# 15. Analisi e modellazione dinamica

Paola Barra
2022/2023

# Come troviamo le classi?

Abbiamo già esaminato diverse sorgenti per l'identificazione delle classi:

- Analisi del Dominio Applicativo: troviamo le classi parlando con il cliente e identificando le astrazioni osservando l'utente
- Conoscenza generale e intuizione
- Analisi testuale dei flussi di eventi nei casi d'uso (Abbott)

## Identifichiamo le classi dai modelli dinamici

Due buone euristiche:

- Azioni e attività nei diagrammi degli stati sono candidate per le operazioni nelle classi
- Le linee di attività nei diagrammi delle sequenze sono candidate per gli oggetti

# Modellazione dinamica e UML

Due tipi di diagrammi UML per la modellazione dinamica:

- Diagrammi delle interazioni: descrivono il comportamento dinamico tra gli oggetti

- Diagrammi degli stati: descrivono il comportamento dinamico di un singolo oggetto

# Modellazione dinamica

Il modello dinamico descrive le componenti del sistema che hanno comportamenti dinamici interessanti.

Il modello dinamico è descritto con

- Diagrammi degli stati: un diagramma degli stati **per ogni classe** con comportamento interessante
- Le classi che non hanno un comportamento interessante non sono modellate con i diagrammi di stato
- Diagrammi delle sequenze: per l'interazione tra classi

Scopo: determinare e fornire le operazioni del modello ad oggetti

# Come determiniamo le operazioni?

Cerchiamo gli oggetti che stanno interagendo ed estraiamo il loro “protocollo”

Cerchiamo gli oggetti che individualmente hanno un comportamento interessante

Un buon punto di partenza: flusso di eventi nella descrizione dei casi d’uso

Dal flusso di eventi procediamo con il diagramma delle sequenze per trovare gli oggetti partecipanti

# Cosa è un evento?

Qualcosa che si verifica in un certo istante temporale

Un evento comporta l'invio di informazioni da un oggetto ad un altro

Gli eventi possono avere delle associazioni tra loro:

- Relazione causale: Un evento si verifica sempre prima o dopo un altro evento
- Relazione non causale: Gli eventi occorrono concorrentemente

Gli eventi possono anche essere raggruppati in classi di eventi con una struttura gerarchica => Tassonomia degli eventi.

# Diagramma delle sequenze

Euristiche per trovare gli oggetti partecipanti:

- Un evento ha sempre un mittente ed un destinatario
- Trovare **mittenti** e **destinatari** per ciascun evento => questi sono gli oggetti che partecipano in un caso d'uso

# Mapping casi d’uso - oggetti, con il diagramma delle sequenze

Un diagramma di sequenze lega i casi d’uso con gli oggetti.

Mostra come il comportamento di un caso d’uso (o scenario) sia distribuito tra gli oggetti partecipanti.

Non sono molto adeguati come mezzo di comunicazione con i clienti

Richiedono un certo background sulla notazione

Possono però essere, per alcuni clienti, intuitivi e più precisi dei casi d’uso

In ogni caso, rappresentano una ulteriore prospettiva che consente agli sviluppatori di individuare oggetti mancanti o punti oscuri nella specifica dei requisiti

# Diagramma di sequenze per ReportEmergercy

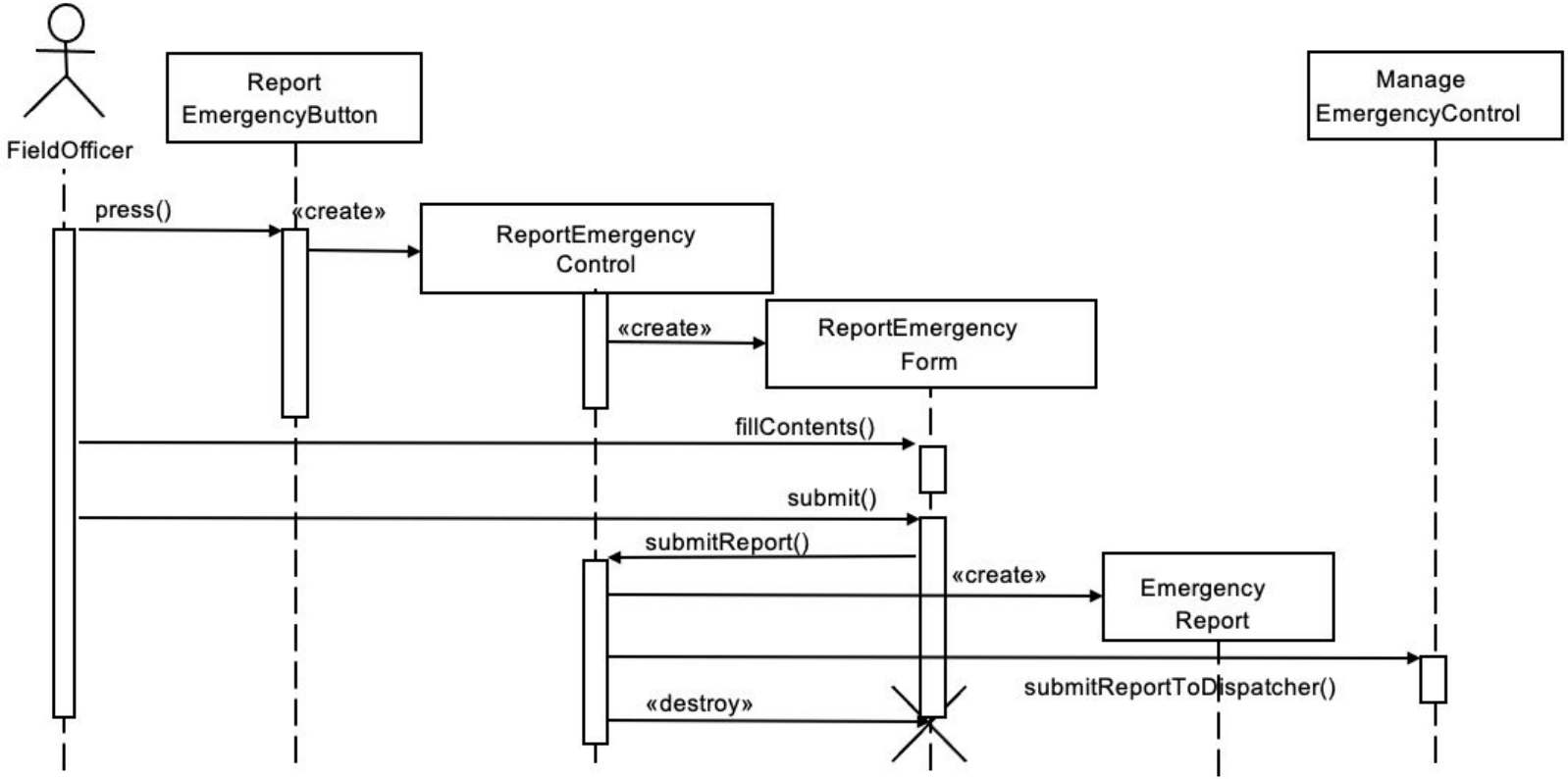
Vediamo i diagrammi di sequenze associati con il caso d'uso *ReportEmergency*

- Le colonne rappresentano gli oggetti che partecipano al caso d'uso
- La colonna più a sinistra rappresenta l'attore che inizia il caso d'uso
- Le frecce orizzontali attraverso le colonne rappresentano messaggi o stimoli inviati da un oggetto all'altro
- Il tempo trascorre verticalmente dall'alto in basso

# Oggetti control del caso d'uso ReportEmergercy control

| | |
|---|---|
| **ReportEmergencyControl** | Gestisce la funzione di reporting del ReportEmergency sulla FieldOfficerStation. Questo oggetto è creato quando il FieldOfficer seleziona il pulsante "Report Emergency". Esso dopo crea una EmergencyReportForm e la presenta al FieldOfficer. Dopo aver sottomesso la form, tale oggetto raccoglie le informazioni dalla form, crea un EmergencyReport, e lo invia al Dispatcher. L'oggetto control dopo aspetta un'accettazione proveniente dalla DispatcherStation. Quando è ricevuta l'accettazione, l'oggetto ReportEmergencyControl crea un AcknowledgeNotice e la visualizza al FieldOfficer. |
| **ManageEmergencyControl** | Gestisce la funzione di reporting del ReportEmergency sulla DispatcherStation. Questo oggetto è creato quando è ricevuto un EmergencyReport. Esso dopo crea una IncidentForm e la visualizza al Dispatcher. Una volta che il Dispatcher ha creato un Incident, allocato Resourse e sottomesso un'accettazione, ManageEmergencyControl invia l'accettazione alla FieldOfficerStation |

# Diagramma delle sequenze per ReportEmergency



# Diagramma delle sequenze per ReportEmergency

Un'operazione può essere pensata come un servizio che un oggetto fornisce ad altri oggetti

I diagrammi di sequenza illustrano anche il tempo di vita degli oggetti

- Gli oggetti che esistono già prima dell'occorrenza del primo stimolo nel diagramma sono disegnati in cima
- Gli oggetti creati durante l'interazione sono disegnati con il messaggio *create* che punta all'oggetto
- Le istanze che sono distrutte durante l'interazione sono disegnate con una croce che indica quando l'oggetto cessa di esistere

Tra il rettangolo che rappresenta l'oggetto e la croce (o il fondo del diagramma, se l'oggetto esiste dopo l'interazione), una linea tratteggiata rappresenta l'arco di tempo in cui l'oggetto può ricevere messaggi

L'oggetto non può ricevere messaggi al di sotto della croce

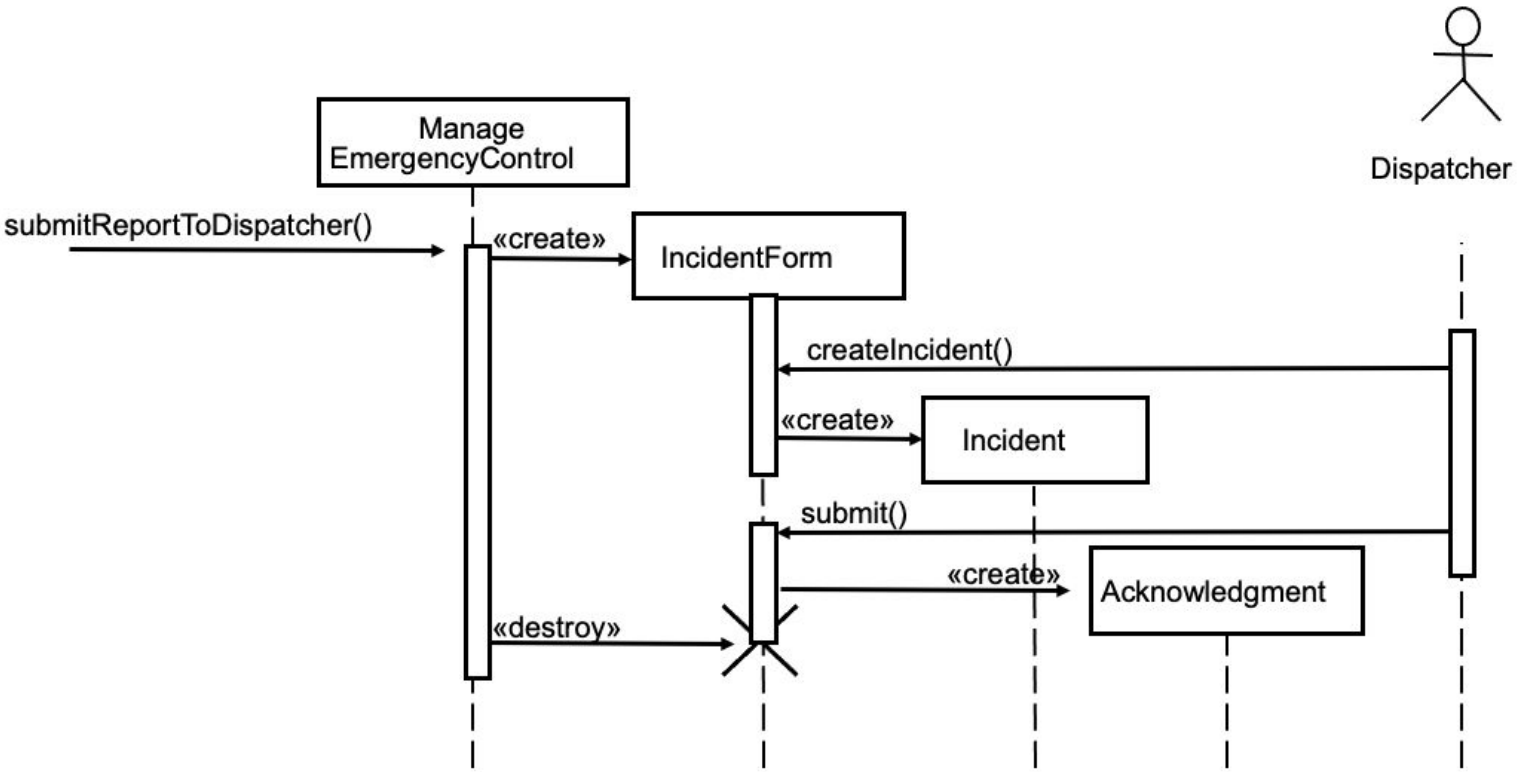
# Diagramma delle sequenze per ReportEmergency

In generale, la seconda colonna di un diagramma di sequenze rappresenta l'oggetto *boundary* con cui l'attore interagisce per iniziare il caso d'uso (es., *ReportEmergencyButton*)

La terza colonna è un oggetto control che gestisce il resto del caso d'uso (es., *ReportEmergencyControl*)

- Da quel momento in poi, l'oggetto *control* crea altri oggetti *boundary* e può interagire anche con altri oggetti *control* (*ManageEmergencyControl*)

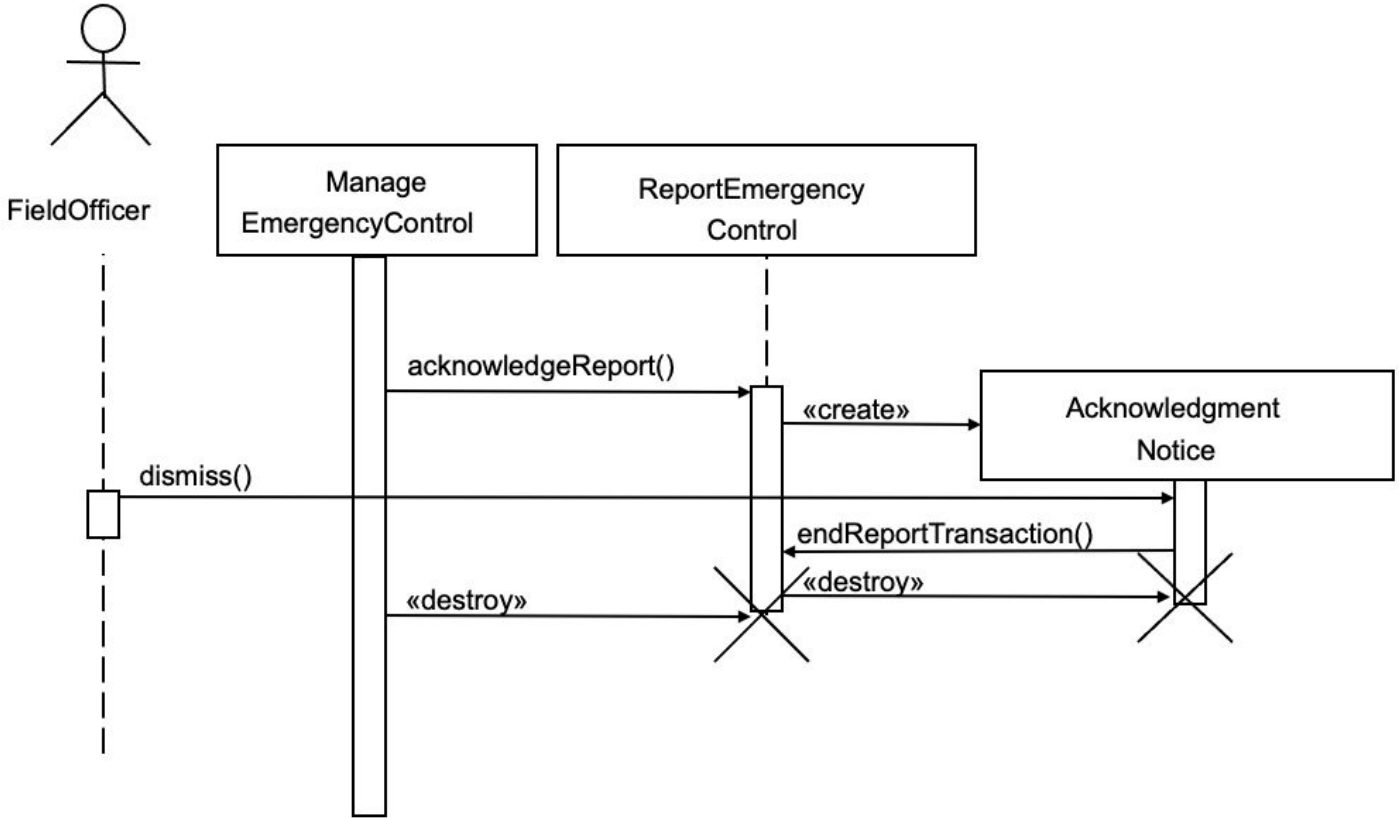
# Diagramma delle sequenze per ReportEmergency (II)



# Oggetto Acknowledgement per ReportEmergency

- Quando si descrive l'oggetto Acknowledgment ci si rende conto che il caso d'uso originale ReportEmergency è incompleto
- Esso evidenzia solo l'esistenza  di un Acknowledgment e non descrive le informazioni ad esso associate
- Gli sviluppatori hanno bisogno di un chiarimento dal cliente per definire quale informazione è necessaria in Acknowledgment
- Ottenuto il chiarimento viene aggiunto l'oggetto Acknowledgment al modello di analisi e il caso d'uso ReportEmergency è aggiornato di conseguenza

# Diagramma delle sequenze per ReportEmergency (III)



# Oggetto Acknowledgement per ReportEmergency

| | |
|---|---|
| **Acknowledgment** | Risposta di un *Dispatcher* ad un *EmergencyReport* di un *FieldOfficer*. Inviando un *Acknowledgment*, il *Dispatcher* comunica con il *FieldOfficer* che ha ricevuto un *EmergencyReport*, ha creato un *Incident* e vi ha assegnato delle *Resource*. L'*Acknowledgment* contiene le risorse associate e il loro tempo di arrivo stimato |

| Nome | ReportEmergency |
| --- | --- |
| Condizioni di entrata | 1. Il FieldOfficer attiva la funzione “Report Emergency” dal proprio terminale. |
| Flusso eventi: | 3. FRIEND risponde presentando una form al FieldOfficer. La form include un menu del tipo di emergenza (emergenza generale, incendio, trasporto) e la località, la descrizione dell’incidente, la richiesta di risorse, i campi dei materiali nocivi.<br>4. Il FieldOfficer riempie la form specificando al minimo il tipo di emergenza e i campi descrizione. Il FieldOfficer descrive anche possibili risposte alla situazione di emergenza e può richiedere risorse specifiche. Appena finito il FieldOfficer invia la form premendo il pulsante “Send Report”, e il Dispatcher viene notificato.<br>4. Il Dispatcher rivede le informazioni sottomesse dal FieldOfficer e crea un Incidente nel DB invocando il caso d’uso OpenIncident. Tutte le informazioni contenute nella form del FieldOfficer sono incluse automaticamente nell’Incident. Il Dispatcher seleziona una risposta allocando le risorse all’Incidente (con il caso d’uso AllocateResources) e notifica il rapporto di emergenza inviando un breve messaggio al FieldOfficer. **L’Acknowledgment indica al FieldOfficer che l’EmergencyReport è stato ricevuto, un Incident creato e le Resource allocate all’Incident. L’Acknowledgment include le risorse ed il loro tempo di arrivo stimato.** |
| Condizioni di uscita | Il FieldOfficer riceve l’accettazione e la risposta selezionata |

# Diagramma di sequenze: distribuzione del cambiamento

Con i diagrammi di sequenze non solo si modella l'ordine delle interazioni fra gli oggetti ma si distribuisce anche il comportamento del caso d'uso

- Si attribuiscono le responsabilità ad ogni oggetto sotto forma di insieme di operazioni
- Queste operazioni possono essere condivise da qualsiasi caso d'uso in cui un dato oggetto partecipa
- E' da osservare che la definizione di un oggetto condiviso attraverso due o più casi d'uso dovrebbe essere identica
- Se un'operazione appare in più di un diagramma di sequenze, il suo comportamento dovrebbe essere identico

# Diagramma di sequenze: condivisione operazioni

Condividere le operazioni attraverso i casi d'uso consente agli sviluppatori di eliminare le ridondanze nella specifica dei requisiti e di migliorarne la consistenza

Alla chiarezza bisognerebbe sempre dare la precedenza rispetto all'eliminazione della ridondanza

Frammentare il comportamento attraverso molte operazioni complica inutilmente la specifica dei requisiti

# Diagramma di sequenze: uso nell'analisi

Nell'analisi, i diagrammi delle sequenze sono usati per aiutare ad identificare nuovi oggetti partecipanti e comportamenti mancanti

Questioni legate all'implementazione come le prestazioni non dovrebbero essere affrontate in questo punto

I diagrammi delle sequenze richiedono molto tempo per cui gli sviluppatori dovrebbero focalizzarsi prima su funzionalità **problematiche** o **non specificate**

Disegnare diagrammi delle sequenze per parti semplici o ben definite del sistema non rappresenta un buon investimento di risorse

# Euristiche per i diagrammi delle sequenze

## Layout

- Colonna 1: attore del caso d'uso
- Colonna 2: oggetto *boundary*
- Colonna 3: oggetto *control* che gestisce il resto del caso d'uso

## Creazione degli oggetti

- Creare gli oggetti *control* all'inizio del flusso di controllo
- Gli oggetti *control* creano gli oggetti *boundary*

## Accesso degli oggetti

- Gli oggetti *entity* sono acceduti dagli oggetti *control* e *boundary*
- Gli oggetti *entity* non dovrebbero accedere ad oggetti *control* e *boundary*

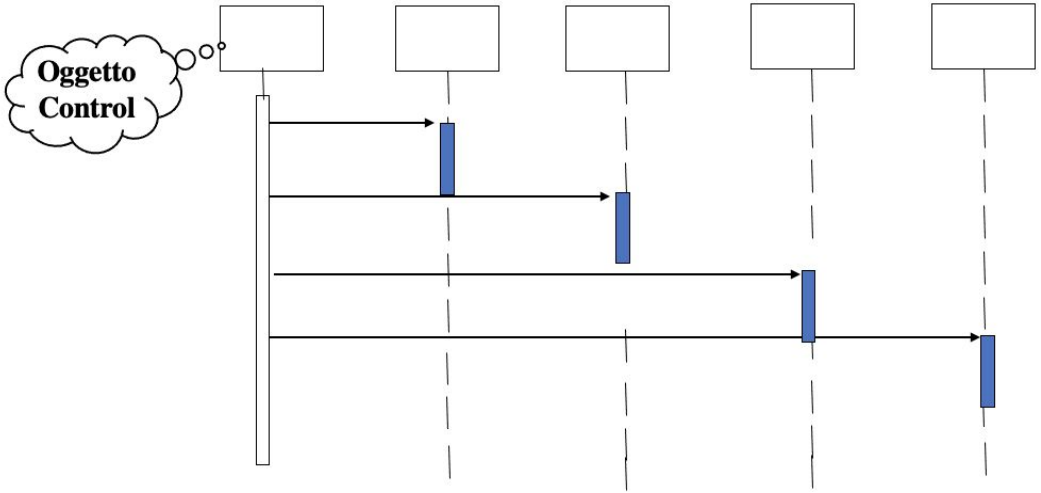
# Cosa possiamo aggiungere sui diagrammi delle sequenze?

- I diagrammi delle sequenze sono derivati dai casi d'uso
- La struttura del diagramma delle sequenze ci aiuta a determinare quanto è decentralizzato il sistema
- Distinguiamo due strutture per I diagrammi delle sequenze
- Fork Diagrams and Stair Diagrams (Ivar Jacobsen)

# Fork Diagram

Il comportamento dinamico è posto in un singolo oggetto, solitamente l'oggetto control

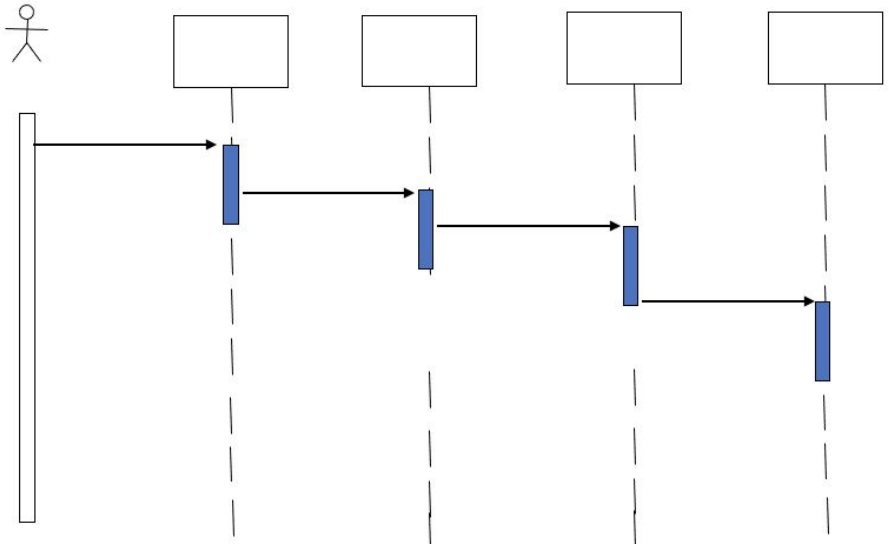
•Esso conosce tutti gli altri oggetti e spesso li usa per domande e comandi diretti



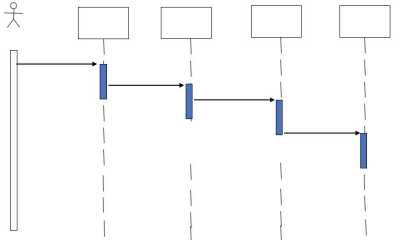
# Stair Diagram

Il comportamento dinamico è distribuito. Ogni oggetto delega la resposanbilità ad altri oggetti

•Ogni oggetto conosce solo pochi altri oggetti e sa quali oggetti possono aiutare con un comportamento specifico



# Fork o Stair?

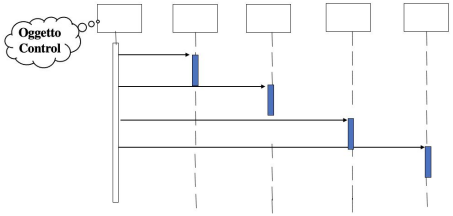


I fan “orientati agli oggetti” sostengono che la struttura stair sia migliore

Consiglio di modellazione:

Scegliere stair - una struttura di controllo decentralizzata - se

- Le operazioni hanno una forte connessione
- - Le operazioni saranno sempre eseguite nello stesso ordine




Scegliere fork - una struttura di controllo centralizzata - se

- Le operazioni possono cambiare ordine
- Ci si aspetta che possano essere aggiunte nuove operazioni a seguito di nuovi requisiti

# Quale modello è dominante?

Modello ad oggetti:

Il Sistema ha classi con stati non banali  molte relazioni tra le classi

Modello dinamico:

Il modello ha molti tipi di eventi diversi: input, output, eccezioni, errori, ecc.

Modello funzionale:

Il modello esegue trasformazioni complicate (cioé computazioni che consistono di molti passi)

Quale modello è dominante in queste applicazioni?

- Compilatore
- Database
- Spreadsheet

# Esempio di modelli dominanti

Compilatore:

- Il modello funzionale è più importante
- Il modello dinamico è banale poiché c'è solo un tipo di input  solo pochi output

Database:

- Il modello ad oggetti è il più importante
- Il modello funzionale è banale, poiché lo scopo delle funzioni è memorizzare, organizzare e recuperare i dati

Spreadsheet:

- Il modello funzionale è il più importante
- Il modello dinamico è interessante se il programma consente di fare calcoli sulle celle
- Il modello ad oggetti è banale

# Modellare il comportamento dipendentemente dallo stato degli oggetti

•I diagrammi delle sequenze sono usati per distribuire il comportamento tra gli oggetti e per identificare le operazioni

•I diagrammi delle sequenze rappresentano il comportamento del sistema dalla prospettiva di un caso d'uso singolo

•I **diagrammi degli stati** rappresentano il comportamento dal punto di vista di un singolo oggetto

•Vedere il comportamento dalla prospettiva di ogni oggetto consente allo sviluppatore di costruire una descrizione più formale del comportamento dell'oggetto, e conseguentemente, di identificare casi d'uso mancanti

# Modellare il comportamento dipendentemente dallo stato degli oggetti

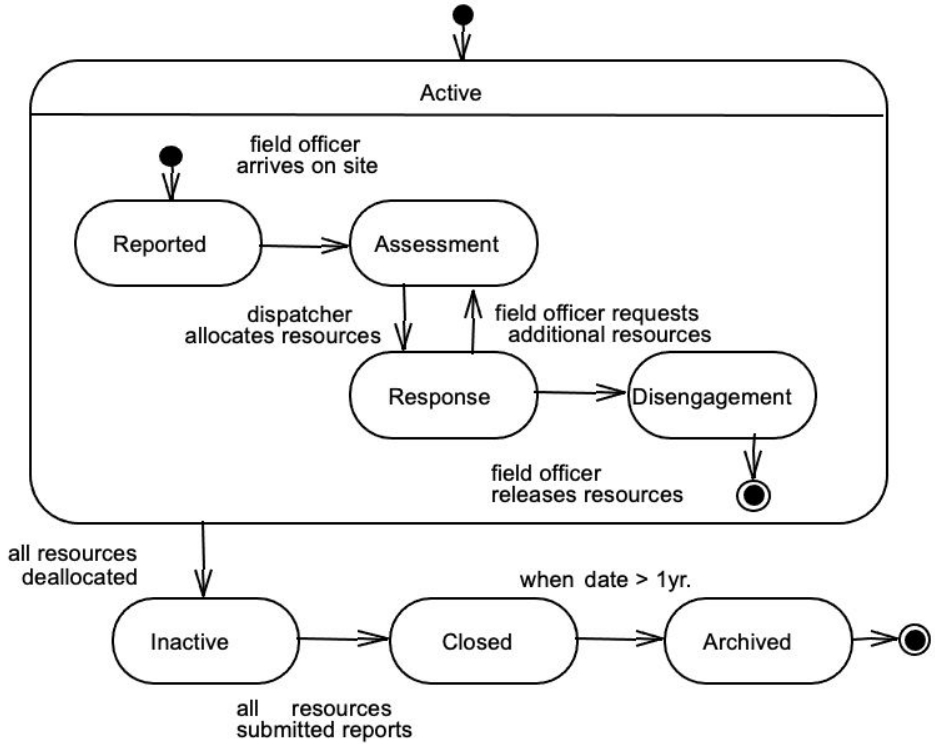
•Focalizzandosi sugli stati, gli sviluppatori possono identificare nuovi comportamenti

•E’ da osservare che non è necessario costruire diagrammi degli stati per ogni classe del sistema

•E’ importante considerare solo gli oggetti con un arco di vita esteso e con un comportamento che dipende dallo stato

•Questa è quasi sempre la regola per gli oggetti control, meno spesso per gli oggetti entity e quasi mai il caso per gli oggetti boundary

# Diagrammi degli stati per Incident



# Rivedere il modello di analisi

Il modello di analisi è costruito incrementalmente ed iterativamente

Raramente si ottiene un modello corretto o anche completo al primo passo

Sono necessarie numerose iterazioni con clienti ed utenti prima che il modello di analisi converga verso una specifica corretta, usabile dagli sviluppatori per il progetto e l'implementazione

Una volta che il modello di analisi diventa stabile (cioè, quando il numero di modifiche al modello sono minimali e l'ambito delle modifiche è localizzato), viene prima rivisto dagli sviluppatori (revisione interna) e poi anche con i clienti

L'obiettivo della revisione è assicurare che la specifica dei requisiti sia corretta, completa, consistente e non ambigua

# Rivedere il modello di analisi

Sviluppatori e clienti effettuano la revisione anche se i requisiti sono realistici e verificabili

Gli sviluppatori dovrebbero essere preparati a scoprire errori a valle ed effettuare le modifiche alle specifiche

Tuttavia, sarebbe importante individuare quanti più errori possibili nei requisiti a monte

La revisione può essere facilitata da una lista di controllo o una lista di domande

# Domande da porsi perchè il modello sia corretto

- Il glossario degli oggetti entità è comprensibile all'utente?
- Le classi astratte corrispondono ai concetti a livello utente?
- Le descrizioni sono tutte in accordo con le definizioni dell'utente?
- Tutti gli oggetti entity e boundary hanno predicati nominali significativi come nomi?
- I casi d'uso e gli oggetti control hanno tutti i predicati verbali significativi come nomi?
- I casi d'errore sono tutti descritti e gestiti?

# Domande da porti perchè il modello sia completo

- Per ogni oggetto: è necessario per un caso d'uso? In quale caso d'uso è creato? Modificato? Distrutto? Può essere acceduto da un oggetto boundary?
- Per ogni attributo: quando è impostato? quale è il suo tipo?
- Per ogni associazione: quando è attraversata? Perché è stata scelta una specifica molteplicità?
- Per ogni oggetto control: ha le necessarie associazioni per accedere gli oggetti che partecipano nel caso d'uso corrispondente?

# Domande da porsi perchè il modello sia consistente

- Ci sono classi o casi d'uso multipli con lo stesso nome?
- Entità (casi d'uso, classi, attributi) con lo stesso nome denotano concetti simili?
- Ci sono oggetti con attributi e associazioni simili che non sono nella stessa gerarchia di generalizzazione?

# Domande da porsi perchè il modello sia realistico

- Ci sono caratteristiche nuove nel sistema? Sono stati costruiti prototipi o eseguiti studi per assicurarne la fattibilità?

- I requisiti delle prestazioni e dell'affidabilità possono essere soddisfatti? Questi requisiti sono stati verificati da qualche prototipo eseguito su hardware selezionato?

# Domande per l'analisi dei requisiti

1. Quali sono le trasformazioni? → **Modellazione funzionale**

   Creare *scenari e diagrammi dei casi d'uso*

   - Parlare con il cliente, osservare, acquisire informazioni da archivi

2. Qual è la struttura del sistema? → **Modellazione degli oggetti**

   Creare i *diagrammi delle classi*

   - Identificare gli oggetti
   - Quali sono le associazioni tra loro?
   - Qual è la loro molteplicità?
   - Quali operazioni sono definite sugli oggetti?

3. Qual è il comportamento? → **Modellazione dinamica**

   Creare i diagrammi delle sequenze

   - Identificare mittenti e destinatari
   - Mostrare sequenze di eventi scambiati tra gli oggetti
   - Identificare le dipendenze tra eventi e concorrenza di eventi

   Creare i diagrammi degli stati

   - Solo per oggetti dinamicamente interessanti

# Analisi in pratica

## 1. Analizzare la definizione del problema

- Identificare I requisiti funzionali
- Identificare I requisiti non funzionali
- Identificare I vincoli (pseudo requisiti)

## 2. Costruire il modello funzionale:

- Sviluppare I casi d'uso per illustrare i requisiti delle funzionalità

## 3. Costruire il modello dinamico:

- Sviluppare I diagrammi delle sequenze per illustrare le interazioni tra gli oggetti
- Sviluppare I diagrammi di stato per gli oggetti con comportamento interessante

## 4. Costruire il modello ad oggetti:

- Sviluppare I diagrammi delle classi che mostrano la struttura del sistema

# Identificare le associazioni

Un’associazione mostra una relazione fra due o più classi

Ad esempio, un *FieldOfficer* scrive un *EmergencyReport*

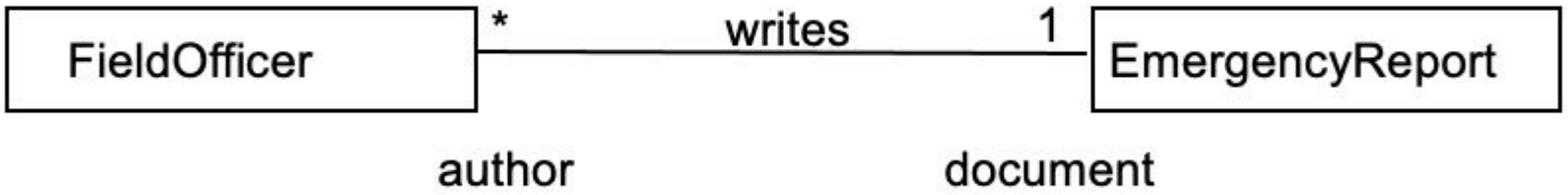
Identificare le associazioni comporta due vantaggi

- Chiarisce il modello di analisi rendendo esplicite le relazioni tra gli oggetti (es., *EmergencyReport* può essere creato da un *FieldOfficer* o un *Dispatcher*)
- Consente agli sviluppatori di scoprire casi limite associati con i collegamenti

I casi limite sono eccezioni che devono essere chiarite nel modello

Ad esempio, è intuitivo assumere che più di un *EmergencyReport* è scritto da un *FieldOfficer*. Tuttavia, è lecito chiedersi se il sistema dovrebbe supportare *EmergencyReport* scritti da più di una persona o consentire *EmergencyReport* anonimi

# Esempio di associazioni tra classi FieldOfficer e EmergencyReport



# Identificare le associazioni

Le associazioni hanno diverse proprietà:

- Un **nome** per descrivere le associazioni tra due classi (*write* nella figura precedente)
- I nomi delle associazioni sono opzionali e non necessariamente devono essere globalmente unici
- Un **ruolo** in ciascuna estremità, identificando la funzione di ogni classe rispetto alle associazioni (ad esempio, *author* è il ruolo del *FieldOfficer* nell'associazione *writes*)
- Una **molteplicità** in ciascuna estremità, identificando il numero di possibili istanze (es., * indica che un *FieldOfficer* può scrivere zero o più *EmergencyReport*, mentre 1 indica che ogni *EmergencyReport* ha esattamente un *FieldOfficer* come autore)

# Identificare le associazioni

Le associazioni fra gli oggetti sono le più importanti

- Rivelano molte informazioni sul dominio applicativo

In accordo alle euristiche di Abbott, le associazioni possono essere identificate esaminando verbi e predicati verbali che denotano uno stato (es., *ha*, *è parte di*, *gestisce*, *si rapporta a*, *è innescato da*, *è contenuto in*, *parla a*, *include*)

# Euristiche per identificare le associazioni

Esaminare i predicati verbali

Assegnare nomi e ruoli alle associazioni in modo preciso

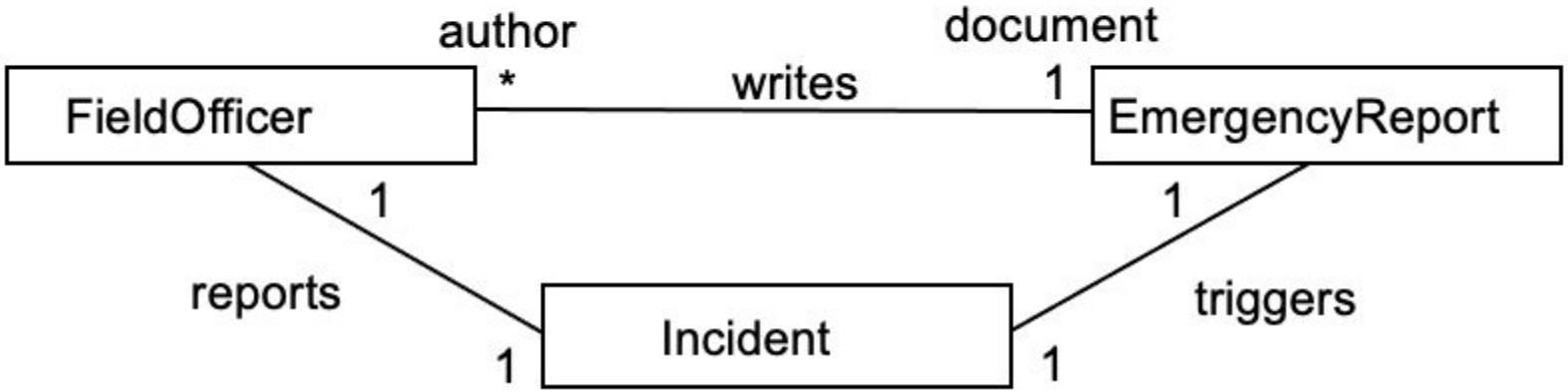
Eliminare qualsiasi associazione che deriva da altre associazioni

Non preoccuparsi delle molteplicità fino a che l'insieme delle associazioni è stabile

Troppe associazioni rendono un modello illeggibile

# Eliminare le associazioni ridondanti

Il modello ad oggetti include, inizialmente, troppe associazioni se gli sviluppatori includono tutte le associazioni identificate dopo aver esaminato i predicati verbali



# Relazione di dipendenza

Il client utilizza e dipende dal fornitore. Quindi una modifica al fornitori influisce anche a una modifica del client.

# Identificazione delle associazioni

Molti oggetti entity hanno una caratteristica che li identifica

- usata dagli attori per accederli

*FieldOfficer* e *Dispatcher* hanno un numero di badge

*Incident* e *Report* hanno associati dei numeri e sono archiviati per data

Una volta che il modello di analisi include molte classi e associazioni, gli sviluppatori dovrebbero esaminare ogni classe e controllare il modo in cui sono identificate dagli attori e in che contesto

# Esempio

I numeri di badge dei FieldOfficer sono unici in tutto l'universo dell'applicazione? In una città? Una stazione di polizia?

Se sono unici nell'ambito di una città, il sistema FRIEND può conoscere i FieldOfficer di più città?

Questo approccio può essere formalizzato esaminando ogni classe ed identificando la sequenza di associazioni che devono essere attraversate per accedere a specifiche istanze di quella classe

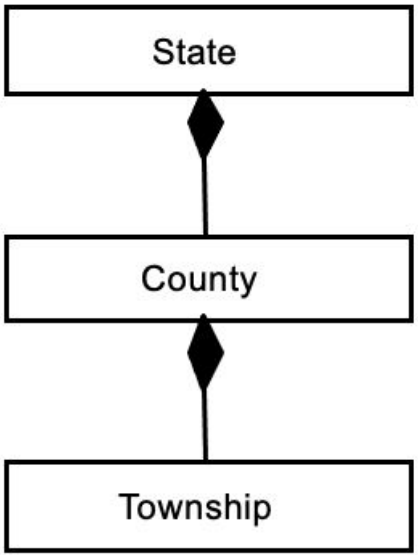
# Identificare le aggregazioni

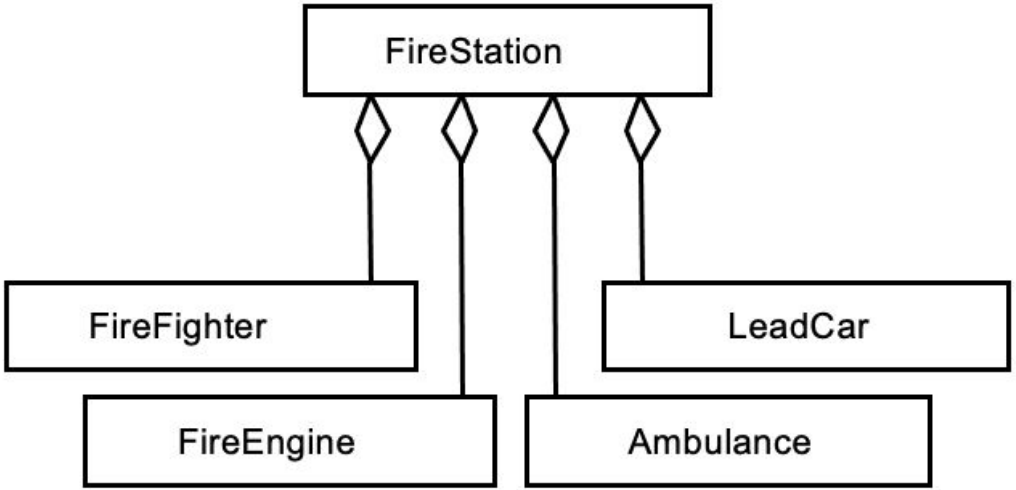
Le aggregazioni sono speciali tipi di associazioni che denotano una relazione *tutto-parte*

- Una *FireStation* consiste di un numero di *FireFighter*, *FireEngine*, *Ambulance* e una *LeadCar*
- Uno stato è composto da un numero di regioni che, a loro volta, sono composte da un numero di città

Un'aggregazione è denotata con un diamante sull'estremità della parte *tutto*

# Esempi di aggregazioni e composizioni



# Aggregazioni: composizioni e condivisione

Esistono due tipi di aggregazioni

- Composizioni, denotate da un diamante pieno
- Condivise, denotate da un diamante vuoto

Un’aggregazione di composizione indica che l’esistenza delle parti dipende dal tutto

Una regione è parte esattamente di uno stato

Una città è parte esattamente di una regione

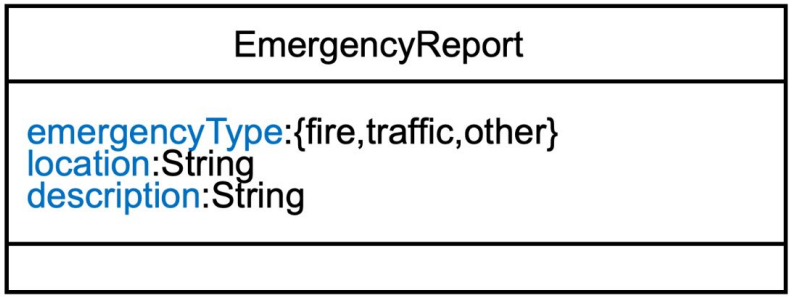
Un’aggregazione condivisa indica che il tutto e la parte possono esistere in modo indipendente

Sebbene una *FireEngine* sia parte di al più una *FireStation* alla volta, essa può essere riassegnata a differenti *FireStation* durante il suo tempo di vita

# Identificare gli attributi

Gli attributi sono proprietà degli oggetti

- *EmergencyReport* ha le proprietà *tipo*, *località* e *descrizione*




- Immesse *dal FieldOfficer* quando rapporta un'emergenza e mantenute dal sistema

Quando si identificano le proprietà degli oggetti dovrebbero essere considerati solo gli attributi rilevanti per il sistema

- *FieldOfficer* ha un **codice fiscale che non è rilevante per il sistema di gestione emergenze** al contrario del *badgeNumber* che è usato dal sistema per identificare i *FieldOfficer*

# Attributi

Gli attributi hanno:

- Un nome che li identifica in un oggetto

(EmergencyReport può avere un attributo reportType e un attributo emergencyType)

- Una breve descrizione
- Un tipo che descrive i valori leciti che può assumere

(String, integer ecc.)

# Ancora sugli attributi

Gli attributi rappresentano la parte meno stabile del modello ad oggetti

Spesso sono scoperti o aggiunti tardi nello sviluppo quando il sistema è valutato dagli utenti

Gli attributi aggiunti (se non relativi a nuove funzionalità) non comportano cambiamenti radicali nella struttura dell'oggetto (e del sistema)

- Gli sviluppatori non necessitano di impiegare troppe risorse nel'identificare e dettagliare gli attributi che rappresentano aspetti meno importanti del sistema

# Euristiche per identificare gli attributi

Esaminare frasi possessive

Rappresentare stati memorizzati come attributi di oggetti entity

Descrivere ogni attributo

Non rappresentare un attributo come un oggetto; usare invece un'associazione

Non sprecare tempo nel descrivere dettagli prima che la struttura dell'oggetto sia stabile

# Modellare relazioni di ereditarietà tra oggetti

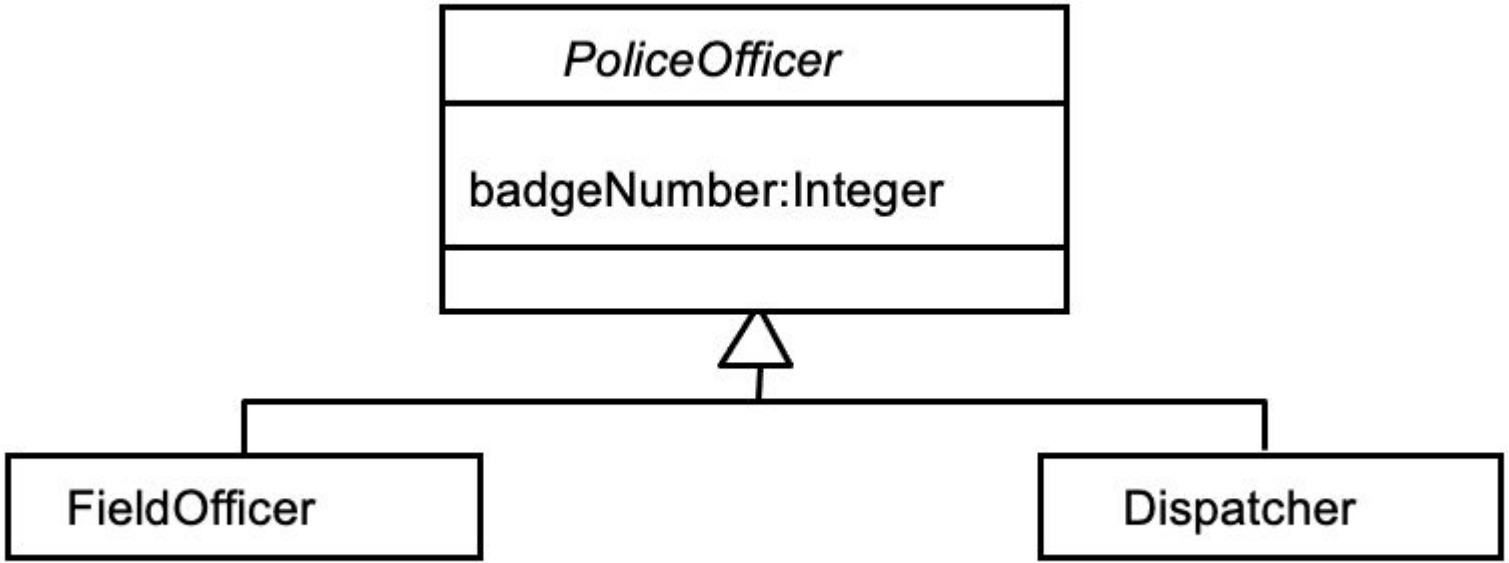
La generalizzazione è usata per eliminare la ridondanza dal modello di analisi

Se due o più classi condividono attributi o un comportamento, le similitudini sono consolidate in una superclasse

- *Dispatcher* e *FieldOfficer* hanno entrambi un attributo *badgeNumber* che serve per identificarli nel contesto di una città
- *FieldOfficer* e *Dispatcher* sono entrambi *PoliceOfficer* a cui sono assegnate funzioni diverse

Per modellare esplicitamente questa similitudine si introduce una classe astratta *PoliceOfficer* da cui le classi *FieldOfficer* e *Dispatcher* ereditano

# Un esempio di relazione di ereditarietà



# Sommario dell'analisi

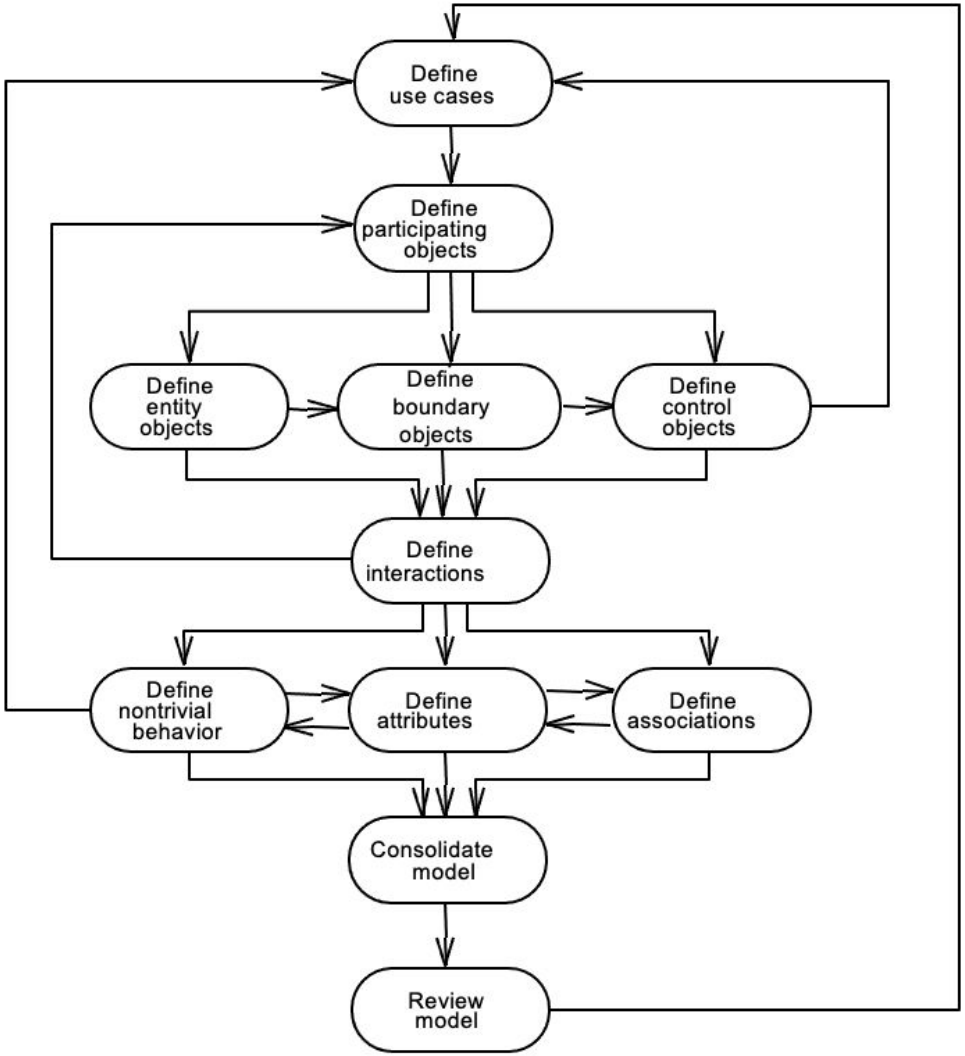
L'attività di scoperta dei requisiti è altamente iterativa e incrementale

Sono delineate e proposte funzionalità a utenti e clienti

- Il cliente aggiunge requisiti, critica le funzionalità esistenti e modifica i requisiti esistenti
- Gli sviluppatori investigano i requisiti non funzionali attraverso i prototipi e studi della tecnologia e valutano ogni requisito proposto

Inizialmente, la scoperta dei requisiti somiglia ad un'attività di brainstorming

Non appena la descrizione del sistema cresce ed i requisiti diventano più concreti gli sviluppatori devono estendere e modificare il modello di analisi in maniera più ordinata per gestire la complessità delle informazioni




# Attività di Analisi

# Template del Documento di Analisi dei Requisiti

1. Introduzione
2. Sistema attuale
3. Sistema proposto
   - 3.1 Overview
   - 3.2 Requisiti funzionali
   - 3.3 Requisiti non funzionali
   - 3.4 Vincoli (“Pseudo requisiti”)
   - 3.5 Modello del sistema
     - 3.5.1 Scenari
     - 3.5.2 Modello dei casi d’uso
     - 3.5.3 Modello degli oggetti
       - 3.5.3.1 Dizionario dati
       - 3.5.3.2 Diagramma delle classi
     - 3.5.4 Modelli dinamici
     - 3.5.5 Interfaccia utente
4. Glossario

# Sezione 3.5 Modello del Sistema

3.5.1 Scenari

- Scenari As-is, scenari visionari

3.5.2 Modello dei casi d'uso

- Attori e casi d'uso

3.5.3 Modello degli oggetti

- Dizionario dei dati
- Diagramma delle classi (classi, associazioni, attributi e operazioni)

3.5.4 Modello dinamico

- Diagramma degli stati per classi con comportamento dinamico significativo
- Diagrammi delle sequenze per gli oggetti che collaborano

3.5.5 Interfaccia utente

- Percorsi di navigazione, Screen mockup