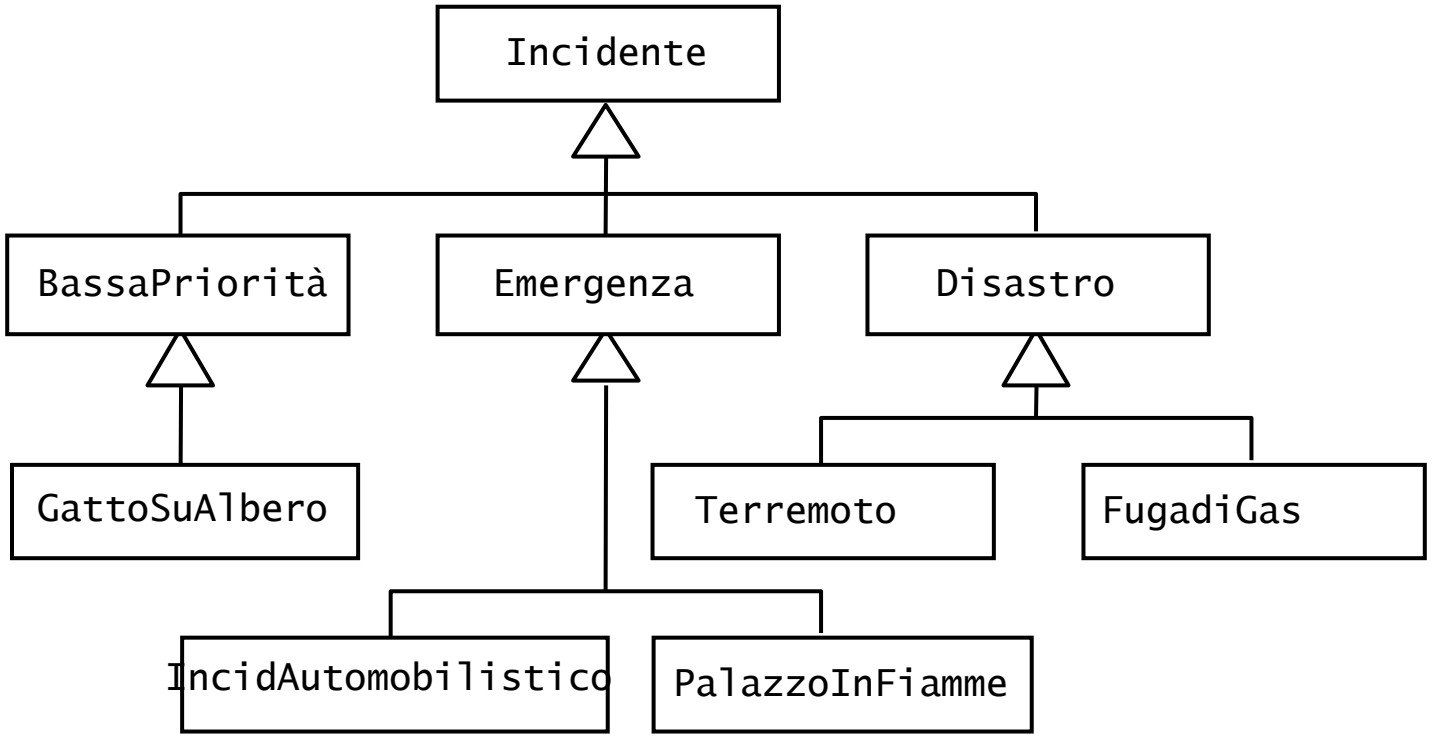


Le associazioni tra entity object sono le più importanti poichè rivelano altre informazioni sul dominio di applicazione

Generalizzazione e specializzazione

- Ereditarietà consente di organizzare concetti in gerarchie:
  - Al top della gerarchia concetti più generali
  - Al bottom concetti più specializzati
- Generalizzazione: attività di modellazione che identifica concetti astratti da quelli di più basso livello
  - Es. Stiamo facendo reverse-engineering di un sistema di gestione delle emergenze e analizzando le videate per la gestione di incidenti autostradali e incendi. Osservando concetti comuni, creiamo un concetto astratto Emergenze
- Specializzazione: attività che identifica concetti più specifici da quelli di più ad alto livello
  - Es. Stiamo costruendo un sistema di gestione delle emergenze e stiamo discutendo le funzionalità con il cliente: il cliente introduce prima il concetto di incidente, quindi descrive tre tipi di incidenti: disastri, emergenze, incidenti a bassa priorità
- Come risultato sia della specializzazione che della generalizzazione abbiamo la specifica di ereditarietà tra concetti

Un esempio di una generalizzazione gerarchica.

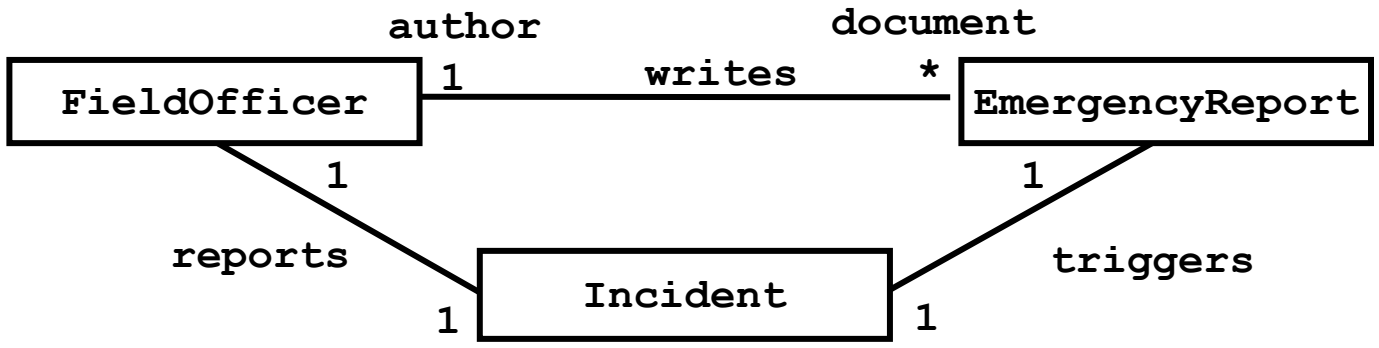


Euristiche per identificare le associazioni

- Esaminare i verbi nelle frasi
- Definire i nomi dei ruoli e delle associazioni
- Eliminare qualsiasi associazione che può essere derivata da altre associazioni
- Non preoccuparsi di specificare le molteplicità finchè l'insieme di associazioni non è stabile
- Troppe associazioni rendono un modello illeggibile

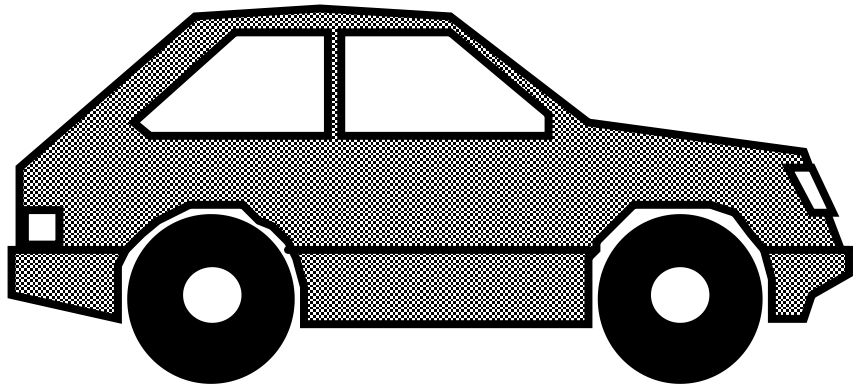
Eliminare le associazioni ridondanti

- La ricevuta di un EmergencyReport causa la creazione di un Incident da un Dispatcher.
- Dato che EmergencyReport ha una associazione con il FieldOfficer che la scrive, non è necessario mantenere un'associazione tra FieldOfficer e Incident.

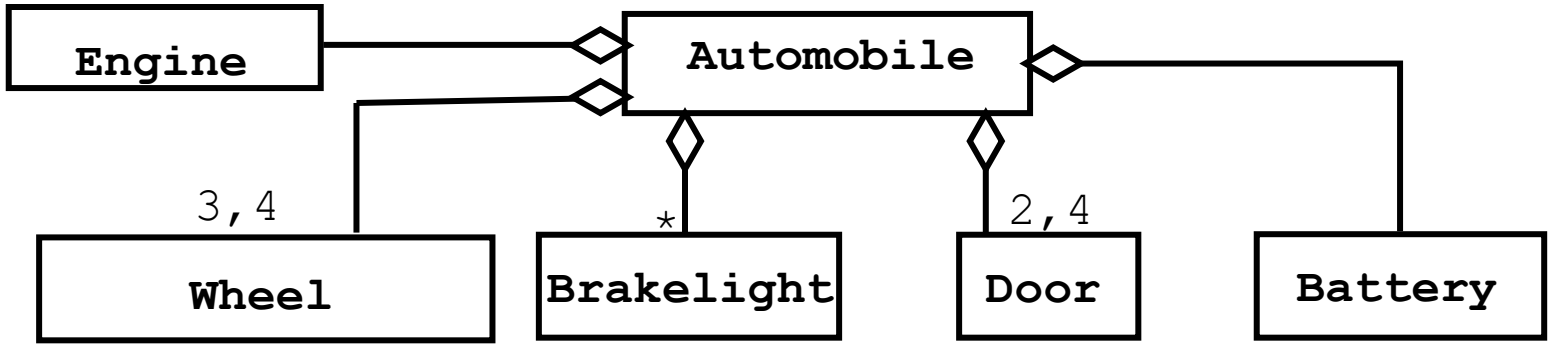


Aggregazione e Composizione

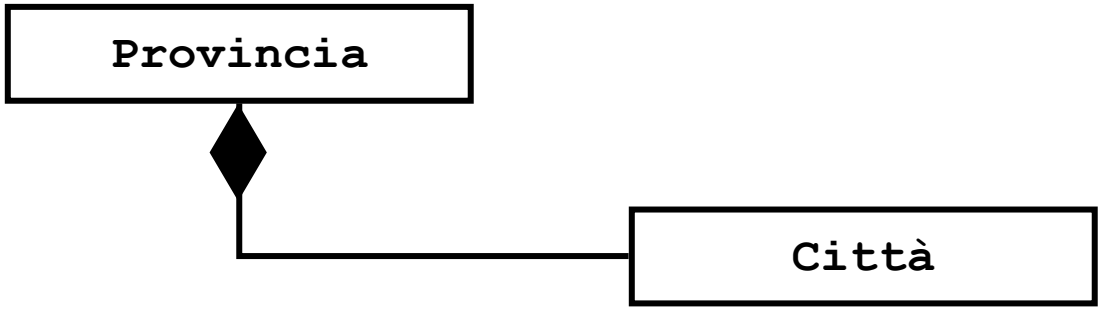
- Sono tipi speciali di associazioni che denotano relazioni whole-part
- Notazione UML : Come una associazione ma con un piccolo rombo indicante la parte assemblata della relazione.



Aggregazione e composizione

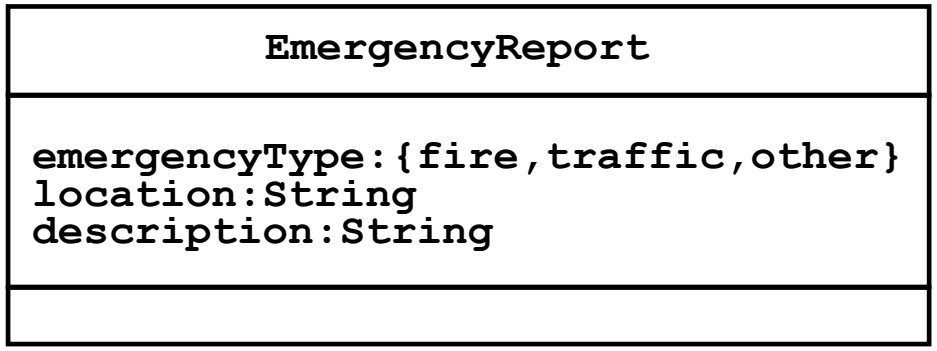


- L'automobile e le singole parti possono esistere indipendentemente
- La composizione indica che l'esistenza delle singole parti dipendono dall'intero



Identificare gli attributi

- Le proprietà rappresentate da oggetti non sono attributi (ad es. Author per EmergencyReport)
- Bisogna identificare tutte le associazioni prima di iniziare l'identificazione degli attributi



- Le proprietà rappresentate da oggetti non sono attributi (ad es. Author per EmergencyReport)
- Bisogna identificare tutte le associazioni prima di iniziare l'identificazione degli attributi

Identificare gli attributi

- Gli attributi hanno:
  - Un nome
  - Una breve descrizione
  - Un tipo che descrive i valori che può assumere
- Spesso altri attributi sono scoperti nella fase di sviluppo quando il sistema è valutato dall'utente

Euristiche per Identificare gli attributi

- Esaminare le frasi possessive
- Rappresentare lo stato memorizzato come un attributo dell'oggetto
- Descrivere ogni attributo
- Non rappresentate un attributo come un oggetto; usare invece una associazione
- Non sprecare tempo a descrivere i dettagli prima che la struttura ad oggetti non si sia stabilizzata

Statechart Diagrams

- Sono grafi i cui nodi sono stati e i cui archi sono transizioni etichettate da nomi di eventi.
- Occorre distinguere tra due tipi di operazioni:
  - Attività: Operazioni che impiegano tempo per essere completate
    - sono associate con gli stati
  - Azioni: Operazioni istantanee
    - Associate con gli eventi
    - associate con gli stati (riducono la complessità di disegno): Entry, Exit, Internal Action
- Uno statechart diagram rela eventi e stati per una classe
  - Un modello ad oggetto con un insieme di oggetti ha un insieme di state diagram

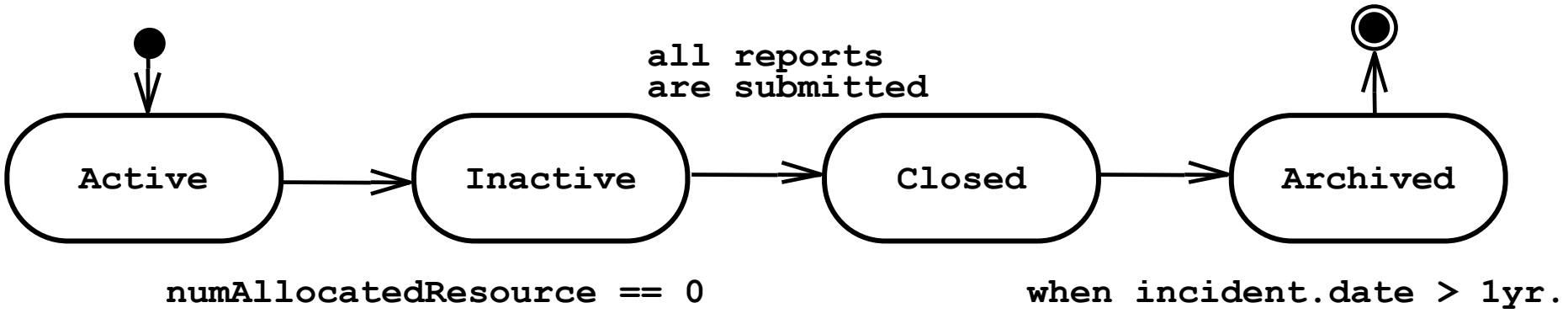
Statechart Diagrams

- Permettono allo sviluppatore di costruire una descrizione più formale dell'oggetto
- Di conseguenza permettono di identificare casi d'uso mancanti
- Focalizzando l'attenzione sui singoli stati gli sviluppatori possono identificare nuovi comportamenti
- Non è necessario creare uno statechart per ogni classe nel sistema (solo quelli con una lunga vita ed un comportamento dipendente dallo stato), molto spesso sono control object

UML Statechart Diagram Notation

- Sequence diagram rappresentano il comportamento del sistema dal punto di vista di un singolo caso d'uso
- Statechart diagram rappresentano il comportamento del sistema dal punto di vista di un singolo oggetto

statechart for Incident.



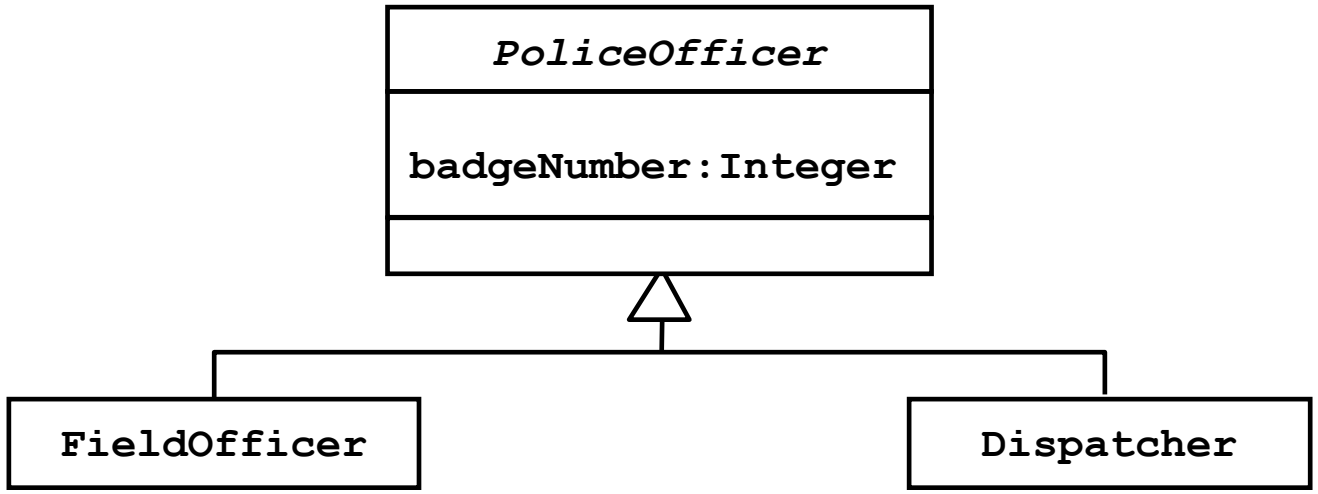
Statechart Diagram vs Sequence Diagram

- Statechart diagrams aiutano ad identificare:
  - Cambiamenti degli oggetti nel tempo

- Sequence diagrams aiutano ad identificare:
  - Le relazioni temporali tra gli oggetti nel tempo
  - Sequenze di operazioni come risposta ad uno o più eventi

Ereditarietà

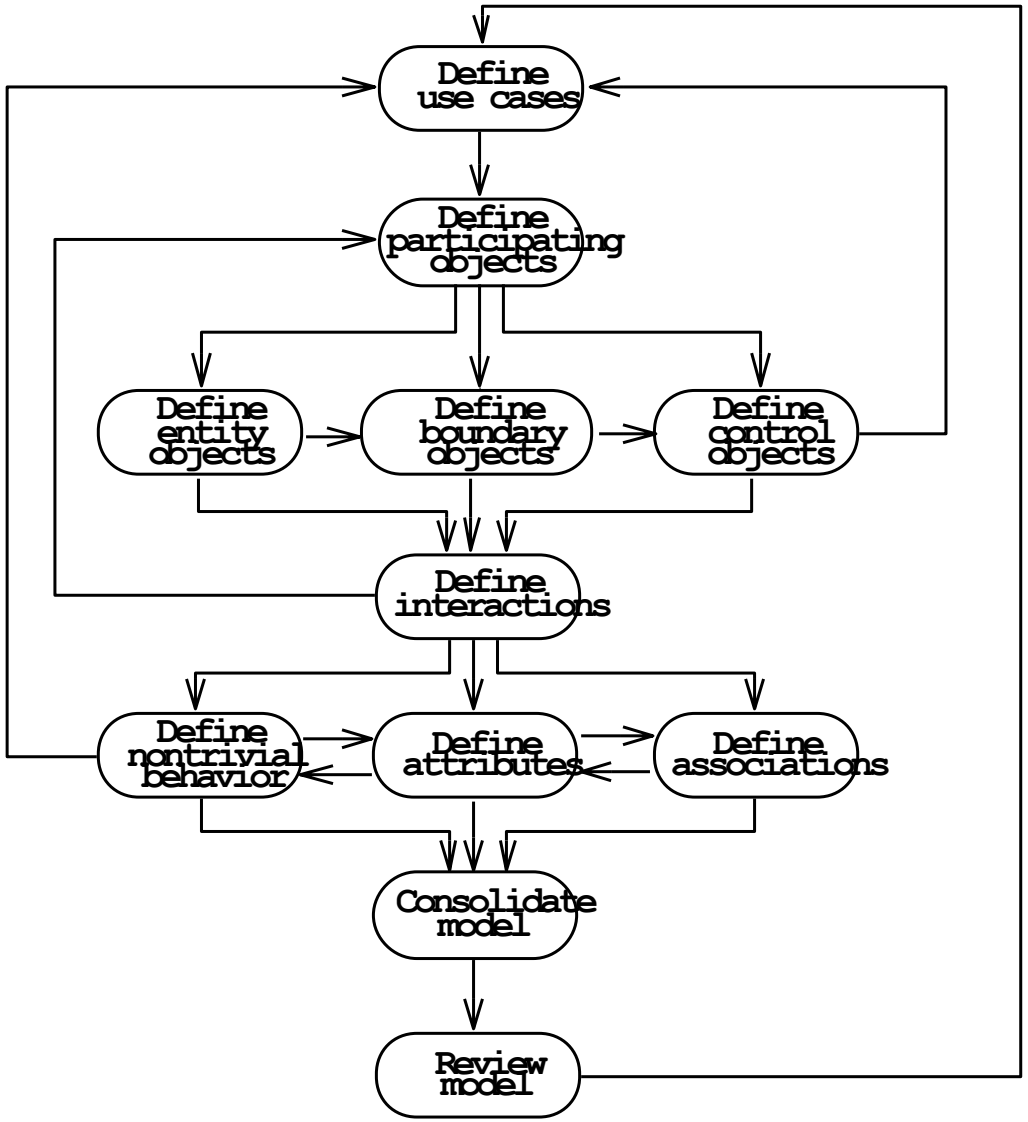
- La generalizzazione è utilizzata per eliminare le ridondanze dal modello di analisi



Revisione del Modello di Analisi

- Il modello di Analisi è costruito in maniera incrementale ed iterativa
- Una volta che il modello diventa stabile, il modello di analisi viene revisionato prima dagli sviluppatori (revisione interna), poi congiuntamente dagli sviluppatori e dal cliente
- Obiettivo: una specifica dei requisiti corretta, completa, consistente e non ambigua

Attività di Specifica dei Requisiti



Documento dei Requisiti Software

- I risultati della raccolta dei requisiti e dell'analisi sono documentati nel DRS
- Il DRS descrive completamente il sistema in termini di requisiti funzionali e non funzionali e serve come base del contratto tra cliente e sviluppatori.
- I partecipanti coinvolti sono: cliente, utenti, project manager, analisti del sistema, progettisti
- La prima parte del documento, che include Use Case e requisiti non funzionali, è scritto durante la raccolta dei requisiti
- La formalizzazione della specifica in termini di modelli degli oggetti è scritto durante l'analisi

Documento dei Requisiti Software

- 1. Introduction
- 1.1. Purpose of the system
- 1.2. Scope of the system
- 1.3. Objectives and success criteria of the project
- 1.4. Definition, acronyms, and abbreviations
- 1.5. References
- 1.6. Overview
- 2. Current system
- 3. Proposed system
- 3.1. Overview
- 3.2. Functional requirements
- 3.3. Nonfunctional requirements
- 3.4. System models
- 3.4.1. Scenarios
- 3.4.2. Use case model
- 3.4.3. Object model (during analysis)
- 3.4.4. Dynamic model (during analysis)
- 3.4.5. User interface – navigational path and screen mock-up
- 4. Glossary

Assegnare le responsabilità

- L’analisi richiede la partecipazione di molti individui
- Ci sono tre tipi di ruoli: generatori di informazioni, integratori e revisori.

  - End user: generano informazioni sul sistema
  - Client: integra le informazioni del dominio di applicazione e risolve le inconsistenze
  - Analyst: modella il sistema e genera informazioni sul sistema da costruire
  - Architect: integra i casi d’uso e i modelli ad oggetti dal punto di vista del sistema
  - Reviewer: valida il RAD

Accettazione del cliente

- Il cliente e gli sviluppatori convergono su una singola idea e concordano sulle funzioni e le caratteristiche che dovrà avere il sistema
- Si accordano anche su:
  - Una lista di priorità
  - Un processo di revisione
  - Una lista di criteri da utilizzare per accettare o rifiutare il sistema
  - Una pianificazione e un budget