DESIGNING SOFTWARE: TIPS AND TRICKS

Prof. Mariacarla Staffa

OVERVIEW

01

Scoperta dei requisiti

02

Analisi e modellazione a oggetti

03

Analisi e modellazione dinamica

04

Progettazione del sistema

SCOPERTA DEI REQUISITI

ATTIVITÀ DI SCOPERTA DEI REQUISITI

- Le attività di scoperta dei requisiti fanno corrispondere la definizione di un problema ad una specifica dei requisiti, che può essere presentata come un insieme di
  - Attori
  - Scenari
  - Casi d'uso
- Le attività di scoperta dei requisiti includono
  - Identificazione degli attori
  - Identificazione degli scenari
  - Identificazione dei casi d'uso
  - Identificazione delle relazioni tra attori e casi d'uso
  - Identificazione degli oggetti iniziali dell'analisi
  - Identificazione dei requisiti non funzionali

IDENTIFICAZIONE DEGLI ATTORI

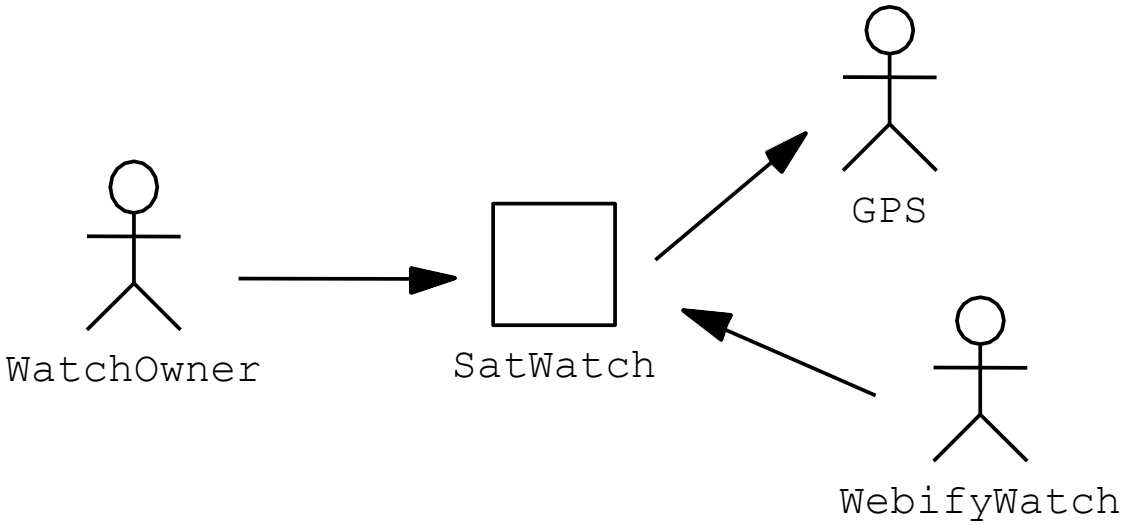
- Gli attori rappresentano entità esterne che interagiscono con il sistema
  - Può essere un utente o un sistema esterno
- Nell'esempio ***SatWatch***, il **proprietario** dell'orologio, il satellite **GPS** e il dispositivo seriale **Webify Watch** sono attori
  - Tutti scambiano informazioni con *SatWatch*
  - E' da osservare che tutti hanno interazioni specifiche con *SatWatch*:
    - Il proprietario indossa e guarda l'orologio
    - L'orologio monitora il segnale dal satellite GPS
    - Webify Watch scarica nuovi dati nell'orologio

ATTORI PER IL SISTEMA SATWATCH

WatchOwner indossa l'orologio, si sposta (eventualmente attraverso diversi fusi) e lo consulta per conoscere l'ora

GPS interagisce con l'orologio per il calcolo della posizione

WebifyWatch aggiorna i dati contenuti nell'orologio per riflettere le modifiche dell'orario

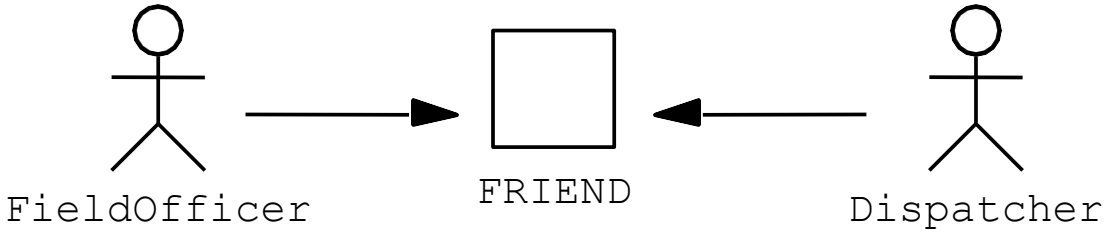


IDENTIFICAZIONE DEGLI ATTORI IN FRIEND

- Consideriamo l'esempio del sistema **FRIEND**, un sistema informativo distribuito per la gestione degli incidenti
- Include molti attori
  - FieldOfficer, gli ufficiali di polizia e dei vigili del fuoco che rispondono ad un incidente
  - Dispatcher, l'ufficiale di polizia responsabile di rispondere alle chiamate del 911 e di allocare le risorse ad un incidente
- **FRIEND** supporta ambo gli attori tenendo traccia degli incidenti, delle risorse e dei piani delle attività
  - Inoltre, supporta l'accesso a database multipli, come db dei materiali nocivi e delle procedure delle operazioni di emergenza
  - Gli attori **FieldOfficer** e **Dispatcher** interagiscono attraverso varie interfacce
    - I **FieldOfficer** accedono a FRIEND attraverso un laptop
    - I **Dispatcher** accedono a FRIEND attraverso una workstation

ATTORI DEL SISTEMA FRIEND

I ***FieldOfficer*** non solo hanno accesso a differenti funzionalità, ma usano anche differenti computer per accedere al sistema



IDENTIFICAZIONE DEGLI ATTORI

- Attori e ruoli sono astrazioni e non necessariamente corrispondono a persone
  - La stessa persona può ricoprire il ruolo di **FieldOfficer** e **Dispatcher** in tempi diversi
  - Le funzionalità a cui accedono sono sostanzialmente differenti
    - Quindi, i due ruoli sono modellati con due attori differenti

SCOPERTA DEI REQUISITI E IDENTIFICAZIONE ATTORI

- Il primo passo nella scoperta dei requisiti è l'identificazione degli attori
  - Serve per definire i confini del sistema e per trovare tutte le prospettive da cui gli sviluppatori necessitano di considerare il sistema
- Quando il sistema è inserito in un'organizzazione esistente, solitamente esistono molti attori prima che il sistema sia sviluppato:
  - corrispondono a ruoli nell'organizzazione

SCOPERTA DEI REQUISITI E IDENTIFICAZIONE ATTORI

- Durante la fase iniziale dell'identificazione degli attori, è difficile distinguere gli attori dagli oggetti
  - Ad esempio, un sottosistema **DB** può essere un attore in un frangente e parte del sistema in un altro
- Una volta definito il confine del sistema, non ci sono più problemi nella distinzione
  - Gli attori sono fuori dal confine del sistema, **sono esterni**
  - I sottosistemi e gli oggetti sono all'interno del confine del sistema, **sono interni**
  - Ogni sistema software **esterno** che usa il sistema da sviluppare è un attore

DOMANDE PER IDENTIFICARE GLI ATTORI

Quali gruppi di utenti sono supportati dal sistema per eseguire il proprio lavoro?

Quali gruppi di utenti eseguono le funzioni principali del sistema?

Quali gruppi di utenti eseguono le funzioni secondarie, come manutenzione e amministrazione?

Con quali sistemi software e hardware esterni interagirà il sistema?

ATTORI DEL SISTEMA FRIEND

- Nell'esempio FRIEND, queste domande hanno portato ad una lunga lista di potenziali attori:
  - Vigile del fuoco, ufficiale di polizia, dispatcher, investigatore, sindaco, governatore, db per materiali pericolosi, amministratore di sistema ecc.
  - Dopo è necessario consolidare questa lista in un piccolo numero di attori, che differiscono dal punto di vista dell'uso del sistema
    - Ad esempio, un **Vigile del fuoco** ed un **Ufficiale di polizia** possono condividere la stessa interfaccia al sistema, poiché sono entrambi coinvolti con un singolo incidente in loco
    - Un **Dispatcher**, gestisce più incidenti concorrenti e richiede l'accesso ad un maggiore volume di informazioni
    - Il Governatore ed il Sindaco non interagiscono direttamente col sistema ma usufruiscono dei servizi di un operatore addestrato

IDENTIFICAZIONE DEGLI SCENARI

Uno scenario è una descrizione concreta, mirata e informale di una singola funzionalità del sistema dal punto di vista di un singolo attore

Gli scenari non possono (e non sono concepiti per) sostituire i casi d'uso, in quanto mirano su specifiche istanze ed eventi concreti

Gli scenari, comunque, potenziano la scoperta dei requisiti fornendo uno strumento comprensibile agli utenti ed ai clienti

Nome Scenario	warehouseOnFire
Istanze attori partecipanti	bob, alice :FieldOfficer John :Dispatcher
Flusso di eventi	1. Bob, guidando lungo la strada principale nella sua patrol, osserva del fumo proveniente da un magazzino. Il suo partner, Alice, attiva la funzione “Report Emergency” dal laptotp del FRIEND. 2. Alice immette l’indirizzo dell’edificio, una breve descrizione della sua ubicazione (angolo nord-ovest) ed un livello di emergenza. In aggiunta ad un’unità dei vigili del fuoco, richiede diverse unità paramediche poiché la zona non è molto trafficata. Conferma l’input ed attende l’accettazione. 3. John, il Dispatcher, è allertato per l’emergenza da un beep dalla sua stazione di lavoro. Rivede le informazioni sottomesse da Alice e accetta il rapporto. Alloca un’unità dei vigili del fuoco e due unità paramediche sul sito dell’Incidente ed invia la stima del tempo di arrivo (ETA) ad Alice. 4. Alice riceve l’accettazione e l’ETA.

STESURA DEGLI SCENARI

- Nella scoperta dei requisiti, sviluppatori e utenti scrivono e rifiniscono una serie di scenari con lo scopo di migliorare la comprensione condivisa di cosa il sistema dovrebbe essere

- Inizialmente, ogni scenario può essere ad alto livello ed incompleto come nel caso del *warehouseOnFire*

DOMANDE PER IDENTIFICARE GLI SCENARI

Quali sono i task che l'attore vuole che il sistema esegua?

A quali informazioni l'attore accede?

- Chi crea i dati?
- Possono essere modificati o rimossi?
- Da chi?

Quali modifiche esterne l'attore deve notificare al sistema?

- Quanto spesso?
- Quando?

Quali eventi il sistema deve notificare l'attore?

- Con quale latenza?

SORGENTI DI INFORMAZIONI PER GLI SCENARI

Gli sviluppatori usano i documenti esistenti sul dominio applicativo per rispondere a tali questioni

Questi documenti includono **manuali utente** di sistemi precedenti, manuali delle procedure, standard della compagnia, note utente, interviste con utenti e clienti

Gli sviluppatori dovrebbero sempre scrivere gli scenari usando la terminologia del dominio applicativo

FRIEND E SCENARI

- Nel sistema FRIEND, identifichiamo quattro scenari che abbracciano i tipi di task che ci si aspetta il sistema supporti
  - warehouseOnFire: E' individuato un incendio in un magazzino; due ufficiali arrivano sulla scena e richiedono le risorse
  - fenderBender: si verifica un incidente automobilistico sull'autostrada senza vittime. Gli ufficiali di polizia documentano l'incidente e gestiscono il traffico mentre i veicoli danneggiati sono portati via dal carro attrezzi
  - catInATree: un gatto è bloccato su un albero. Per il recupero del gatto, è richiamato un un automezzo dei vigili del fuoco. Poiché l'incidente ha una bassa priorità, il mezzo impiega del tempo per arrivare sul luogo. Nel frattempo, il proprietario del gatto, impaziente, si arrampica sull'albero, cade e si frattura una gamba, richiedendo l'intervento di un'ambulanza
  - earthQuake: un terremoto senza precedenti danneggia seriamente edifici e strade, provocando incidenti multipli, innescando l'attivazione di un piano di emergenza nazionale. Il governatore è informato. Le strade danneggiate ostacolano i soccorsi

IDENTIFICAZIONE DEI CASI D'USO

Uno scenario è un'istanza di un caso d'uso

Un caso d'uso specifica tutti i possibili scenari per una data parte di funzionalità

Un caso d'uso è iniziato da un attore.

Dopo l'avvio, un caso d'uso può interagire anche con altri attori

Un caso d'uso rappresenta un flusso di eventi completo attraverso il sistema

Descrive una serie di interazioni collegate, che sono il risultato dell'avvio

Nome	ReportEmergency
Attori Partecipanti	Iniziato dal FieldOfficer Comunica con il Dispatcher
Flusso eventi:	1. Il FieldOfficer attiva la funzione “Report Emergency del terminale 2. FRIEND risponde presentando una form al FieldOfficer 3. Il FieldOfficer riempie la form selezionando il livello di emergenza, tipo, località, breve descrizione della situazione. Il FieldOfficer descrive anche possibili risposte alla situazione di emergenza. Appena finito Il FieldOfficer invia la form 4. FRIEND riceve la form e notifica al Dispatcher 5. Il Dispatcher rivede le informazioni sottomesse e crea un Incidente nel DB invocando il caso d’uso OpenIncident. Il Dispatcher seleziona una risposta e accetta il rapporto. 6. FRIEND visualizza l’accettazione e la risposta selezionata al FieldOfficer
Condizioni di entrata	Il FieldOfficer è loggato in FRIEND
Condizioni di uscita	Il FieldOfficer ha ricevuto un’accettazione e una risposta selezionata dal Dispatcher, OR Il FieldOfficer ha ricevuto una spiegazione indicante perché la transazione non è stata eseguita
Requisiti di qualità	Il rapporto del FieldOfficer è accettato entro 30 secondi La risposta selezionata arriva non più tardi di 30 secondi dopo che è stata inviata dal Dispatcher

ORGANIZZARE I CASI D'USO

- Inizialmente, gli sviluppatori danno i nomi ai casi d'uso, li allegano agli attori che li iniziano e forniscono una descrizione ad alto livello del caso d'uso
- Il nome di un caso d'uso dovrebbe essere una frase verbale che denota l'attore cosa sta cercando di realizzare
  - **ReportEmergency** indica che un attore sta cercando di fornire una rapporto di emergenza al sistema
  - Non è chiamato **RecordEmergency** poiché il nome dovrebbe riflettere la prospettiva dell'attore, non del sistema
  - Non è nemmeno chiamato **AttemptToReportanEmergency** poiché il nome dovrebbe riflettere l'obiettivo del caso d'uso e non l'attività corrente

DESCRIZIONE DEI CASI D'USO

- Allegare i casi d'uso agli attori che li iniziano consente agli sviluppatori di chiarire i ruoli dei diversi utenti
  - Focalizzandosi su chi inizia ogni caso d'uso, gli sviluppatori identificano nuovi attori che precedentemente sono sfuggiti
- La descrizione di un caso d'uso comporta la specifica di quattro campi:
  - Descrizione delle condizioni di entrata e di uscita
  - Descrizione del flusso di eventi
  - Descrizione dei requisiti di qualità

ESEMPIO: *ALLOCATE A RESOURCE*

- **Attori:**
  - *Field Supervisor:* E' l'ufficiale sul luogo dell'emergenza
  - *Resource Allocator:* E' responsabile per l'impegno ed il disimpegno delle risorse gestite dal sistema FRIEND
  - *Dispatcher:* Immette, aggiorna, e rimuove Incidenti, Azioni e Richieste nel sistema
    - Provvede anche a chiudere le pratiche sugli incidenti
  - *Field Officer:* Produce i rapporti sul luogo dell'emergenza

ESEMPIO: *ALLOCATE A RESOURCE*

Nome caso d'uso: AllocateResources

Attori partecipanti:

- Field Officer (Bob e Alice nello Scenario)
- Dispatcher (John nello Scenario)
- Resource Allocator
- Field Supervisor

Condizione di ingresso

- Il Resource Allocator ha selezionato una risorsa disponibile
- La risorsa correntemente non è allocata

Flusso eventi

- Il Resource Allocator seleziona un incidente
- La risorsa è impegnata sull'incidente

Condizione di uscita

- Il caso d'uso termina quando la risorsa è impegnata
- La risorsa selezionata ora è non disponibile per altri incidenti o richieste di risorsa

Requisiti speciali

- Il Field Supervisor è responsabile della gestione delle risorse

GUIDA ALLA SCRITTURA DEI CASI D'USO

Scrivere i casi d'uso è un'arte

Gli analisti imparano con l'esperienza

Analisti diversi tendono a sviluppare stili diversi

Risulta difficile produrre una specifica dei requisiti consistente

E' possibile usare una guida alla stesura dei casi d'uso

GUIDA ALLA STESURA DEI CASI D'USO (I)

I casi d'uso dovrebbero essere definiti con frasi verbali. Il nome del caso d'uso dovrebbe indicare cosa sta cercando di realizzare l'utente (ReportEmergency, OpenIncident, ecc.)

Gli attori dovrebbero essere indicate con dei sostantivi (FieldOfficer, Dispatcher, ecc.)

Il confine del sistema dovrebbe essere chiaro. I passi realizzati dall'attore ed i passi realizzati dal sistema dovrebbero essere chiaramente distinguibili

I passi dei casi d'uso nel flusso di eventi dovrebbero essere descritti con voce attiva. Ciò rende esplicito chi esegue il passo

La relazione di causalità tra i passi successivi dovrebbe essere chiara

GUIDA ALLA STESURA DEI CASI D'USO (II)

- Un caso d'uso dovrebbe descrivere una transazione utente completa (ad esempio, il caso d'uso **ReportEmergency** descrive tutti i passi tra l'avvio del rapporto di emergenza e la ricezione della notifica avvenuta)
- Le eccezioni dovrebbero essere descritte separatamente
- Un caso d'uso non dovrebbe descrivere l'interfaccia utente del sistema. Ciò distoglie l'attenzione dai passi effettivi realizzati dall'utente
- Un caso d'uso non dovrebbe superare una lunghezza di due o tre pagine. Altrimenti, si devono usare le relazioni **include** e **extend** per decomporlo in casi d'uso più piccoli

Cattivo nome: Che cosa sta cercando di realizzare l'utente?

Nome	Accident
Attore che avvia	Iniziato dal FieldOfficer
Flusso eventi:	1. Il FieldOfficer notifica un incidente 2. E' inviata un'ambulanza 3. Il Dispatcher riceve una notifica quando l'ambulanza arriva sul luogo

Causalità: Quali azioni comportano la notifica del FieldOfficer? Voce passiva: Chi invia l'ambulanza?

Transazione incompleta: Cosa fa il FieldOfficer dopo che è stata inviata l'ambulanza?

RIFINIRE I CASI D'USO

- Consideriamo il caso d'uso **ReportEmergency**
  - E' sufficientemente illustrativo per descrivere come FRIEND supporta la creazione dei rapporti di emergenza e per ottenere un feedback generale dagli utenti
  - Tuttavia, esso non fornisce dettagli sufficienti per la specifica dei requisiti
  - ReportEmergency si può ampliare includendo dettagli sul tipo di incidenti noti a FRIEND e interazioni dettagliate che indicano come il Dispatcher notifica il FieldOfficer

Nome	ReportEmergency
Attori Partecipanti	Iniziato dal FieldOfficer Comunica con il Dispatcher
Flusso eventi:	1. Il FieldOfficer attiva la funzione “Report Emergency” del terminale 2. FRIEND risponde presentando una form al FieldOfficer 3. Il FieldOfficer riempie la form selezionando il livello di emergenza, tipo, località, breve descrizione della situazione. Il FieldOfficer descrive anche possibili risposte alla situazione di emergenza. Appena finito Il FieldOfficer invia la form 4. FRIEND riceve la form e notifica al Dispatcher 5. Il Dispatcher rivede le informazioni sottomesse e crea un Incidente nel DB invocando il caso d’uso OpenIncident. Il Dispatcher seleziona una risposta e accetta il rapporto. 6. FRIEND visualizza l’accettazione e la risposta selezionata al FieldOfficer
Condizioni di entrata	Il FieldOfficer è loggato in FRIEND
Condizioni di uscita	Il FieldOfficer ha ricevuto un’accettazione e una risposta selezionata dal Dispatcher, OR Il FieldOfficer ha ricevuto una spiegazione indicante perché la transazione non è stata eseguita
Requisiti di qualità	Il rapporto del FieldOfficer è accettato entro 30 secondi La risposta selezionata arriva non più tardi di 30 secondi dopo che è stata inviata dal Dispatcher

Nome	ReportEmergency
Attori Partecipanti	Iniziato dal FieldOfficer Comunica con il Dispatcher
Flusso eventi:	1. Il FieldOfficer attiva la funzione “Report Emergency” del terminale 2. 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*. 3. 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 4. FRIEND riceve la form e notifica al Dispatcher *con un finestra pop-up*. 5. Il Dispatcher rivede le informazioni sottomesse 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.* 6. FRIEND visualizza l’accettazione e la risposta selezionata al FieldOfficer
Condizioni di entrata	…

EURISTICHE PER SCRIVERE SCENARI E CASI D'USO

Usare gli scenari per comunicare con gli utenti e per validare le funzionalità

Rifinire un singolo scenario per capire le assunzioni sul sistema dell'utente

Definire scenari non molto dettagliati per definire lo scopo del sistema. Validare con l'utente

Usare mock-up solo come supporto visuale; il progetto dell'interfaccia utente dovrebbe avvenire come task separato dopo che la funzionalità è sufficientemente stabile

Presentare all'utente più alternative diverse. Valutare diverse alternative allarga l'orizzonte dell'utente. Generare alternative differenti forza gli sviluppatori a "pensare oltre l'ordinario"

Dettagliare una parte più ampia quando lo scopo del sistema e le preferenze dell'utente sono ben capiti. Validare con l'utente

RIFINIRE I CASI D'USO

L'enfasi di questa attività è la completezza e la correttezza

Gli sviluppatori identificano le funzionalità non coperte dagli scenari e le documentano rifinendo i casi d'uso o scrivendone nuovi

Gli sviluppatori descrivono raramente casi e gestione delle eccezioni così come visti dagli attori

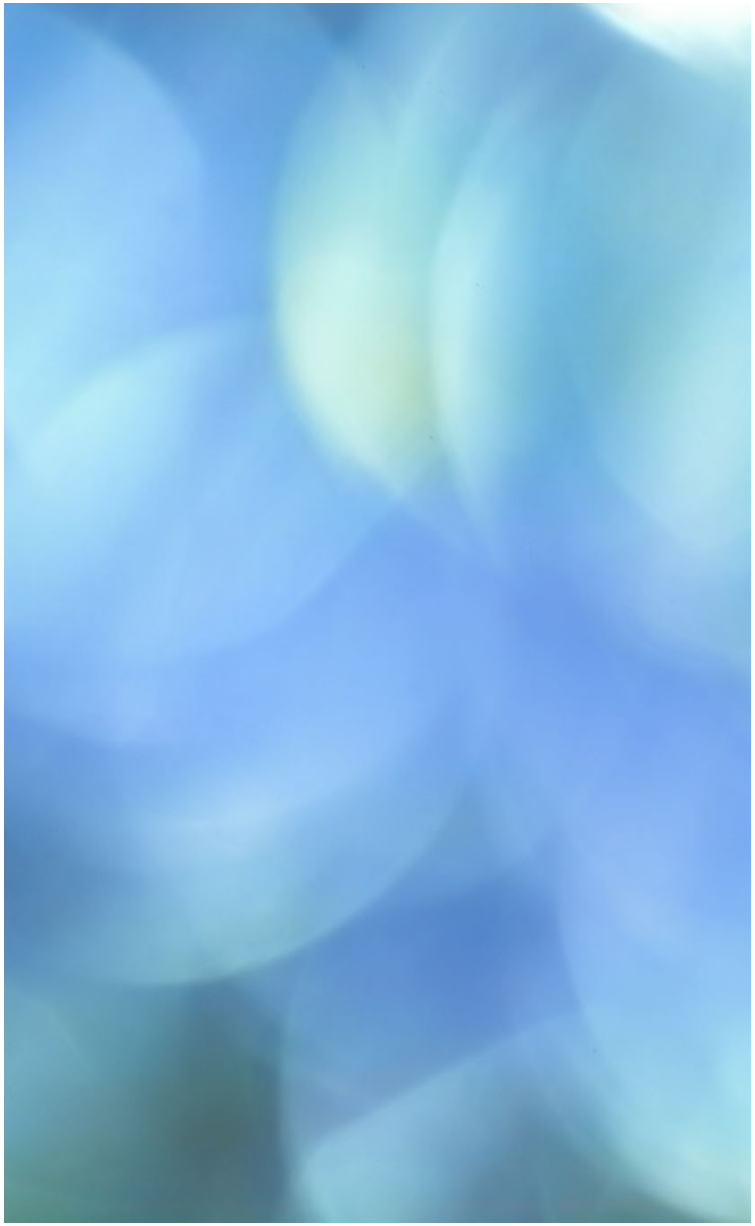
Mentre l'identificazione iniziale dei casi d'uso e degli attori mira a stabilire il confine del sistema, la rifinitura dei casi d'uso porta di volta in volta più dettagli sulle caratteristiche fornite dal sistema e i vincoli ad esse associati

RIFINIRE I CASI D'USO

- Gli aspetti dei casi d'uso che sono inizialmente ignorati e dettagliati durante la rifinitura sono:
  - Gli elementi manipolati dal sistema (in *ReportEmergency* sono stati aggiunti dettagli sugli attributi della form del rapporto di emergenza ed il tipo di incidente)
  - La sequenza a basso livello delle interazioni tra l'attore ed il sistema (sono state aggiunte informazioni su come il *Dispatcher* genera una notifica selezionando le risorse)
  - Permessi di accesso (quale attore può invocare quale caso d'uso)
  - Eccezioni mancanti e la loro gestione
  - Funzionalità comuni tra i casi d'uso

ASSOCIAZIONI NEI CASI D'USO

- Un modello dei casi d'uso consiste di casi d'uso ed associazioni di casi d'uso
- Un'associazione è una relazione tra casi d'uso
- Le associazioni più importanti sono: *Communication*, *Include*, *Extend*, *Generalization*



Identificazione delle relazioni tra attori e casi d'uso

Anche sistemi di medie dimensioni hanno numerosi casi d'uso

Le relazioni tra gli attori ed i casi d'uso permettono a sviluppatori ed utenti di ridurre la complessità del modello ed aumentarne la comprensione

- Si usano **relazioni di comunicazione** tra attori e casi d'uso per descrivere il sistema in livelli di funzionalità
- Le **relazioni di estensione** sono usate per separare flussi di eventi eccezionali da flussi di eventi comuni
- Le **relazioni di inclusione** sono usate per ridurre la ridondanza tra casi d'uso
- Le **generalizzazioni** specializzano casi d'uso astratti

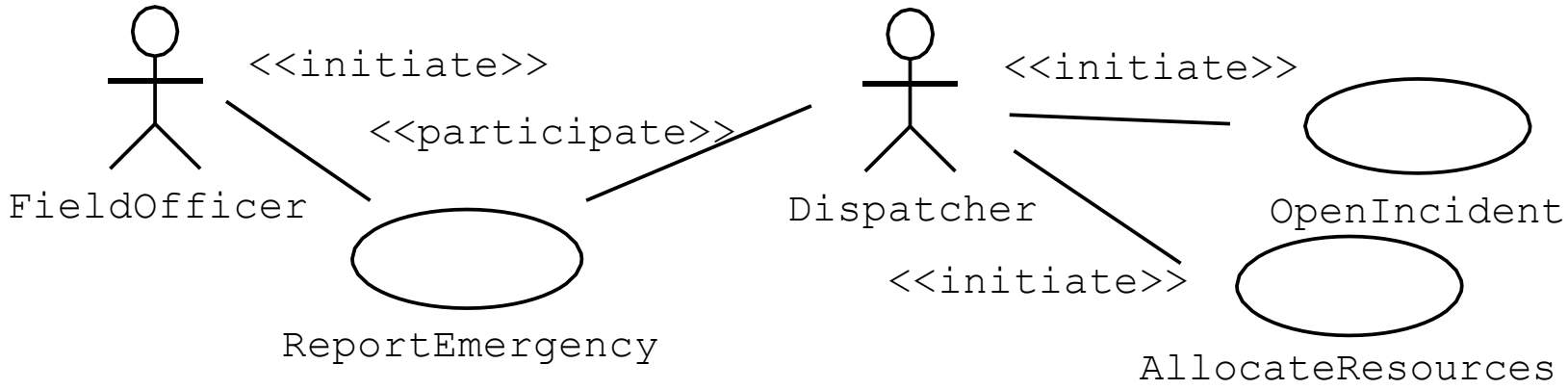
RELAZIONI DI COMUNICAZIONE

Rappresentano il flusso di informazioni durante il caso d'uso

- L'attore che inizia il caso d'uso dovrebbe essere distinto da altri attori con cui il caso d'uso comunica
- Specificando quale attore può invocare uno specifico caso d'uso, specifichiamo implicitamente quale attore non può invocare il caso d'uso
- Similmente, specificando quali attori comunicano con uno specifico caso d'uso, specifichiamo quali attori accedono informazioni specifiche e quali non possono
- In definitiva, documentando l'iniziazione e le relazioni di comunicazioni tra attori e casi d'uso, specifichiamo grosso modo il controllo di accesso per il sistema

Le relazioni tra attori e casi d'uso sono identificate quando sono identificati i casi d'uso

ESEMPIO DI RELAZIONI DI COMUNICAZIONE TRA ATTORI E CASI D'USO IN FRIEND



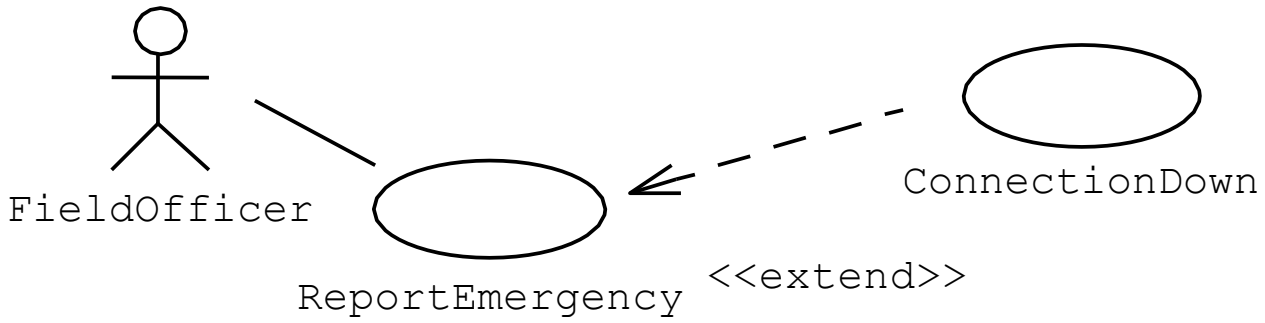
ASSOCIAZIONE EXTEND

- Problema
  - La funzionalità nel problema originale ha la necessità di essere estesa
- Soluzione
  - Un'associazione *extend* dal caso d'uso B al caso d'uso A indica che il caso d'uso B è un'estensione del caso d'uso A

RELAZIONI *EXTEND* TRA CASI D'USO

- Un caso d'uso estende un altro caso d'uso se il caso d'uso esteso può inglobare il comportamento dell'estensione sotto certe condizioni
- In FRIEND, supponiamo che la connessione tra le stazioni di *FieldOfficer* e *Dispatcher* sia interrotta mentre il *FieldOfficer* sta compilando il form (ad esempio, l'auto del *FieldOfficer* entra in un tunnel)
  - La stazione del *FieldOfficer* deve notificare il *FieldOfficer* che il suo form non è stato consegnato e quali misure dovrebbe intraprendere
  - Il caso d'uso *ConnectionDown* viene rappresentato come un'estensione di *ReportEmergency*
    - Le condizioni sotto cui il caso d'uso *ConnectionDown* è iniziato sono descritte in *ConnectionDown* anziché in *ReportEmergency*

UTILIZZO DELLA RELAZIONE *EXTEND*



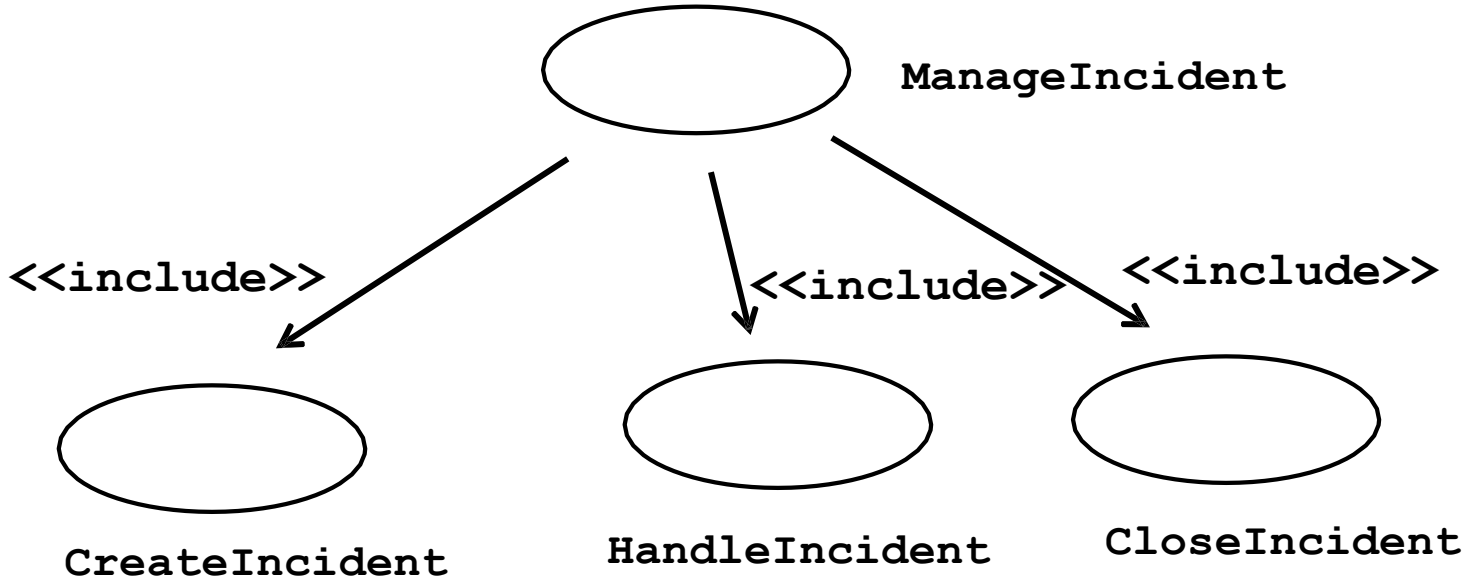
Osservazione: nelle associazioni *extend*, il caso d'uso base può essere eseguito senza la sua estensione

RELAZIONI *EXTEND* TRA CASI D'USO

- Separare flussi di eventi eccezionali ed opzionali dal caso d'uso base comporta due vantaggi
  - Il caso d'uso base è reso più breve e più semplice da capire
  - Il caso d'uso comune è distinto dal caso eccezionale, ciò consente agli sviluppatori di trattare ogni tipo di funzionalità in modo differente
    - Ottimizzare il caso d'uso comune rispetto al tempo di risposta
    - Ottimizzare il caso eccezionale rispetto alla robustezza
  - Ambo i casi d'uso esteso ed estensione sono casi d'uso completi
    - Devono avere condizioni di entrata ed uscita ed essere comprensibili all'utente in modo indipendente

ASSOCIAZIONE *INCLUDE*: DECOMPOSIZIONE FUNZIONALE

- Problema: Una funzione nella definizione del problema originale è troppo complessa per risolverla immediatamente
- Soluzione: Descrivere la funzione come un'aggregazione di un insieme di funzioni più semplici. Il caso d'uso associato è decomposto in casi d'uso più piccoli



INCLUDE: RIUSO DI FUNZIONALITÀ ESISTENTI

Problema

- Ci sono già funzioni esistenti. Come possiamo riusarle?

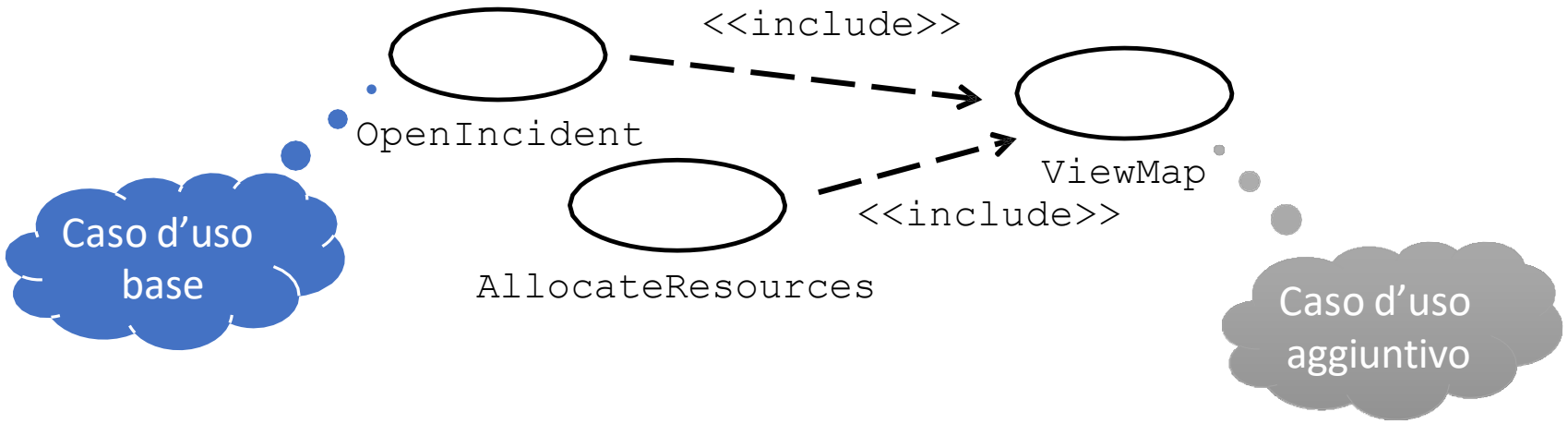
Soluzione

- L'associazione *include* da un caso d'uso A ad un caso d'uso B indica che un'istanza del caso d'uso A esegue tutti i comportamenti descritti nel caso d'uso B ("A delega a B")

RELAZIONE *INCLUDE* TRA CASI D'USO

- In questo modo la ridondanza tra casi d'uso può essere esplicitata usando la relazione di inclusione
  - Il *Dispatcher* deve consultare la mappa della città quando inizia la gestione di un incidente (ad esempio, per valutare quali zone sono a rischio durante un incendio) e quando deve allocare le risorse sul luogo (per verificare quali sono le risorse più vicine)
  - Il caso d'uso *ViewMap* descrive il flusso degli eventi richiesto quando si consulta la mappa
    - Usato sia da *OpenIncident* che da *AllocateResources*

RELAZIONE *INCLUDE* TRA CASI D'USO



Osservazione: il caso base non può esistere da solo. E' sempre invocato insieme al caso d'uso aggiuntivo

RELAZIONE *INCLUDE* TRA CASI D'USO

- Esplicitare comportamenti condivisi dai casi d'uso ha molti benefici
  - **Descrizioni più brevi e meno ridondanze**
- Il comportamento dovrebbe essere esplicitato in un caso d'uso separato solo se è condiviso tra due o più casi d'uso
- Un'eccessiva frammentazione della specifica dei requisiti attraverso un elevato numero di casi d'uso rende confusa la specifica ai clienti ed agli utenti

RELAZIONI *EXTEND* VS RELAZIONI *INCLUDE*

- Entrambe le relazioni sono simili
  - Inizialmente potrebbe non essere chiaro allo sviluppatore quando usarle
- La principale distinzione tra le due relazioni sta nella direzione della relazione
  - Per le relazioni *include*, l'evento che innesca il caso d'uso target (quello incluso) è descritto nel flusso di eventi del caso d'uso sorgente
  - Per le relazioni *extend*, l'evento che innesca il caso d'uso sorgente (quello che estende) è descritto nel caso d'uso sorgente come precondizione
  - In altre parole, per le relazioni *include* , ogni caso d'uso che include deve specificare dove il caso d'uso incluso dovrebbe essere invocato
  - Per le relazioni *extend*, solo il caso d'uso che estende specifica quali casi d'uso sono estesi

ReportEmergency (relazione extend)	ReportEmergency (relazione include)
1. … **2. …** 3. Il *FieldOfficer* completa il form selezionando il livello, il tipo, l'ubicazione dell'emergenza e una breve descrizione della situazione. Il *FieldOfficer* descrive anche le risposte possibili alla situazione di emergenza. Una volta che il form è completo, il *FieldOfficer* sottomette il form, al cui punto, è notificato il *Dispatcher*. ***Se la connessione con il Dispatcher è interrotta, è usato il caso d'uso ConnectionDown.*** 4. Se la connessione è ancora valida, il *Dispatcher* rivede le informazioni sottomesse e crea un *Incident* nel db invocando il caso d'uso *OpenIncident*. Il *Dispatcher* seleziona una risposta e accetta il rapporto di emergenza. ***Se la connessione è interrotta, è usato il caso d'uso ConnectionDown.*** 5. …	**1. …** **2. …** 3. Il *FieldOfficer* completa il form selezionando il livello, il tipo, l'ubicazione dell'emergenza e una breve descrizione della situazione. Il *FieldOfficer* descrive anche le risposte possibili alla situazione di emergenza. Una volta che il form è completo, il *FieldOfficer* sottomette il form, al cui punto, è notificato il *Dispatcher*. 4. Il *Dispatcher* rivede le informazioni sottomesse e crea un *Incident* nel db invocando il caso d'uso *OpenIncident*. Il *Dispatcher* seleziona una risposta e accetta il rapporto di emergenza. 5. …

ConnectionDown (relazione include)	**ConnectionDown (relazione extend)**
1. Il FieldOfficer e il Dispatcher sono notificati dell'interruzione della connessione. Sono avvisati delle possibili ragioni per cui si è verificato l'evento (ad esempio, "la stazione del FieldOfficer è in un tunnel?") 2. La situazione vien trascritta dal sistema e recuperata quando la connessione è ristabilita. 3. Il FieldOfficer e il Dispatcher entrano in contatto in altri modi ed il Dispatcher inizia ReportEmergency dalla stazione del Dispatcher.	*Il caso d'uso ConnectionDown estende qualsiasi caso d'uso in cui la comunicazione tra il FieldOfficer e il Dispatcher può essere persa.* 1. Il FieldOfficer e il Dispatcher sono notificati dell'interruzione della connessione. Sono avvisati delle possibili ragioni per cui si è verificato l'evento (ad esempio, "la stazione del FieldOfficer è in un tunnel?") 2. La situazione vien trascritta dal sistema e recuperata quando la connessione è ristabilita. 3. Il FieldOfficer e il Dispatcher entrano in contatto in altri modi ed il Dispatcher inizia ReportEmergency dalla stazione del Dispatcher.

RELAZIONI *EXTEND* VS RELAZIONI *INCLUDE*

- Nella colonna di sinistra (relazione include), dobbiamo inserire il testo in due punti nel flusso di eventi dove il caso d'uso *ConnectionDown* può essere invocato
  - Inoltre, se sono descritte situazioni eccezionali aggiuntive (un funzione help sulla stazione del ***FieldOfficer***), il caso d'uso ***ReportEmergency*** dovrà essere modificato è sarà ingombrato dalle condizioni

RELAZIONI *EXTEND* VS RELAZIONI *INCLUDE*

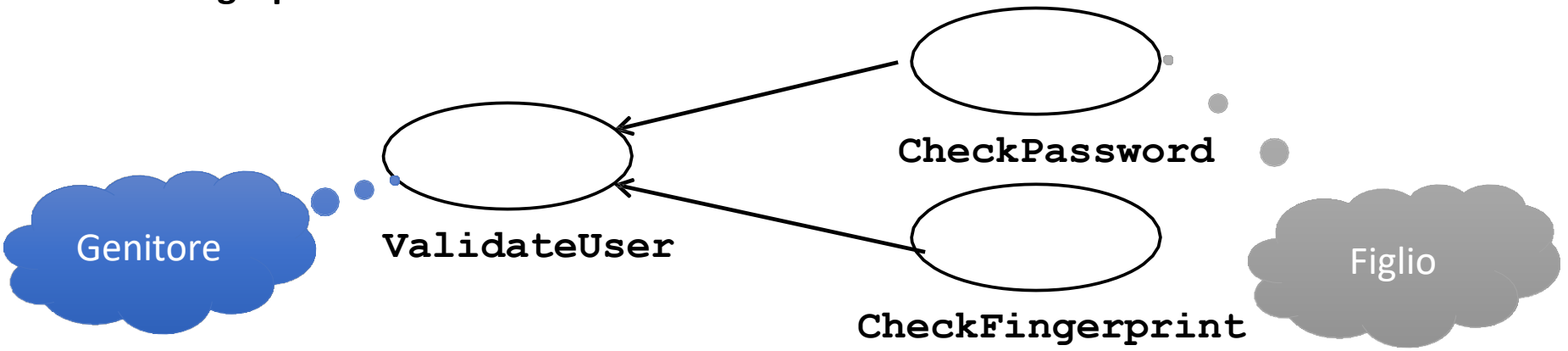
- Nella colonna di destra, dobbiamo descrivere solo le condizioni sotto cui il caso d'uso eccezionale è invocato, il che può includere numerosi casi d'uso
  - Inoltre, situazioni eccezionali aggiuntive possono essere aggiunte senza modificare il caso d'uso base (***EmergencyReport***)
- L'abilità di estendere il sistema senza modificare parti esistenti è critica, poiché consente di lasciare inalterato il comportamento originale
- La distinzione tra *include* e *extend* è una questione di documentazione
  - Usare il tipo giusto di relazione riduce le dipendenze tra i casi d'uso, la ridondanza, e abbassa la probabilità di introdurre errori quando cambiano i requisiti

EURISTICHE PER LE RELAZIONI *EXTEND* E *INCLUDE*

- Usare le relazioni ***extend*** per comportamenti eccezionali, opzionali o che si verificano di rado
  - Un esempio di comportamento raro è il guasto di una risorsa
  - Un esempio di comportamento opzionale è la notifica di risorse nelle vicinanze che rispondono ad un incidente che non le riguarda
- Usare le relazioni ***include*** per un comportamento condiviso tra due o più casi d'uso
- Usare con discrezione le due euristiche precedenti e non sovra strutturare il modello del caso d'uso. Casi d'uso un po' più lunghi (due pagine) sono più semplici da capire e rivedere rispetto a molti casi d'uso brevi (lunghi dieci righe, ad esempio)

ASSOCIAZIONE DI GENERALIZZAZIONE TRA CASI D'USO

- Problema: C'è un comportamento comune ma più specifico tra casi d'uso e vogliamo esplicitarlo
- Soluzione: L'associazione di generalizzazione tra casi d'uso esplicita comportamenti comuni. I casi d'uso figli ereditano il comportamento ed il significato del genitore ed aggiungono altri comportamenti
- Esempio: Consideriamo il caso d'uso **ValidateUser**, responsabile della verifica dell'identità di un utente. Il cliente potrebbe richiedere due realizzazioni: **CheckPassword** e **CheckFingerprint**



IDENTIFICAZIONE DEGLI OGGETTI DI ANALISI INIZIALI

- Uno dei primi ostacoli che sviluppatori ed utenti incontrano quando cominciano a collaborare è la terminologia differente
  - Anche se gli sviluppatori imparano la terminologia degli utenti, il problema si pone ancora quando si aggiungono nuovi sviluppatori al progetto
  - Le incomprensioni si verificano per l'uso di stessi termini usati in contesti differenti con significati diversi
- Per stabilire una terminologia chiara, gli sviluppatori identificano gli oggetti partecipanti per ogni caso d'uso
  - Devono identificare il nome, descriverli in modo non ambiguo e raccoglierli in un glossario
    - La costruzione del glossario costituisce il primo passo verso l'analisi

GLOSSARIO

- Il glossario è incluso nella specifica dei requisiti e successivamente nei manuali utente
- Gli sviluppatori mantengono il glossario aggiornato durante l'evoluzione della specifica dei requisiti
- I benefici del glossario sono diversi
  - I nuovi sviluppatori hanno a disposizione un insieme consistente di definizioni
  - E' usato un termine singolo per ogni concetto
  - Ogni termine ha un preciso e chiaro significato ufficiale

IDENTIFICAZIONE DEGLI OGGETTI DI ANALISI INIZIALI

- L'identificazione degli oggetti partecipanti da luogo al modello ad oggetti di analisi iniziale
  - Questo passo, durante la scoperta dei requisiti, costituisce solo un primo passo verso il modello degli oggetti di analisi completo
  - Il modello di analisi completo solitamente non è usato come mezzo di comunicazione per sviluppatori ed utenti
    - Gli utenti il più delle volte non hanno familiarità con concetti orientati agli oggetti
    - Tuttavia, la descrizione degli oggetti ed i loro attributi sono visibili agli utenti e revisionati

EURISTICHE PER IDENTIFICARE GLI OGGETTI DI ANALISI INIZIALE

- Termini che gli sviluppatori o gli utenti devono chiarire per capire i casi d'uso
- Sostantivi ricorrenti nei casi d'uso (ad esempio, **Incident**)
- Entità del mondo reale di cui il sistema deve tenere traccia (ad esempio, **FieldOfficer** e **Resource**)
- Processi del mondo reale di cui il sistema deve tenere traccia (**EmergencyOperationPlan**)
- Casi d'uso (**ReportEmergency**)
- Sorgenti di dati (**Printer**)
- Artefatti con cui l'utente interagisce (**Station**)
- Usare sempre i termini del dominio dell'applicazione

IDENTIFICAZIONE DEGLI OGGETTI

- Durante la scoperta dei requisiti, sono generati gli oggetti partecipanti per ciascun caso d'uso
  - Se due casi d'uso si riferiscono allo stesso concetto, l'oggetto corrispondente dovrebbe essere lo stesso
  - Se due oggetti condividono lo stesso nome e non corrispondono allo stesso concetto, uno o entrambi i concetti sono rinominati per enfatizzarne la differenza
    - Si eliminano così le ambiguità nella terminologia impiegata

OGGETTI PARTECIPANTI AL CASO D'USO *REPORTEMERGENCY*

Dispatcher	Ufficiale di Polizia che gestisce gli Incident. Un Dispatcher apre, documenta, e chiude gli incidenti in risposta ad un EmergencyReport e altre comunicazioni con i FieldOfficer. I Dispatcher sono identificati con numeri di badge.
EmergencyReport	Rapporto iniziale su un Incident da un FieldOfficer ad un Dispatcher. Un EmergencyReport solitamente innesca la creazione di un Incident dal Dispatcher. Un EmergencyReport è composto da un livello di emergenza, un tipo (fuoco, incidente stradale, altro), una località e una descrizione.
FieldOfficer	Ufficiale di Polizia o dei Vigili del fuoco in servizio. Un FieldOfficer può essere allocato al più ad un incidente alla volta. I FieldOfficer sono identificati dai numeri di badge.
Incident	Situazione che richiede l'attenzione di un FieldOfficer. Il rapporto su un Incident può essere inserito nel sistema da un FieldOfficer o chiunque altro esterno al sistema. Un Incident è composto da una descrizione, una risposta, uno stato (aperto, chiuso, documentato), una località, e un numero di FieldOfficer.

EURISTICHE PER IL CONTROLLO INCROCIATO DI CASI D'USO ED OGGETTI PARTECIPANTI

- Quali casi d'uso creano questo oggetto (cioè, durante quali casi d'uso i valori degli attributi di un oggetto sono immessi nel sistema)?
- Quali attori possono accedere a tali informazioni?
- Quali casi d'uso modificano e distruggono questo oggetto (cioè, quali casi d'uso editano o rimuovono questa informazione dal sistema)?
- Quale attore può iniziare questi casi d'uso?
- E' necessario questo oggetto (cioè, c'è almeno un caso d'uso che dipende da questa informazione)?

IDENTIFICAZIONE DEI REQUISITI NON FUNZIONALI

I requisiti non funzionali possono avere un impatto importante sul lavoro dell'utente in modo inaspettato

Per scoprire in modo accurato tutti i requisiti non funzionali essenziali, ambo i clienti e sviluppatori devono collaborare in modo da identificare quali attributi del sistema (minimali) difficili da realizzare sono critici per il lavoro dell'utente

L'insieme risultante di requisiti non funzionali tipicamente include requisiti in conflitto tra loro

EURISTICHE PER I REQUISITI NON FUNZIONALI

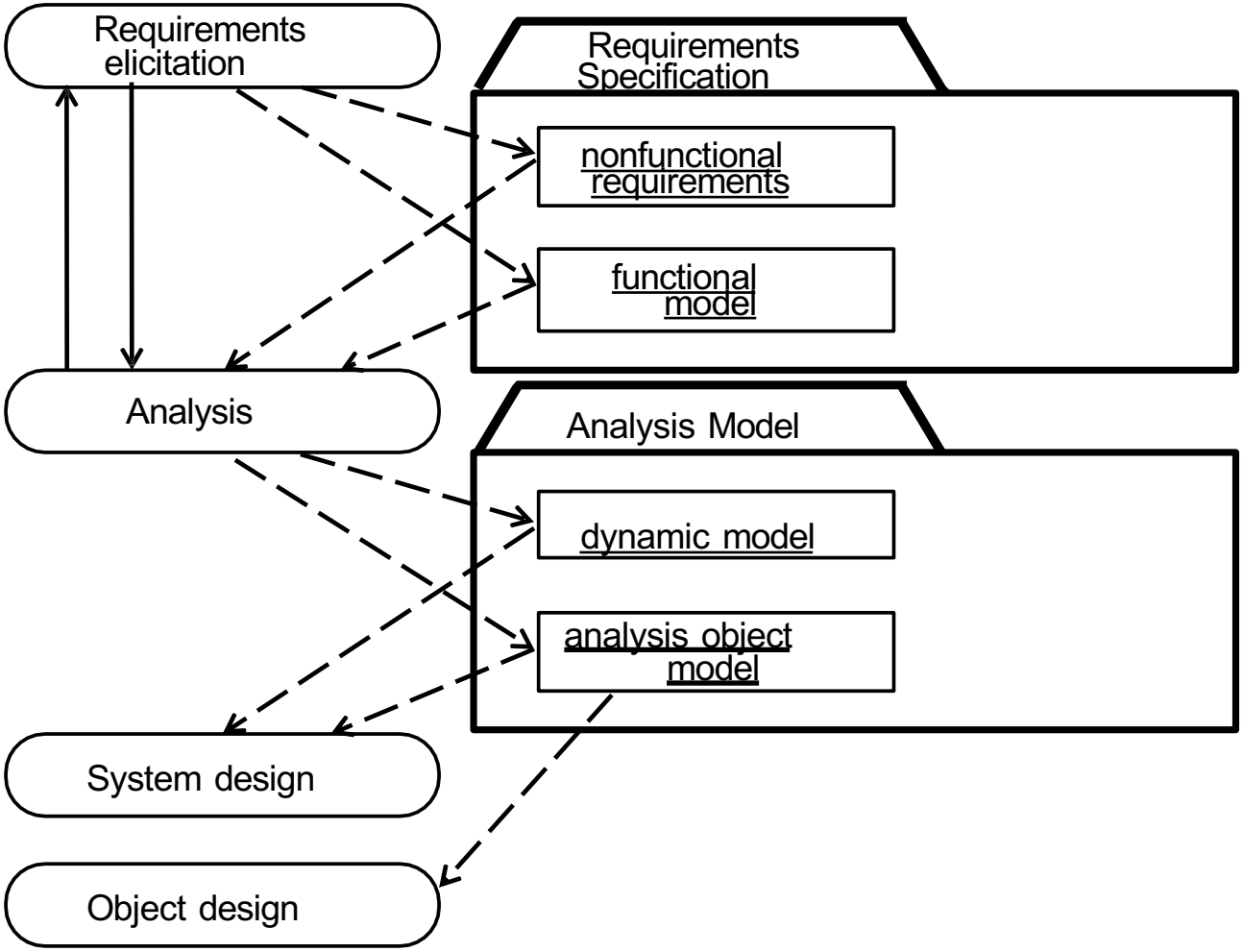
Ci sono pochi metodi sistematici per scoprire i requisiti non funzionali

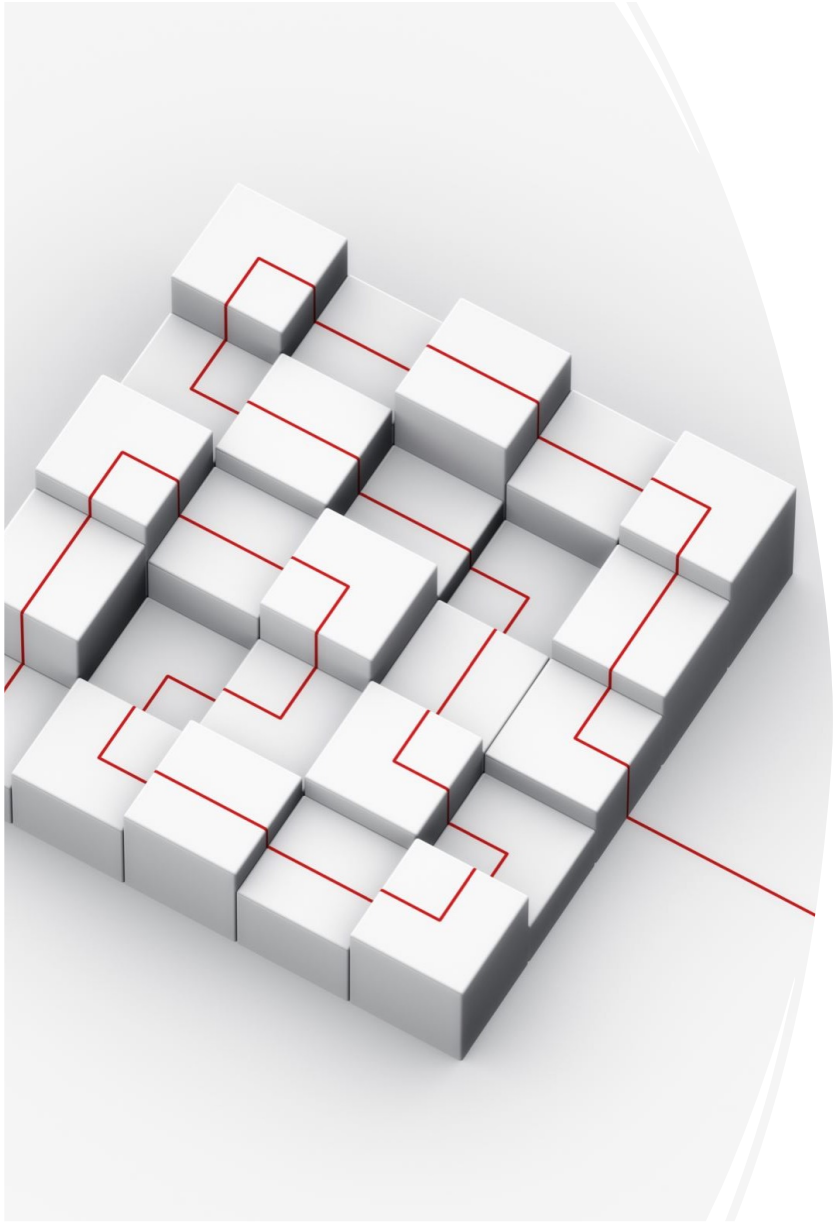
Gli analisti usano una tassonomia (schema FURPS+) dei requisiti non funzionali per generare una lista di controllo di domande per aiutare clienti e sviluppatori a focalizzarsi sugli aspetti non funzionali del sistema

Poiché gli attori in questa fase sono già stati identificati, la lista di controllo può essere organizzata per ruolo e distribuita agli utenti rappresentativi

ANALISI, MODELLAZIONE DEGLI OGGETTI

ANALISI E SUOI PRODOTTI

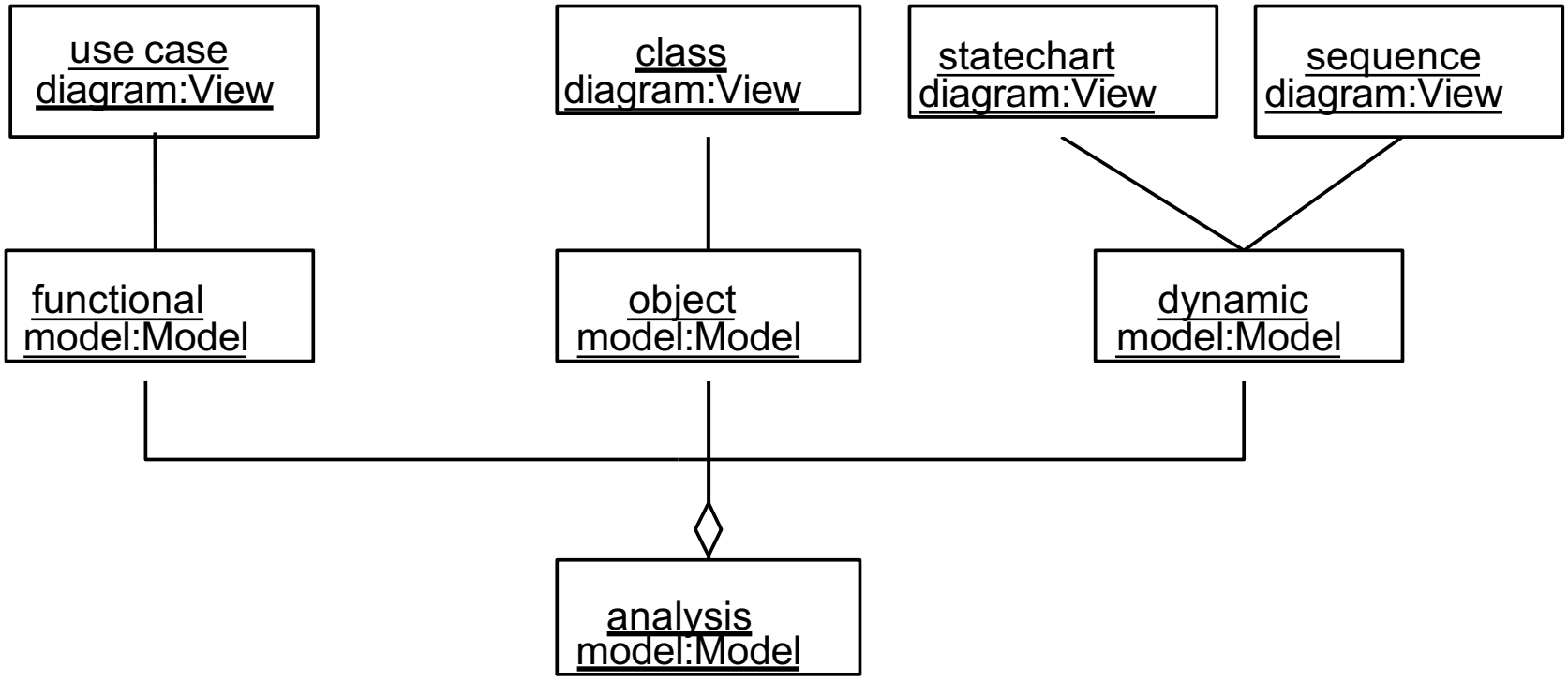




MODELLO DI ANALISI

- Il modello di analisi è composto da tre singoli modelli
  - Il **modello funzionale**, rappresentato da casi d'uso e scenari
  - Il **modello ad oggetti di analisi**, rappresentato da diagrammi delle classi e degli oggetti
  - Il **modello dinamico**, rappresentato dai diagrammi di stato e delle sequenze

IL MODELLO DI ANALISI



ATTIVITÀ DI ANALISI DEI REQUISITI

- **Modellazione degli oggetti**
  - Attività durante la modellazione degli oggetti
  - Identificazione degli oggetti
  - Tipi di oggetti
    - Entità (**Entity**), confine (**Boundary**), controllo (**Control**)
  - Stereotipi
  - Tecnica di Abbott
    - Aiuta nell'identificazione degli oggetti

ATTIVITÀ DURANTE LA MODELLAZIONE DEGLI OGGETTI

Obiettivo principale: trovare le astrazioni importanti

- Passi
  - Identificazione delle classi
- Basata sull'assunto essenziale che possiamo trovare le astrazioni
  - Trovare gli attributi
  - Trovare i metodi
  - Trovare le associazioni tra le classi
- Ordine dei passi
- Obiettivo: determinare le astrazioni desiderate
- L'ordine dei passi è secondario, solo un'euristica
- Cosa accade se troviamo le astrazioni errate?
- Iteriamo e revisioniamo il modello (ricordate la falsificabilità ?)

IDENTIFICAZIONE DELLE CLASSI

- Cruciale
  - Aiuta ad identificare le entità importanti del sistema
- Assunzioni di base
  - Possiamo trovare le classi in un nuovo sistema software (Forward engineering)
  - Possiamo identificare le classi in un sistema esistente (Reverse engineering)
- Come possiamo procedere?

IDENTIFICAZIONE DELLE CLASSI

- Approcci
  - Approccio del dominio applicativo
    - Chiedere agli esperti di identificare le astrazioni più rilevanti
  - Approccio sintattico
    - Iniziare con i casi d'uso
    - Analizzare il testo per identificare gli oggetti
    - Estrarre gli oggetti partecipanti dal flusso di eventi
  - Approccio con i design pattern
    - Usare design pattern riusabili
  - Approccio basato su componenti
    - Identificare classi di soluzioni esistenti

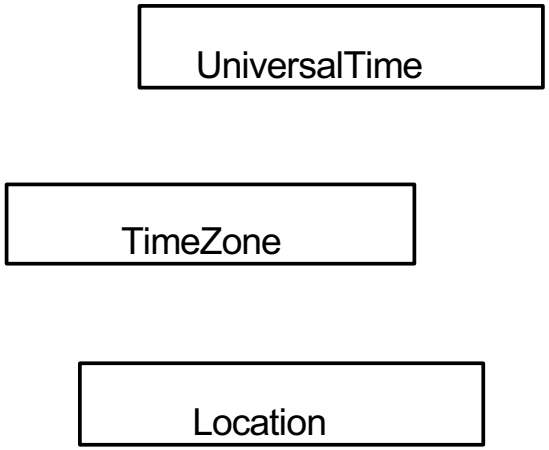
L'IDENTIFICAZIONE OGGETTI È UN PROBLEMA DIFFICILE

- Un problema: definizione del contorno (o confine) del sistema
  - Quali astrazioni sono all'esterno, quali all'interno?
    - Gli attori sono all'esterno
    - Classi/oggetti sono all'interno
- Un altro problema: le classi/oggetti non si trovano considerando solo una prospettiva del dominio o di una scena
  - Il dominio applicativo deve essere analizzato da diverse prospettive
  - A seconda dello scopo del sistema possono essere individuati diversi oggetti
    - Come possiamo identificare lo scopo del sistema?
    - Scenari e casi d'uso -> modello funzionale

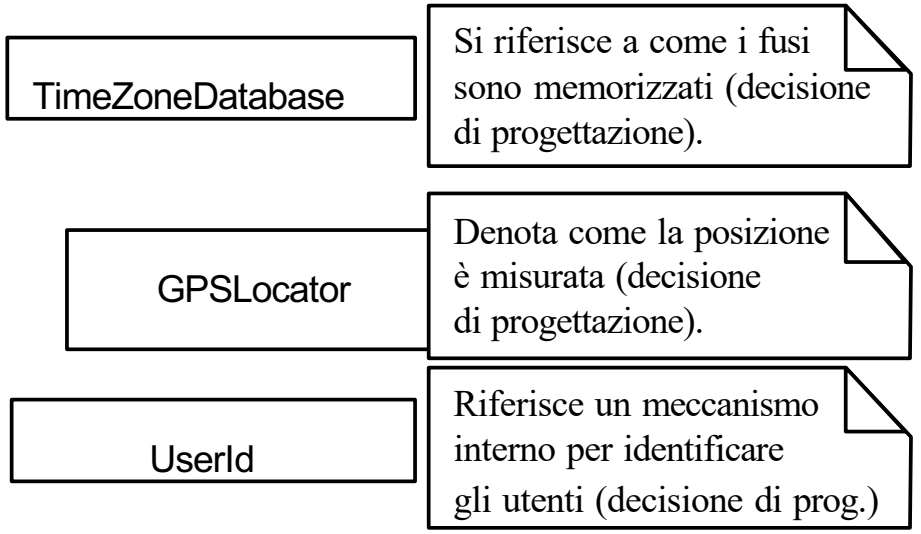
ESEMPI E CONTRO ESEMPI DI CLASSI NEL MODELLO AD OGGETTI DI ANALISI DI *SATWATCH*

- Il modello ad oggetti di analisi rappresentano concetti a livello utente e non le classi o componenti effettivi del software

Concetti del dominio che dovrebbero essere rappresentati nel modello ad oggetti di analisi



Classi software che non dovrebbero essere rappresentate nel modello ad oggetti di analisi



TIPI DIVERSI DI OGGETTI

- Oggetti **Entity**
  - Rappresentano le informazioni persistenti mantenute dal sistema (oggetti del dominio applicativo)
- Oggetti **Boundary**
  - Rappresentano l'interazione tra l'utente ed il sistema
- Oggetti **Control**
  - Rappresentano i compiti di controllo eseguiti dal sistema

ESEMPIO: MODELLAZIONE 2BWATCH

Per distinguere i diversi tipi di oggetti in un modello possiamo usare il meccanismo UML dello Stereotipo

Oggetti Entity | Oggetti Control | Oggetti Boundary

TIPI DI OGGETTI

- Le tre categorie di oggetti consentono ai modelli di essere più resilienti alle modifiche
  - L'interfaccia di un sistema cambia con maggiore probabilità rispetto al controllo
  - Il modo in cui il sistema è controllato cambia con maggiore possibilità rispetto alle entità di un dominio applicativo
- I tipi degli oggetti sono stati introdotti in Smalltalk
  - Model, View, Controller (MVC)
    - Model <-> Oggetto Entity
    - View <-> Oggetto Boundary
    - Controller <-> Oggetto Control

TROVARE GLI OGGETTI PARTECIPANTI NEI CASI D'USO

- Prendere un **caso d'uso** e leggere il flusso degli eventi
- Fare un'analisi testuale (analisi sostantivo-verbo)
  - I sostantivi sono candidati per gli oggetti/classi
  - I verbi sono candidati per le operazioni
  - Questa procedura è nota come tecnica di Abbott
- Dopo che gli oggetti/classi sono stati trovati, identificare i tipi
  - Identificare le entità del mondo reale di cui il sistema deve tener traccia (FieldOfficer-> Oggetto Entity)
  - Identificare procedure del mondo reale di cui il sistema deve tenere traccia (EmergencyPlan -> Oggetto Control)
  - Identificare artefatti di interfaccia (PoliceStation-> oggetto Boundary)

ESEMPIO

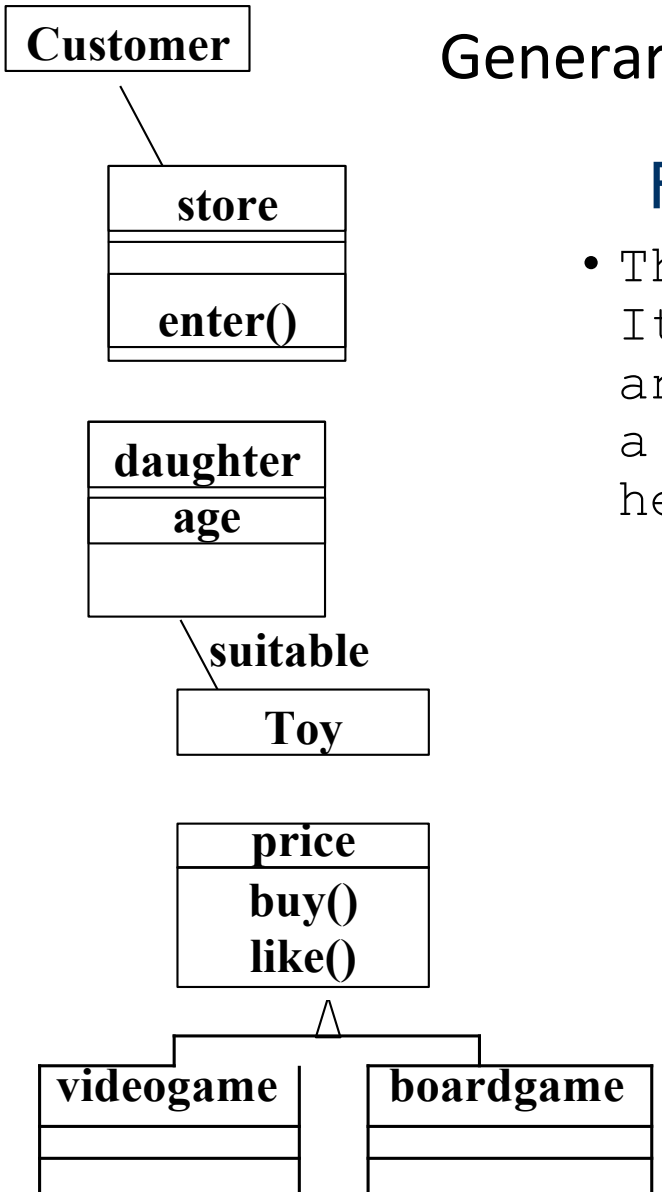
Flusso di eventi:

- 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 5 years old.
- The assistant recommends another type of toy, namely the boardgame “Monopoly".

MAPPING PARTI DEL PARLATO AI COMPONENTI DEL MODELLO (TECNICA DI ABBOTT)

Example	*Part of speech*	*UML model component*
"Monopoly"	Proper noun	object
Toy	Improper noun	class
Buy, recommend	Doing verb	operation
is-a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
dangerous	adjective	attribute
enter	transitive verb	operation
depends on	intransitive verb	Constraint, class, association

Generare il diagramma delle classi dal flusso di eventi



Flusso di eventi:

- 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 5 years old. The assistant recommends another type of toy, namely a boardgame The customer buy the game and leaves the store

MODI PER TROVARE OGGETTI

- Investigazione sintattica con la tecnica di Abbott
  - Flusso di eventi nei casi d'uso
  - Descrizione del problema
- Usare altre sorgenti di conoscenza
  - Conoscenza dell'applicazione: gli utenti e gli esperti conoscono le astrazioni del dominio applicativo
  - Conoscenza della soluzione: astrazioni nel dominio della soluzione
  - Conoscenza generale del mondo: la vostra conoscenza e intuizione

ORDINE DELLE ATTIVITÀ PER L'IDENTIFICAZIONE DEGLI OGGETTI

1. Formulare pochi scenari con l'aiuto di un utente o esperto del dominio applicativo
2. Estrarre i casi d'uso dagli scenari, con l'aiuto di un esperto del dominio applicativo
3. Poi procedere in parallelo con:
   - Analizzare il flusso di eventi in ogni caso d'uso usando la tecnica di analisi testuale di Abbott
   - Generare il diagramma delle classi

PASSI NELLA GENERAZIONE DEL DIAGRAMMA DELLE CLASSI

- Identificazione della classe (analisi testuale, esperto del dominio)
- Identificazione di attributi e operazioni (talvolta prima che le classi siano trovate)
- Identificazione di associazioni tra classi
- Identificazione delle molteplicità
- Identificazione dei ruoli
- Identificazione dell'ereditarietà

ESEMPIO

- Dopo un primo esame del caso d'uso **ReportEmergency** si impiegano la conoscenza del dominio applicativo e le interviste con gli utenti per identificare gli oggetti **Dispatcher**, **EmergencyReport**, **FieldOfficer** e **Incident**
  - E' da osservare che l'oggetto EmergencyReport non è menzionato esplicitamente nel caso d'uso ReportEmergency
    - Il passo 5 del caso d'uso si riferisce ai rapporti di emergenza come "informazioni sottomesse dal FieldOfficer".
    - Dopo una revisione con il cliente si scopre che a questa informazione ci si riferisce solitamente come "emergency report" e si decide quindi di chiamare il corrispondente oggetto EmergencyReport

Nome	ReportEmergency
Attori Partecipanti	Iniziato dal FieldOfficer Comunica con il Dispatcher
Flusso eventi:	1. Il FieldOfficer attiva la funzione “Report Emergency” del terminale 2. 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.* 3. Il FieldOfficer riempie la form *specificando almeno 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 4. FRIEND riceve la form e notifica al Dispatcher *con un finestra pop-up.* 5. 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 in 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.* 6. FRIEND visualizza l’accettazione e la risposta selezionata al FieldOfficer
Condizioni di uscita	Il FIeldOfficer riceve l’accettazione e la risposta selezionata

Oggetti partecipanti al caso d'uso *ReportEmergency*

Dispatcher	Ufficiale di Polizia che gestisce gli incidenti. Un Dispatcher apre, documenta, e chiude gli incidenti in risposta ad un EmergencyReport e altre comunicazioni con i FieldOfficer. I Dispatcher sono identificati con numeri di badge.
EmergencyReport	Rapporto iniziale su un Incident da un FieldOfficer ad un Dispatcher. Un EmergencyReport solitamente innesca la creazione di un Incident dal Dispatcher. Un EmergencyReport è composto da un livello di emergenza, un tipo (fuoco, incidente stradale, altro), una località e una descrizione.
FieldOfficer	Ufficiale di Polizia o dei Vigili del fuoco in servizio. Un FieldOfficer può essere allocato al più ad un incidente alla volta. I FieldOfficer sono identificati dai numeri di badge.
Incident	Situazione che richiede l'attenzione di un FieldOfficer. Il rapporto su un Incident può essere inserito nel sistema da un FieldOfficer o chiunque altro esterno al sistema. Un Incident è composto da una descrizione, una risposta, uno stato (aperto, chiuso, documentato), una località, e un numero di FieldOfficer.

IDENTIFICARE GLI OGGETTI BOUNDARY

- Gli oggetti boundary rappresentano l'interfaccia del sistema con gli attori
- In ciascun caso d'uso, ogni attore interagisce con almeno un oggetto boundary
- L'oggetto boundary raccoglie le informazioni dagli attori e le traducono in una forma che è possibile usare con ambo gli oggetti entity e control

EURISTICHE PER IDENTIFICARE GLI OGGETTI BOUNDARY

- Identificare gli elementi dell'interfaccia utente di cui l'utente necessita per iniziare il caso d'uso (es., ReportEmergencyButton)
- Identificare le form con le quali l'utente immette dati nel sistema (es., EmergencyReportForm)
- Identificare avvisi e messaggi che il sistema usa per rispondere all'utente (AcknowledgmentNotice)
- Quando multipli attori sono coinvolti in un caso d'uso, identificare i terminali degli attori (es., DispatcherStation) per riferirsi all'interfaccia utente in questione
- Non modellare aspetti visuali dell'interfaccia con oggetti boundary
- Usare sempre i termini dell'utente finale per descrivere le interfacce; non usare termini dai domini implementativi

Oggetti boundary del caso d'uso *ReportEmergency*

AcknowledgmentNotice	Avviso usato per visualizzare l'accettazione del Dispatcher al FieldOfficer.
DispatcherStation	Computer usato dal Dispatcher.
ReportEmergencyButton	Pulsante usato dal FieldOfficer per iniziare il caso d'uso ReportEmergency
EmergencyReportForm	Form usata per inserire il ReportEmergency. Questa form è presentata al FieldOfficer sulla FIeldOfficerStation quando è selezionata la funzione "Report Emergency". La EmergencyReportForm contiene i campi per specificare tutti gli attributi di un rapporto di emergenza e un pulsante (o altro controllo) per sottomettere la form completa
FieldOfficerStation	Computer portatile usato dal FieldOfficer
IncidentForm	Form usata per la creazione di Incidenti. Questa form è presentata al Dispatcher sul DispatcherStation quando l'EmergencyReport è ricevuto. Il Dispatcher usa anche questa form per allocare le risorse a per accettare il rapporto del FieldOfficer

OGGETTI BOUNDARY DEL CASO D'USO *REPORTEMERGENCY*

- *IncidentForm* non è menzionato esplicitamente nel caso d'uso *ReportEmergency*
  - L'oggetto è stato identificato osservando che il Dispatcher ha bisogno di una interfaccia per vedere il rapporto di emergenza sottomesso dal FieldOfficer e per spedire un'accettazione
- I termini usati per descrivere gli oggetti boundary nel modello di analisi dovrebbero seguire la terminologia dell'utente

IDENTIFICARE GLI OGGETTI CONTROL

- Gli oggetti control sono responsabili per coordinare gli oggetti boundary ed entity
  - Solitamente non hanno una controparte concreta nel mondo reale
  - Spesso esiste una relazione stretta tra un caso d'uso e un oggetto control
    - Un oggetto control solitamente è creato all'inizio di un caso d'uso e cessa di esistere alla terminazione dello stesso
  - E' responsabile di raccogliere informazioni dagli oggetti boundary agli oggetti entity

IDENTIFICARE GLI OGGETTI CONTROL

- Inizialmente, si modella il flusso di controllo del caso d'uso *ReportEmergency* con un oggetto control per ogni attore
  - ***ReportEmergencyControl*** per il *FieldOfficer* e ***ManageEmergencyControl*** per il *Dispatcher*
- La decisione di modellare il flusso di controllo del caso d'uso *ReportEmergency* con due oggetti control scaturisce sapendo che *FieldOfficerStation* e *DispatcherStation* sono due sottosistemi che comunicano su un link asincrono
  - Rendendo questo concetto visibile nel modello di analisi consente di mettere a fuoco comportamenti eccezionali come la perdita di comunicazione tra ambo le stazioni

Oggetti control del caso d'uso *ReportEmergency*

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

EURISTICHE PER IDENTIFICARE GLI OGGETTI CONTROL

- Identificare un oggetto control per caso d'uso
- Identificare un oggetto control per attore nel caso d'uso
- La durata di vita di un oggetto control dovrebbe coprire l'estensione del caso d'uso o l'estensione di una sessione utente. E' difficile identificare l'inizio e la fine dell'attivazione di un oggetto control, il caso d'uso corrispondente probabilmente non ha condizioni di entrata e di uscita ben definite

IDENTIFICARE GLI OGGETTI CONTROL

- Nel modellare il caso d'uso ReportEmergency, è stata modellata la stessa funzionalità usando gli oggetti entity, boundary e control
- Spostandosi dalla prospettiva del flusso di eventi ad una prospettiva strutturale, si è incrementato il livello di dettaglio della descrizione e selezionato termini standard per riferirsi alle entità principali del dominio applicativo e al sistema

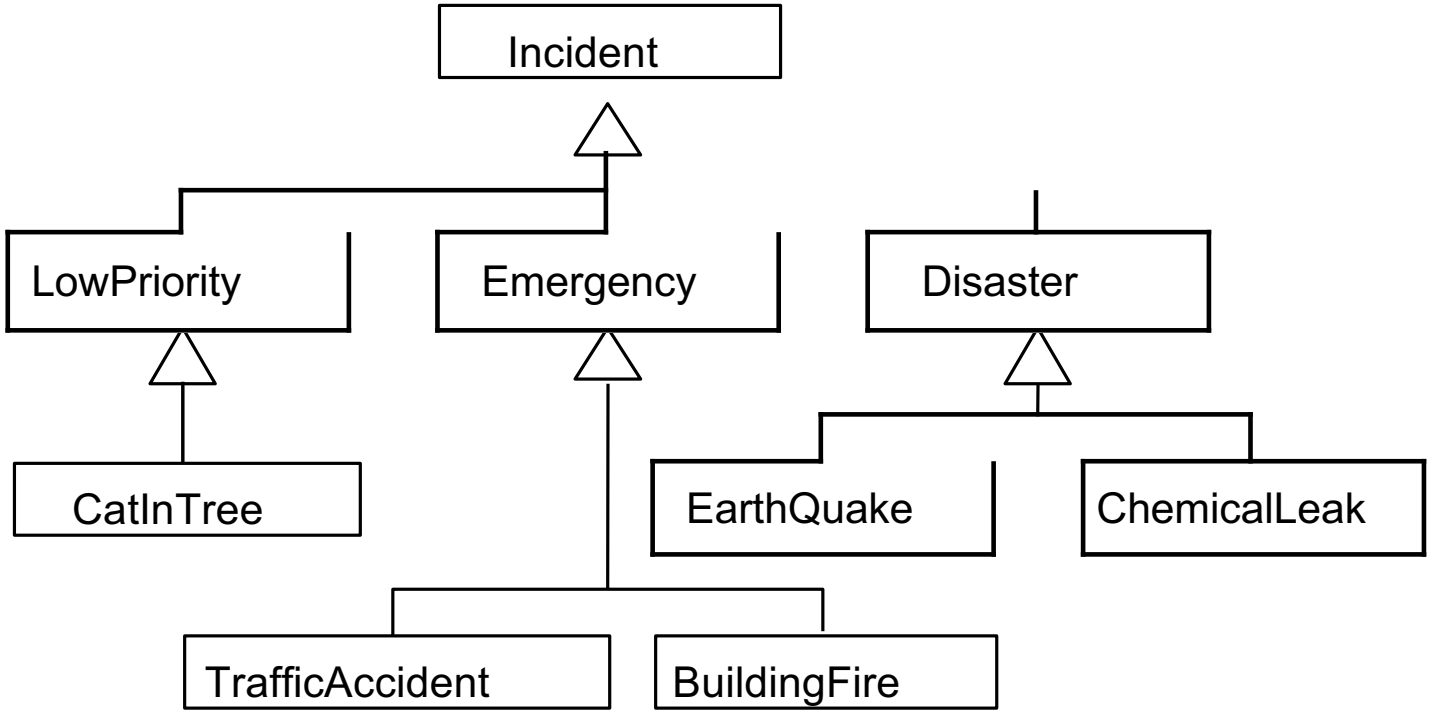
GENERALIZZAZIONE E SPECIALIZZAZIONE

- L'ereditarietà consente di organizzare i concetti in gerarchie
  - In cima alla gerarchia c'è il concetto generale
  - In fondo alla gerarchia troviamo i concetti più specializzati
  - Nel mezzo possono esserci diversi livelli intermedi che rappresentano concetti più o meno specializzati
- Le gerarchie consentono di riferirsi a molti concetti in modo preciso. Ad esempio, in FRIEND
  - Quando ci riferiamo ad un *Incident* intendiamo tutte le istanze del tipo *Incident*
  - Quando ci riferiamo a *Emergency* ci riferiamo solo ad un incidente che richiede una risposta immediata

GENERALIZZAZIONE E SPECIALIZZAZIONE

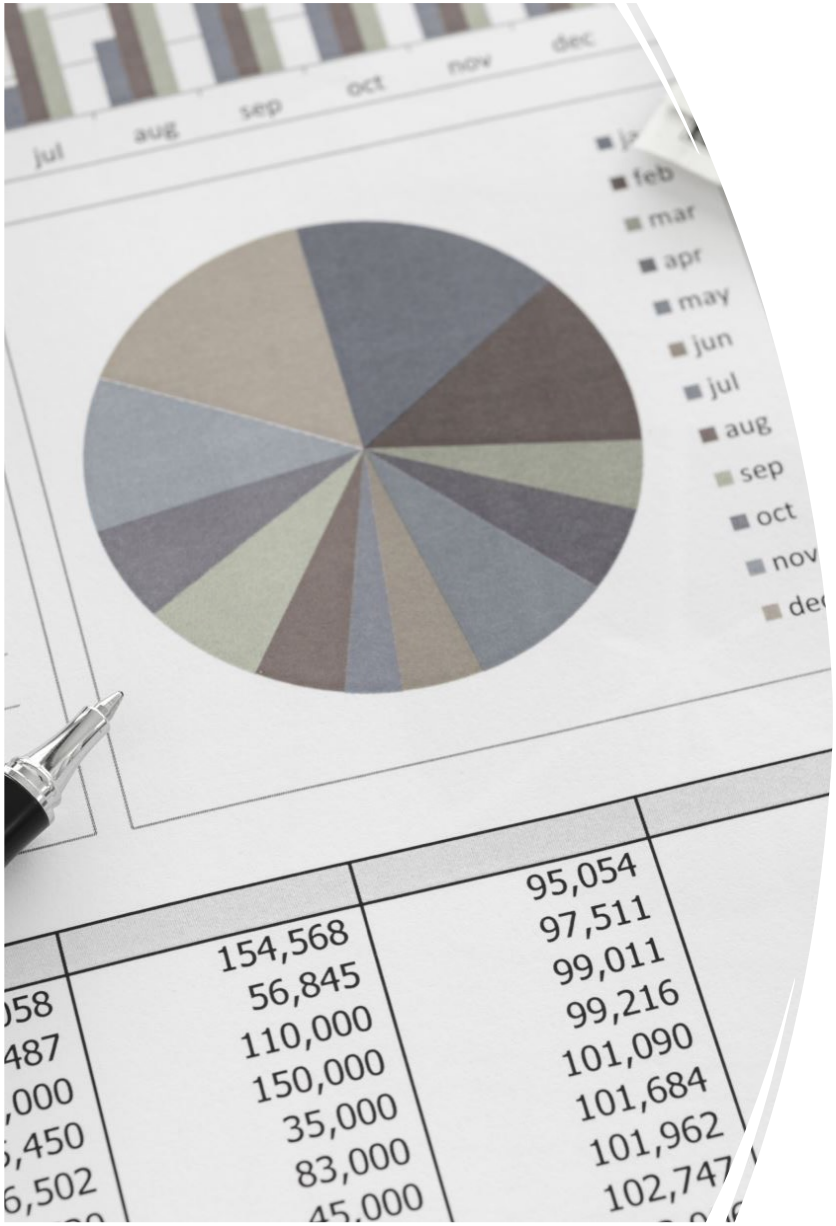
- La specializzazione è l'attività che identifica concetti più specializzati da quelli ad alto livello
  - Esempio: stiamo costruendo un Sistema di gestione emergenze cominciando da zero e stiamo discutendo le funzionalità con il cliente
    - Il cliente ci introduce il concetto di incidente e poi descrive tre tipi di incidenti (*Incidents*)
      - Disastri (*Disasters*), che richiede la collaborazione di diverse agenzie
      - Emergenze (*Emergencies*), che richiede una risposta immediata ma può essere gestita da una singola agenzia
      - Incidenti con priorità bassa (*LowPriorityIncidents*), che non deve essere necessariamente gestita se le risorse sono richieste da altri incidenti con maggiore priorità

ESEMPIO DI GERARCHIA DI GENERALIZZAZIONE



CHI USA I DIAGRAMMI DELLE CLASSI?

- Scopo del diagramma delle classi
  - La descrizione delle proprietà statiche di un sistema
- I principali utenti dei diagrammi delle classi
  - Gli esperti del dominio applicativo
    - Usano i diagrammi delle classi per modellare il dominio applicativo (incluse le tassonomie)
      - Durante la scoperta e l'analisi dei requisiti
  - Lo sviluppatore
    - Usa i diagrammi delle classi durante lo sviluppo di un sistema
      - Durante l'analisi, la progettazione del sistema, la progettazione degli oggetti e l'implementazione



CHI NON USA I DIAGRAMMI DELLE CLASSI?

- Il cliente e l'utente solitamente non sono interessati ai diagrammi delle classi
  - I clienti si focalizzano su problematiche relative alla gestione del progetto
  - Gli utenti sono più interessati nelle funzionalità del sistema

RIEPILOGO

Modellazione del sistema

- Modellazione funzionale + modellazione oggetti + modellazione dinamica

Modellazione funzionale

- Da scenari ai casi d'uso agli oggetti

Modellazione oggetti è attività centrale

- Identificazione classi è l'attività principale della modellazione degli oggetti
- Facili regole sintattiche per trovare classi e oggetti
- Tecnica di Abbott

I diagrammi delle classi sono il centro dell'universo per lo sviluppatore orientato agli oggetti

- L'utente si focalizza più sul modello funzionale e l'usabilità

ANALISI, MODELLAZIONE DINAMICA

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)
- Oggi 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 CON 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

- Definizione di 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

- Un diagramma delle sequenze è una descrizione grafica degli oggetti che partecipano in un caso d'uso usando la notazione DAG
- 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 DIAGRAMMI DI 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

DIAGRAMMI DI SEQUENZE PER *REPORTEMERGENCY*

- 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 *ReportEmergency*

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*

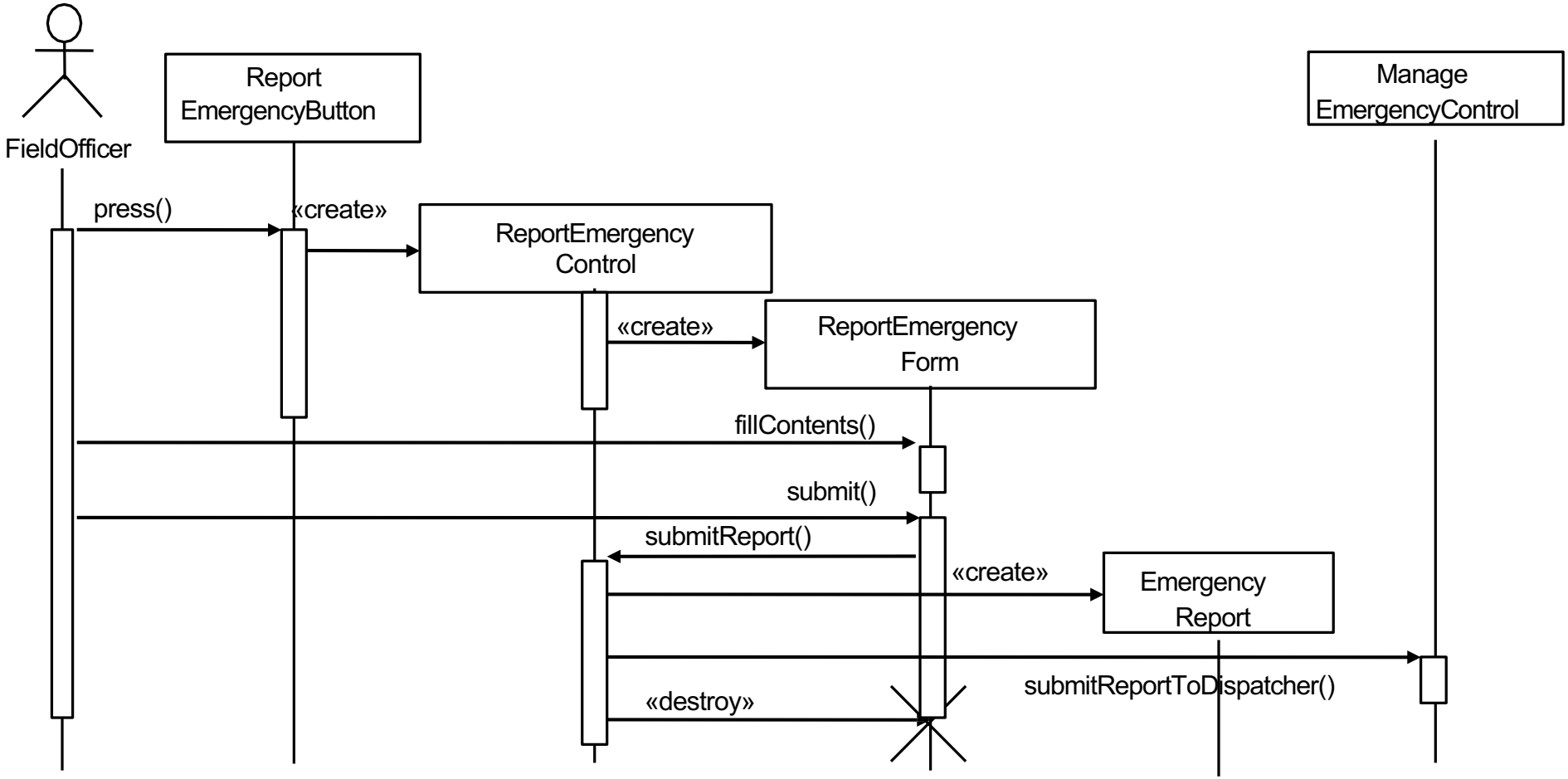


DIAGRAMMA DELLE SEQUENZE PER *REPORTEMERGENCY*

- 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

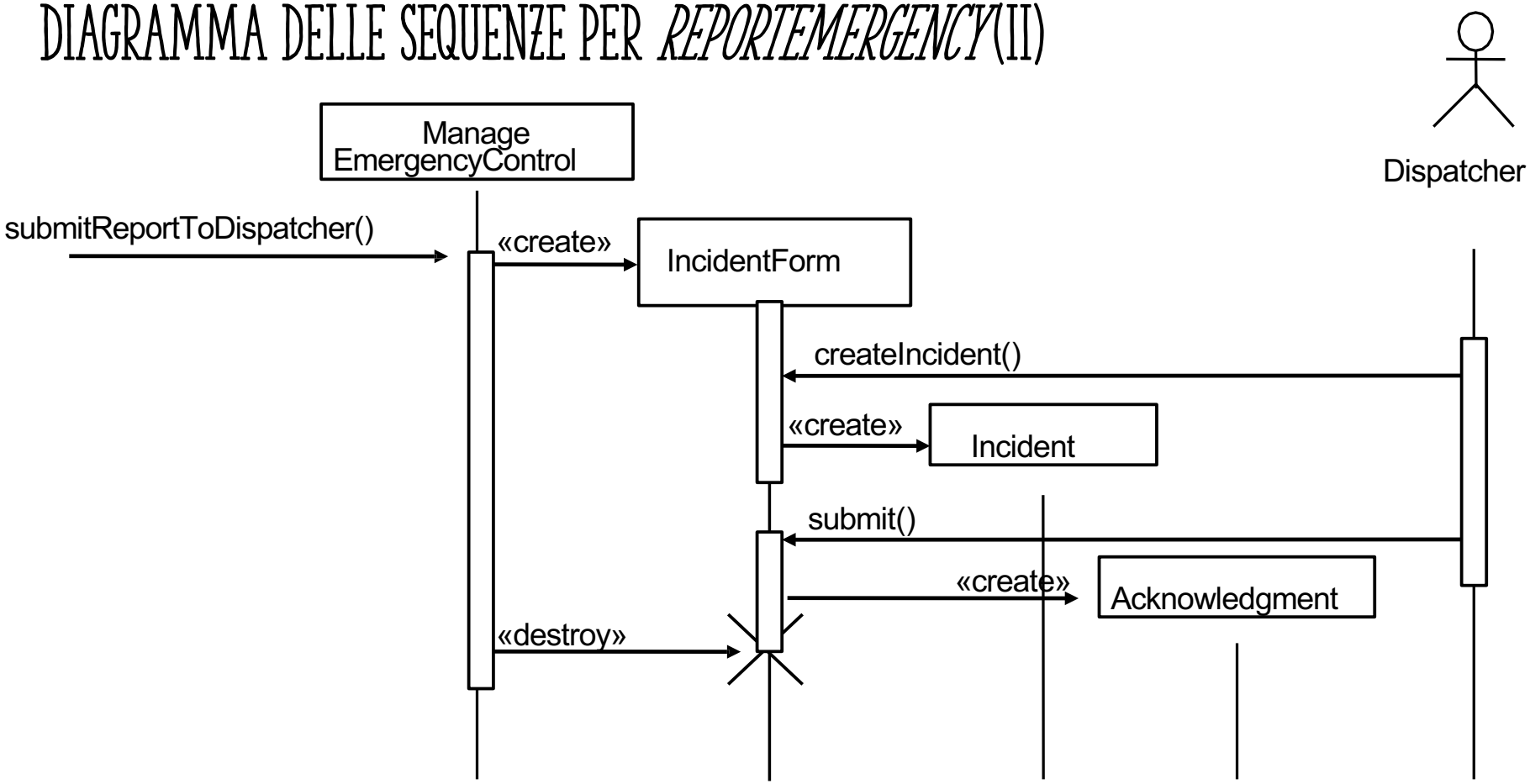
DIAGRAMMA DELLE SEQUENZE PER *REPORTEMERGENCY*

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 *ACKNOWLEDGMENT* PER *REPORTEMERGENCY*

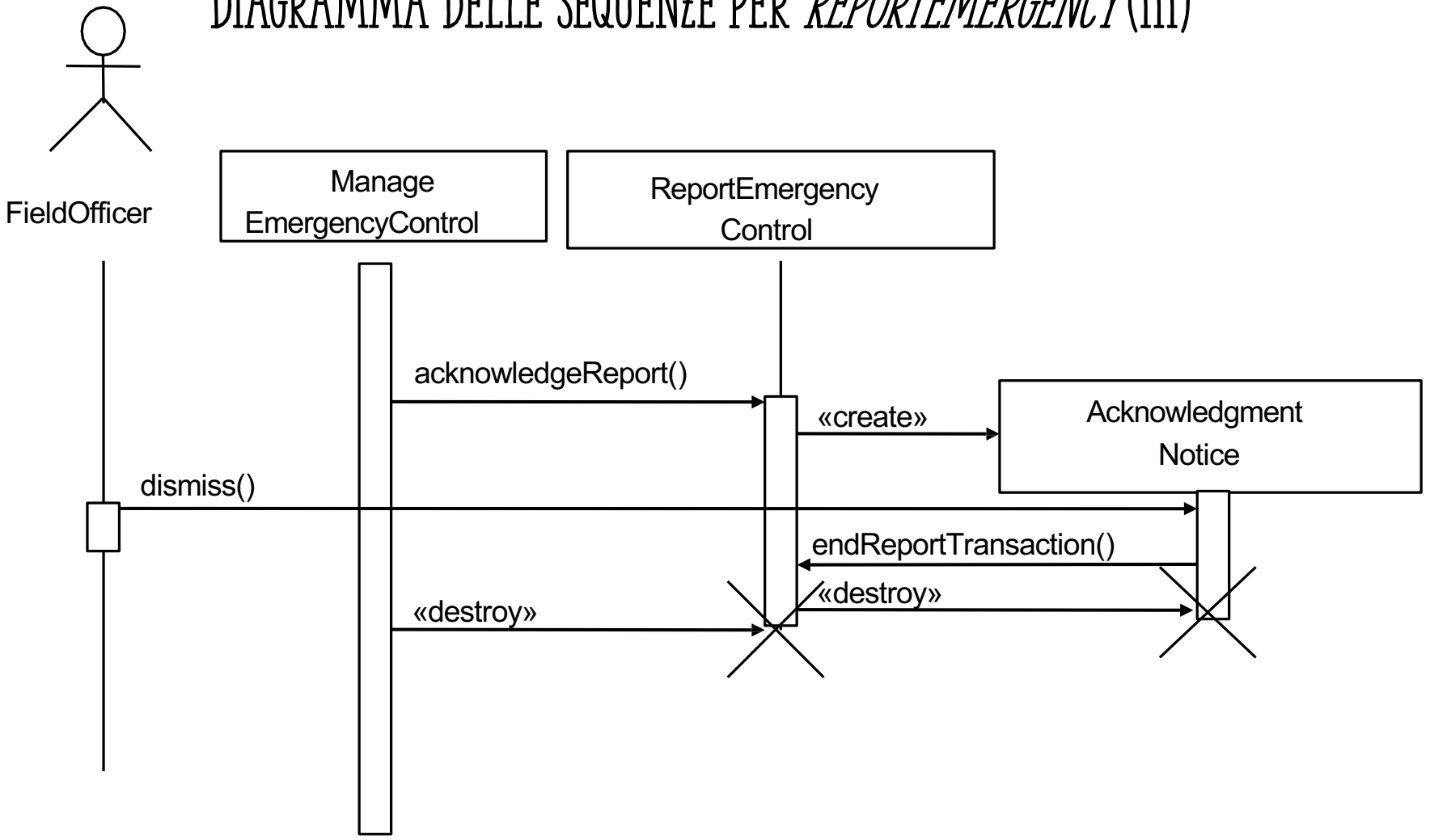
Quando si descrive l'oggetto Acknowledgment ci si rende conto che il caso d'uso originale ReportEmergency è incompleto

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 *Acknowledgment* 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. 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. 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

DIAGRAMMI DI SEQUENZE: DISTRIBUZIONE DEL COMPORTAMENTO

- 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

DIAGRAMMI 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

DIAGRAMMI 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*

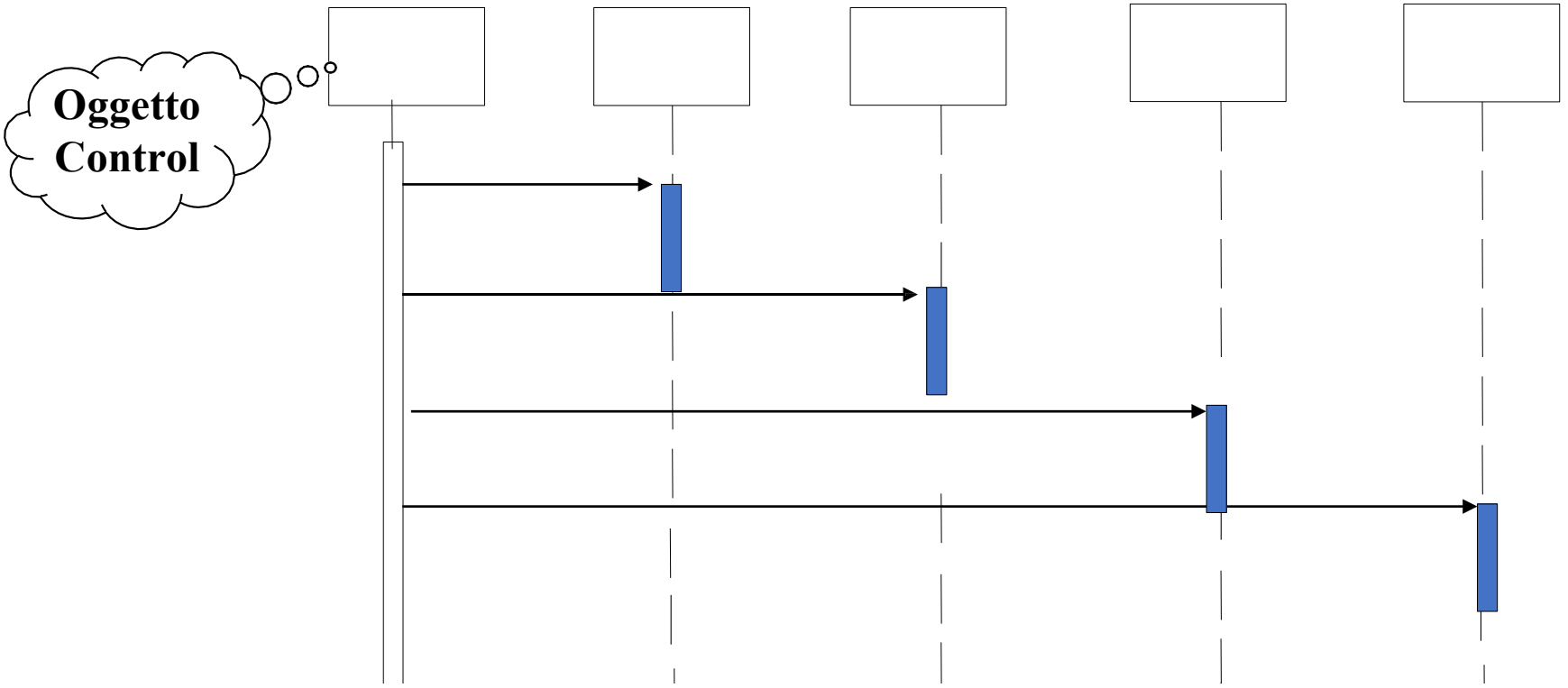
COSA POSSIAMO AGGIUNGERE SUI DIAGRAMMI DELLE SEQUENZE?

- 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)

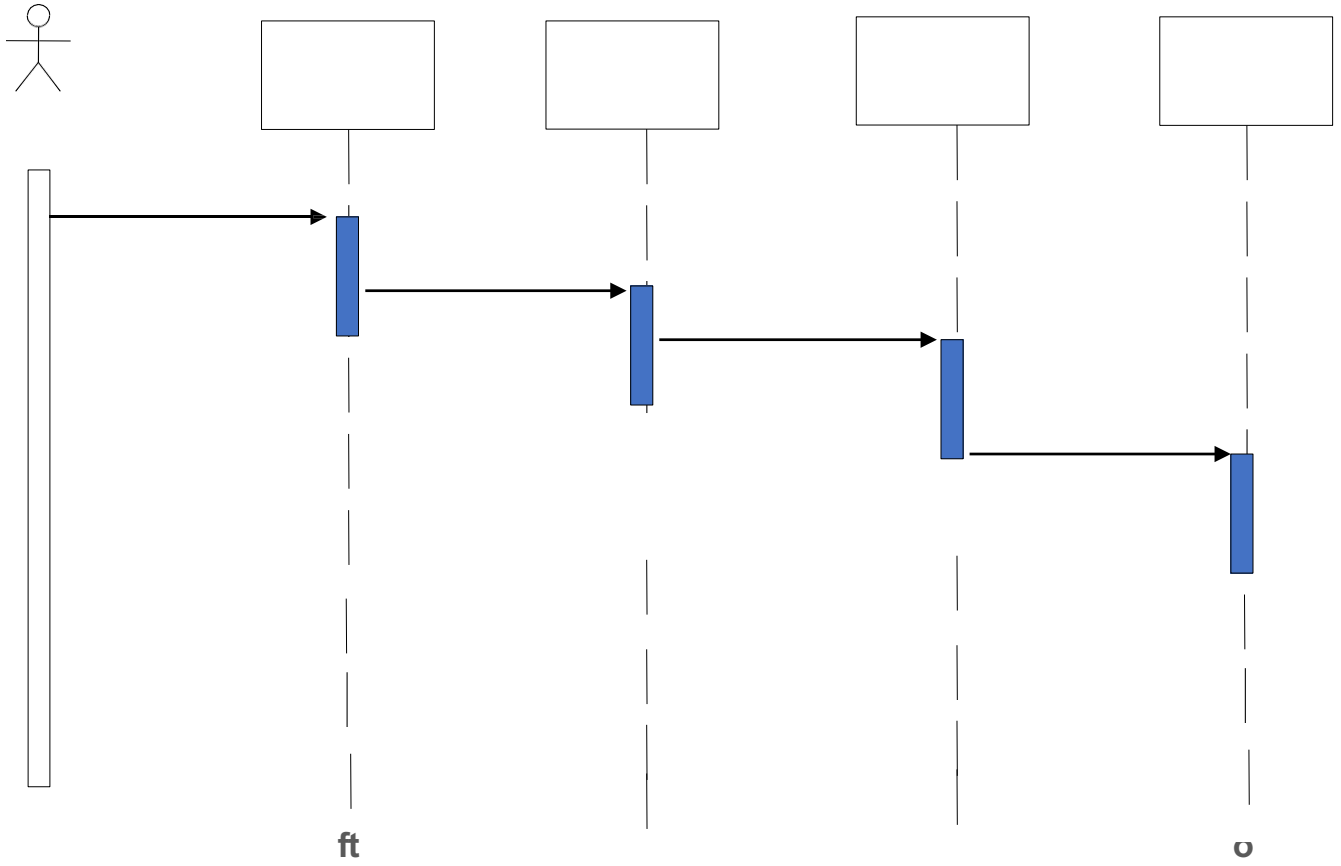
FORK DIAGRAM

- 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
  - Ogi 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 trasformzioni complicate (cioé computazioni che consistono di molti passi)

Quale modello è dominante in queste applicazioni?

- Compilatore
- Database
- Spreadsheet

ESEMPI 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
  - E' vero anche per gli IDE?

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

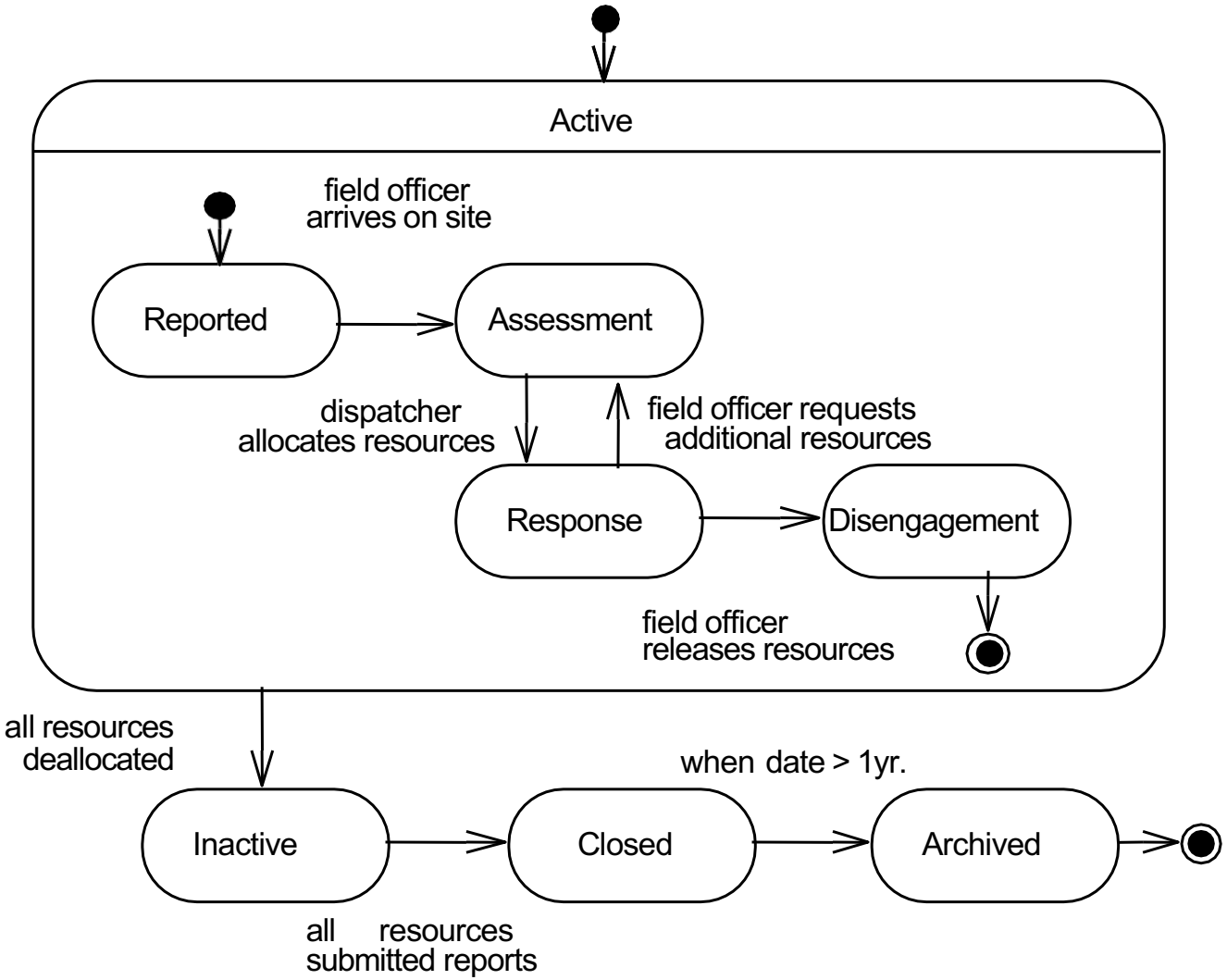
MODELLARE IL COMPORTAMENTO DIPENDENTE DALLO STATO DEGLI OGGETTI

- 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 DIPENDENTE 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

DIAGRAMMA 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 PERCHÉ 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 PORSI PERCHÉ 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 PERCHÉ 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 PERCHÉ 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?

   Creare *scenari e diagrammi dei casi d'uso*

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

Modellazione funzionale

2. Qual è la struttura del sistema?

   Creare i *diagrammi delle classi*

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

Modellazione degli oggetti

3. Qual è il comportamento?

   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

Modellazione dinamica

ANALISI IN PRATICA

Analizzare la definizione del problema

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

Costruire il modello funzionale:

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

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

Costruire il modello ad oggetti:

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

IDENTIFICARE LE ASSOCIAZIONI

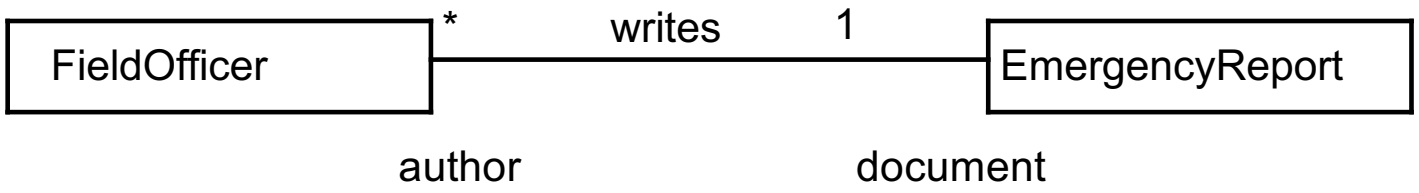
Un'associazione mostra una relazione fra due o più classi

- 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 ASSOCIAZIONE TRA LE CLASSI *EMERGENCYREPORT* E *FIELDOFFICER*



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)

IDENTIFICAZIONE DELLE ASSOCIAZIONI

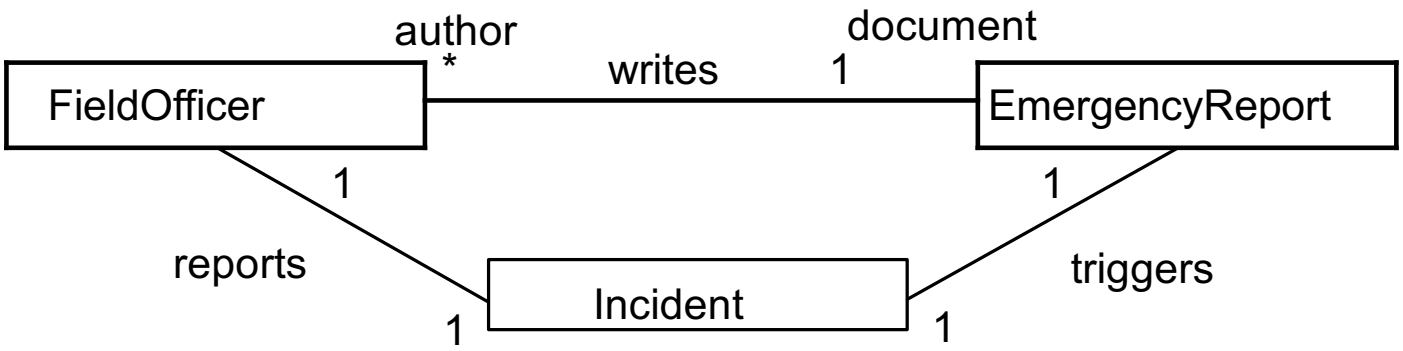
- Le associazioni fra gli oggetti sono le più importanti
  - **Rivelano molte informazioni sul dominio applicativo**
- In accordo alle euristiche di Abbott, le associazioni possono essere identificate esaminando verbi e predicati verbali che denotano uno stato (es., ***ha***, ***è parte di***, ***gestisce***, ***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



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?

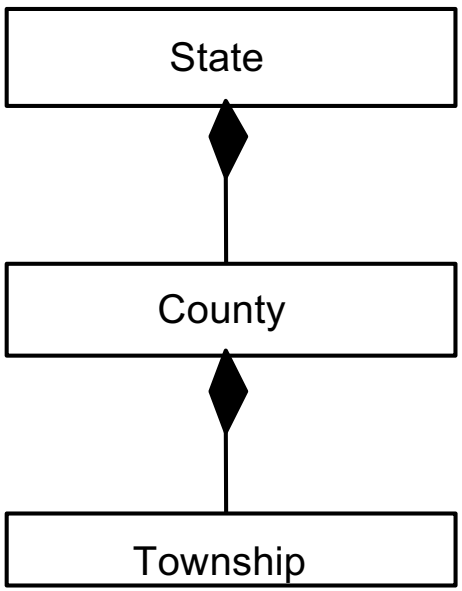
Se sono unici nell'ambito di una città, il sistema FRIEND può conoscere i FieldOfficer di più città?

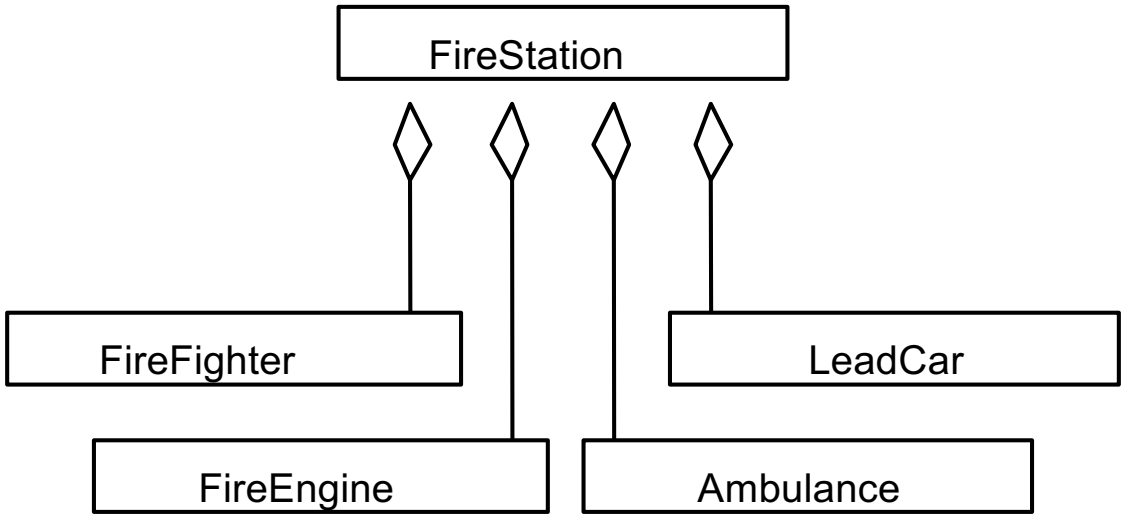
Questo approccio può essere formalizzato esaminando ogni classe ed identificando la sequenza di associazioni che devono essere attraversate per accedere a specifiche istanze di quella classe

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



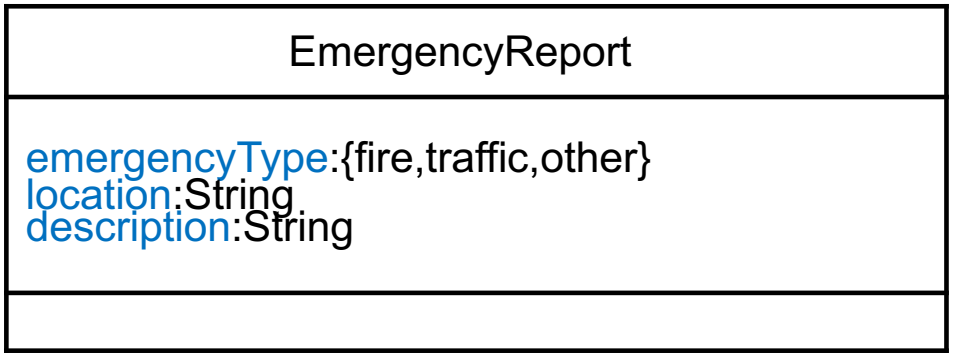


AGGREGAZIONI:COMPOSIZIONE E CONDIVISIONE

- 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

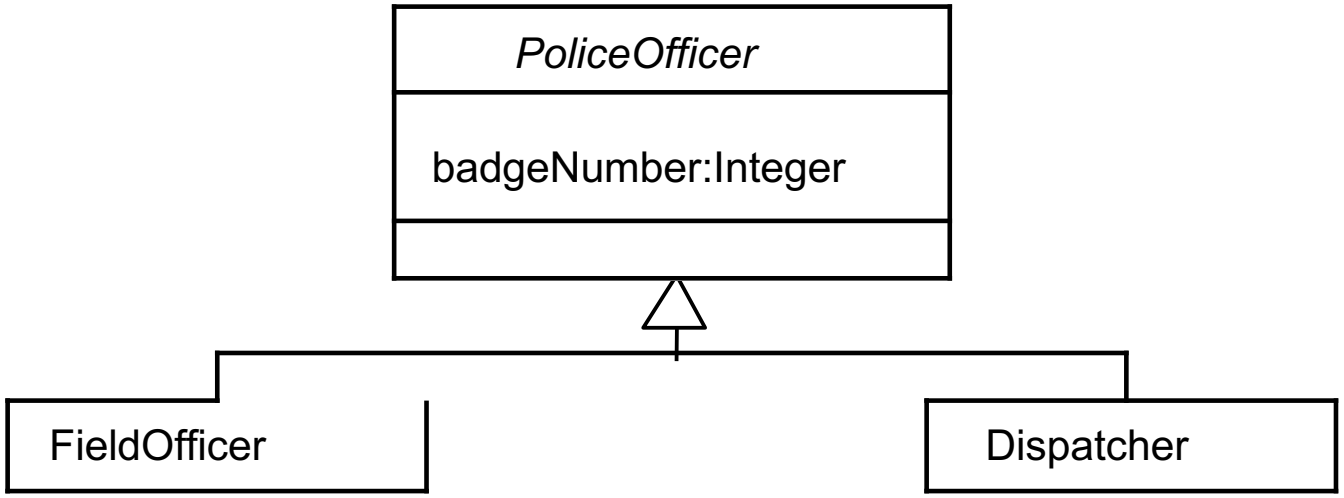
EURISTICHE PER IDENTIFICARE GLI ATTRIBUTI

- Esaminare frasi possessive
- Rappresentare stati memorizzati come attributi di oggetti entity
- Descrivere ogni attributo
- Non rappresentare un attributo come unoggetto; 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

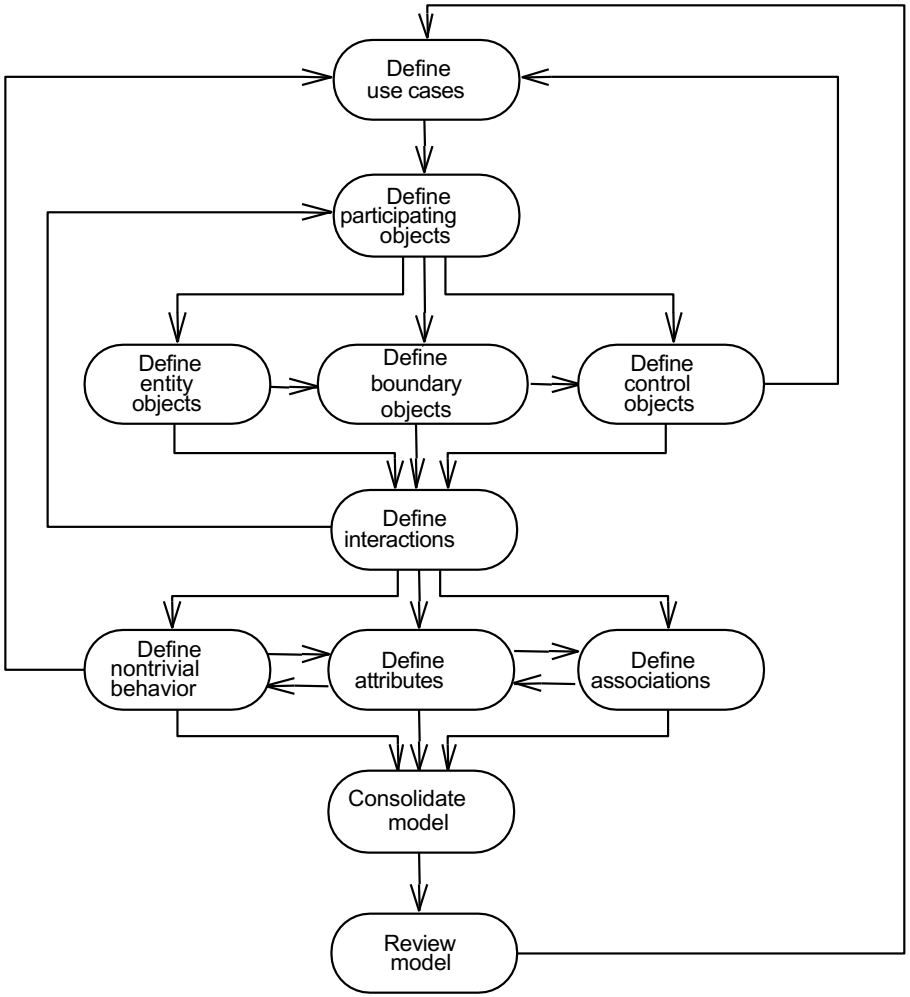
UN ESEMPIO DI RELAZIONE DI EREDITARIETÀ



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
    1. Overview
    2. Requisiti funzionali
    3. Requisiti non funzionali
    4. Vincoli (“Pseudo requisiti”)
    5. Modello del sistema
        1. Scenari
        2. Modello dei casi d’uso
        3. Modello degli oggetti
            1. Dizionario dati: oggetti Entity, Boundary, Control
            2. Diagramma delle classi
        4. Modelli dinamici
        5. Interfaccia utente
4. Glossario

SEZIONE 3.5 MODELLO DEL SISTEMA

1. Scenari
   - Scenari As-is, scenari visionari
2. Modello dei casi d'uso
   - Attori e casi d'uso
3. Modello degli oggetti
   - Dizionario dei dati
   - Diagramma delle classi (classi, associazioni, attributi e operazioni)
4. Modello dinamico
   - Diagramma degli stati per classi con comportamento dinamico significativo
   - Diagrammi delle sequenze per gli oggetti che collaborano
5. Interfaccia utente
   - Percorsi di navigazione, Screen mockup

PROGETTAZIONE DEL SISTEMA

ATTIVITÀ DI PROGETTAZIONE DEL SISTEMA: DAGLI OGGETTI AI SOTTOSISTEMI

- La progettazione del sistema consiste nel trasformare il modello di analisi nel modello di progetto prendendo in considerazione i requisiti non funzionali descritti nel documento dei requisiti
- Illustriamo queste attività con un esempio: *MyTrip*
  - Sistema di pianificazione di percorsi stradali per automobilisti
  - Partiamo con un breve modello di analisi per MyTrip
    - Successivamente descriviamo l'identificazione degli obiettivi di progetto e il progetto della decomposizione iniziale in sottosistemi

MODELLO DI ANALISI PER MYTRIP

- Usando MyTrip un automobilista può pianificare il proprio viaggio dal proprio computer di casa attraverso un servizio di pianificazione viaggio sul Web (***PlanTrip***)
  - Il viaggio viene salvato sul server per successive consultazioni

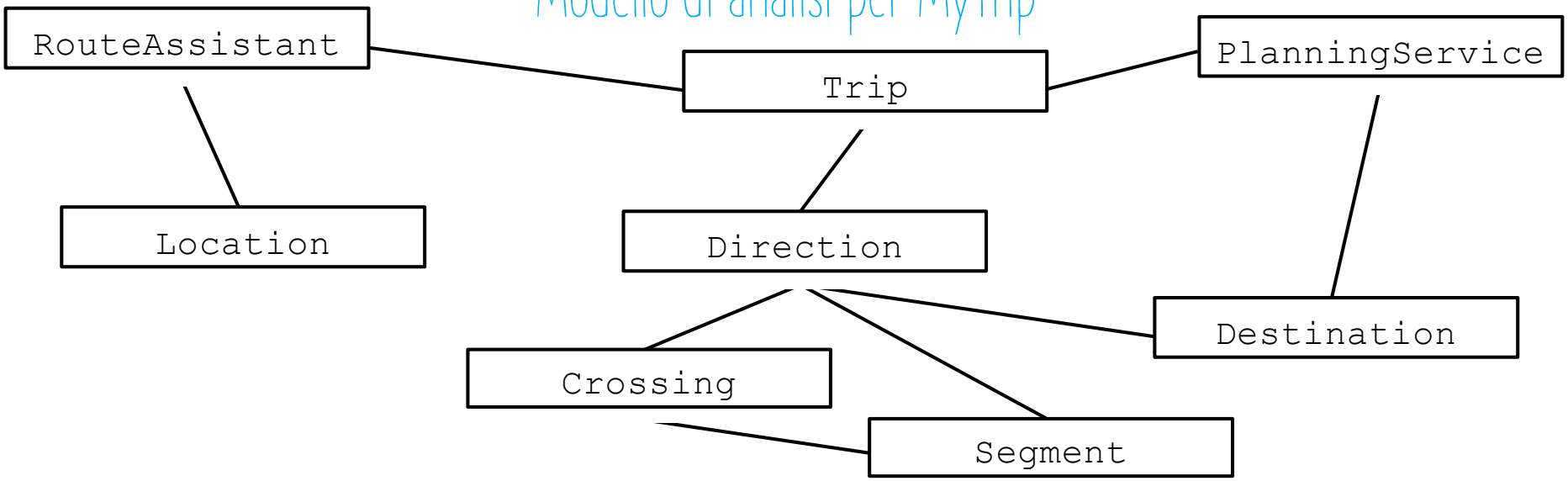
Nome caso d'uso	**PlanTrip**
Flusso di eventi	1. Il Driver attiva il proprio computer e si logga nel servizio Web di pianificazione viaggio 2. Il Driver immette i vincoli per il viaggio come sequenza di destinazioni 3. Sulla base di un db di mappe, il servizio di pianificazione calcola la via più breve per visitare le destinazioni nell'ordine specificato. Il risultato è una sequenza di segmenti che legano una serie di incroci ed una lista di direzioni 4. Il Driver può revisionare il viaggio aggiungendo o rimuovendo destinazioni 5. Il Driver salva il viaggio pianificato per nome nel db del servizio di pianificazione per successivi accessi

CASO D'USO EXECUTETRIP

- A questo punto l'automobilista va in macchina ed inizia il viaggio, mentre il computer di bordo fornisce le direzioni in base a:
  - informazioni di viaggio del servizio di pianificazione e
  - posizione corrente indicata dal sistema GPS di bordo

Nome caso d'uso	ExecuteTrip
Flusso di eventi	1. Il Driver avvia l'automobile e si logga nel sistema di bordo di assistenza al viaggio 2. Avvenuto il log-in, il Driver specifica il servizio di pianificazione ed il nome del viaggio da eseguire 3. L'assistente di bordo ottiene la lista delle destinazioni, le direzioni, i segmenti e gli incroci dal servizio di pianificazione 4. Data la posizione attuale, l'assistente di bordo fornisce all'automobilista il successivo insieme di direzioni 5. Il Driver arriva a destinazione e spegne l'assistente di bordo

Modello di analisi per MyTrip



Crossing	Punto geografico in cui si incontrano diversi segmenti
Destination	Rappresenta una località in cui il Driver desidera andare
Direction	Dato un Crossing ed un Segment adiacente, una Direction descrive in linguaggio naturale come guidare l'auto su un dato Segmento
Location	Posizione dell'auto nota a partire dal sistema GPS
PlanningService	Web server che suggerisce un viaggio, collegando un numero di destinazioni in forma di Crossing e Segment
RouteAssistant	Fornisce le direzioni all'automobilista, data la Location corrente e il prossimo Crossing
Segment	Rappresenta la strada tra due Crossing
Trip	Sequenza di Direction tra due Destination

REQUISITI NON FUNZIONALI PER MYTRIP

- Inoltre, durante la scoperta dei requisiti, il nostro cliente ha specificato i seguenti requisiti non funzionali per MyTrip:
  1. MyTrip è in contatto con il PlanningService via un modem wireless. Assumiamo che il modem funzioni correttamente alla destinazione iniziale
  2. Una volta che il viaggio è iniziato, MyTrip dovrebbe dare le direzioni corrette anche se il modem fallisce nel mantenere una connessione con PlanningService
  3. MyTrip dovrebbe minimizzare il tempo di connessione per ridurre i costi operativi
  4. La ripianificazione è possibile solo se è possibile la connessione con PlanningService
  5. Il PlanningService può supportare almeno 50 differenti automobilisti e 1000 viaggi

IDENTIFICARE GLI OBIETTIVI DI PROGETTO

- Gli obiettivi di progetto identificano le qualità su cui il nostro sistema deve focalizzarsi
- Si possono inferire numerosi obiettivi dai requisiti non funzionali o dal dominio dell'applicazione
  - Mentre altri saranno scoperti dal cliente
- Devono comunque essere definiti esplicitamente in modo che ogni importante decisione di progettazione possa avvenire in modo consistente seguendo lo stesso insieme di criteri

OBIETTIVI PER MYTRIP

Sulla base dei requisiti non funzionali, identifichiamo gli obiettivi di progetto *affidabilità* e *tolleranza agli errori per la perdita di connessione*

Successivamente identifichiamo *sicurezza* poiché numerosi automobilisti accederanno allo stesso server per la pianificazione viaggi

Aggiungiamo *modificabilità* poiché vogliamo fornire la possibilità ai guidatori di selezionare un proprio servizio di pianificazione viaggi

OBIETTIVI DI PROGETTO PER MYTRIP

- Affidabilità: MyTrip dovrebbe essere affidabile [**generalizzazione del requisito 2**]
- Tolleranza agli errori: MyTrip dovrebbe essere tollerante agli errori rispetto alla perdita della connessione con il servizio [**requisito non funzionale 2 riformulato**]
- Sicurezza: MyTrip dovrebbe essere sicuro, cioè, non deve consentire ad altri guidatori o utenti non autorizzati di accedere ai viaggi di un guidatore [**dedotto dal dominio applicativo**]
- Modificabilità: MyTrip dovrebbe essere modificabile per usare diversi servizi stradali [**anticipazione di modifiche da parte degli sviluppatori**]

OBIETTIVI DI PROGETTO

- In generale, selezioniamo gli obiettivi di progetto dalla lunga lista di qualità desiderabili in cui i criteri di design sono organizzati, in
  - Prestazioni
  - Affidabilità
  - Costi
  - Manutenzione
  - End user
- I criteri *Prestazioni*, *Affidabilità* ed *End user* solitamente sono **specificati nei requisiti** o **inferiti dal dominio applicativo**
- I criteri costi e manutenzione sono **dettati dal cliente e dal fornitore**

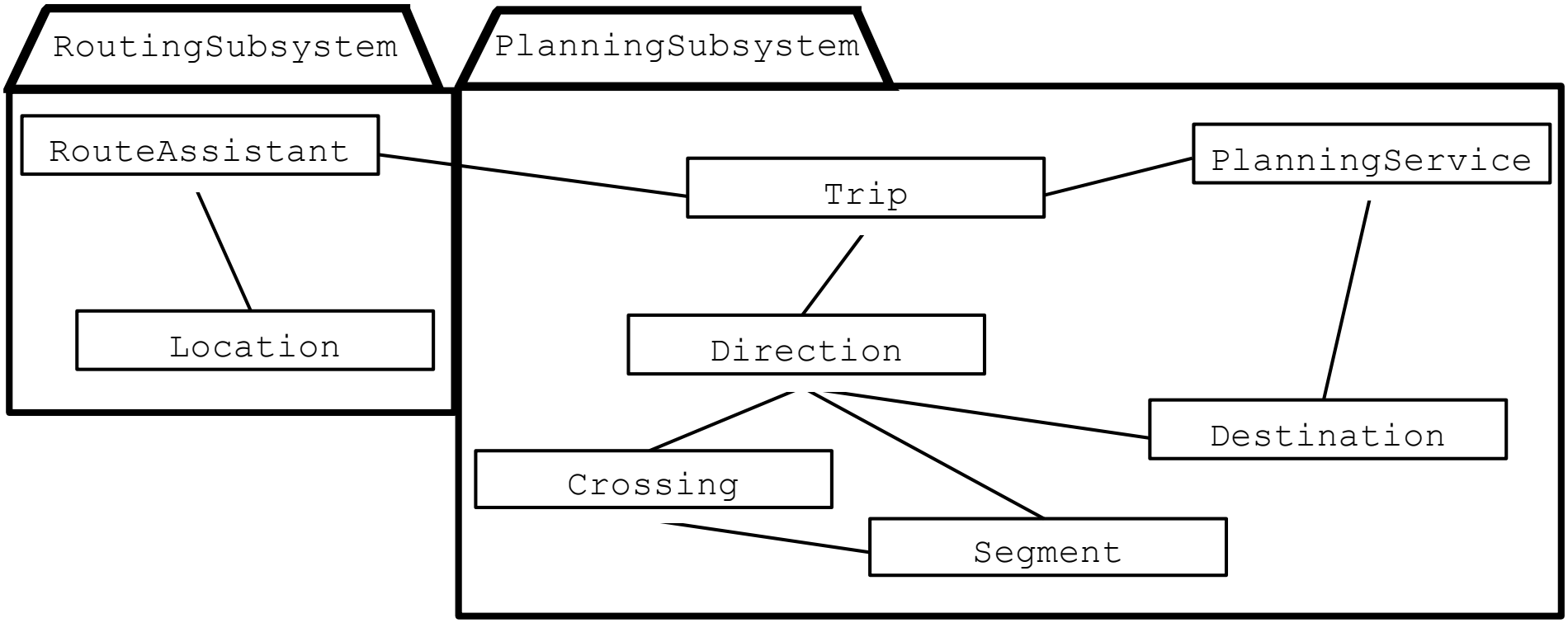
IDENTIFICARE I SOTTOSISTEMI

- L'individuazione dei sottosistemi durante la progettazione del sistema è simile al modo di trovare gli oggetti durante l'analisi
- Ad esempio, alcune tecniche di identificazione degli oggetti di analisi come le euristiche di Abbott possono essere applicate all'identificazione dei sottosistemi
- Inoltre, la decomposizione in sottosistemi viene revisionata costantemente laddove sono esaminate nuove questioni
  - Diversi sottosistemi sono fusi in un sottosistema, un sottosistema complesso è suddiviso in parti, sono aggiunti altri sottosistemi per gestire nuove funzionalità
  - Le prime iterazioni sulla decomposizione in sottosistemi possono introdurre cambiamenti drastici nel modello di progetto del sistema
    - Sono solitamente meglio gestiti attraverso dei brainstorming

SOTTOSISTEMI PER MYTRIP

- La decomposizione iniziale in sottosistemi dovrebbe essere derivata dai requisiti funzionali
  - In MyTrip identifichiamo due gruppi principali di oggetti: quelli coinvolti durante il caso d'uso **PlanTrip** e quelli coinvolti nel caso d'uso **ExecuteTrip**
  - Le classi Trip, Direction, Crossing, Segment e Destination sono condivise tra i due casi d'uso. Tale insieme di classi è altamente accoppiato poiché sono usate come un tutt'uno per rappresentare un viaggio
    - Decidiamo allora di assegnarle con PlanningService a PlanningSubsystem e le restanti classi sono assegnate a RoutingSubsystem
    - Otteniamo così una sola associazione tra i due sottosistemi

Decomposizione per MyTrip



E’ da osservare che tale decomposizione è un repository in cui PlanningSubsystem è responsabile per la struttura dati centrale

IDENTIFICAZIONE DEI SOTTOSISTEMI

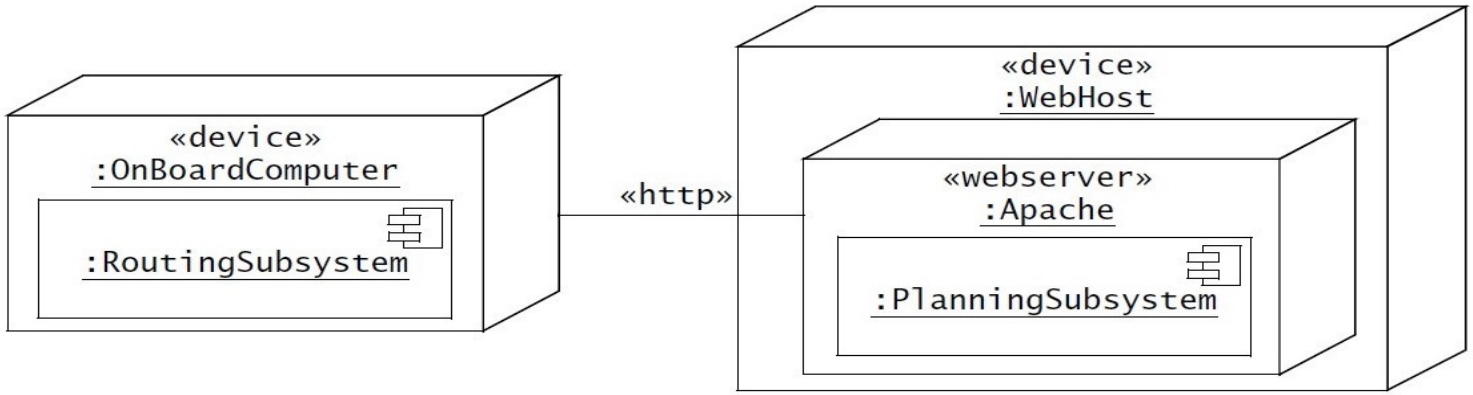
- Un'altra euristica per l'identificazione dei sottosistemi consiste nel mantenere insieme oggetti funzionalmente collegati
- Un punto di partenza è assegnare gli oggetti partecipanti identificati in ciascun caso d'uso ai sottosistemi
  - Alcuni gruppi di oggetti, come il gruppo *Trip* in *MyTrip* sono condivisi e usati per comunicare le informazioni da un sottosistema ad un altro
- Possiamo creare sia un nuovo sottosistema nel quale allocarli che assegnarli al sottosistema che li crea
- Per gestire la complessità del dominio della soluzione minimizzando l'accoppiamento tra i sottosistemi è possibile ricorrere ai **design pattern**
  - I design pattern sono schemi di soluzioni progettuali che gli sviluppatori hanno messo a punto e rifinito nel corso degli anni per risolvere una gamma di problemi ricorrenti

EURISTICHE

- Assegnare gli oggetti identificati in un caso d'uso nello stesso sottosistema
- Creare un sottosistema dedicato per gli oggetti usati per muovere i dati tra i sottosistemi
- Minimizzare il numero di associazioni che attraversano i confini dei sottosistemi
- Tutti gli oggetti nello stesso sottosistema dovrebbero essere funzionalmente legati

MAPPING DEI SOTTOSISTEMI A PROCESSORI E COMPONENTI

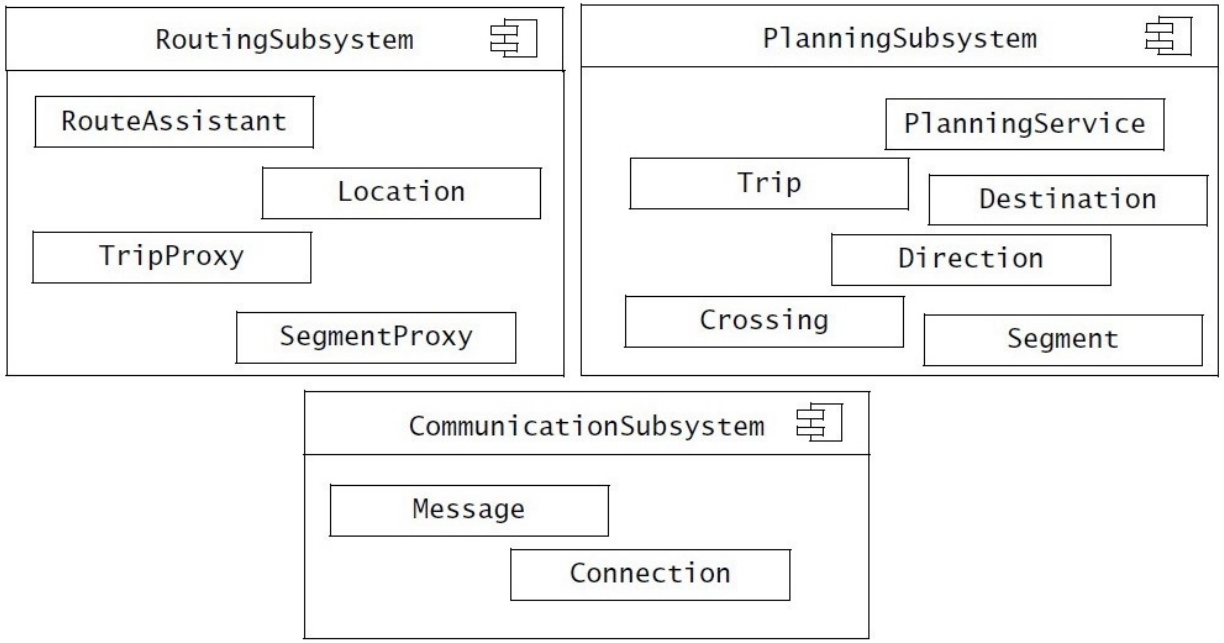
- L'attività di mapping dello HW ha un impatto fondamentale su prestazioni e complessità del sistema, si esegue nelle prime fasi della progettazione
- In Mytrip, deduciamo dai requisiti che PlanningSubsystem e RoutingSubsystem sono eseguiti su due nodi diversi:
  - Il primo è un servizio basato sul web su un host Internet
  - il secondo è eseguito su un computer di bordo
- La figura mostra l'allocazione dello hw per Mytrip con due dispositivi chiamati **:OnBoardComputer** e **:WebHost**, e un ambiente di esecuzione chiamato **:Apache**



ALLOCAZIONE DI OGGETTI E SOTTOSISTEMI AI NODI

- Tale fase spesso comporta l'identificazione di nuovi oggetti e sottosistemi per per trasportare i dati tra i nodi
- MyTrip
  - RoutingSubsystem e PlanningSubsystem condividono gli oggetti Trip, Destination, Crossing, Segment e Direction
  - Le istanze delle classi comunicano via modem mediante un protocollo di comunicazione
    - Si introduce un nuovo sottosistema per supportare la comunicazione: CommunicationSubsystem, localizzato su ambo i nodi per gestirne la comunicazione

ALLOCAZIONE DI OGGETTI E SOTTOSISTEMI AI NODI



CommunicationSubsystem	The CommunicationSubsystem is responsible for transporting objects from the PlanningSubsystem to the RoutingSubsystem.
Connection	A Connection represents an active link between the Planning-Subsystem and the RoutingSubsystem. A Connection object handles exceptional cases associated with loss of network services.
Message	A Message represents a Trip and its related Destinations, Segments, Crossings, and Directions, encoded for transport.

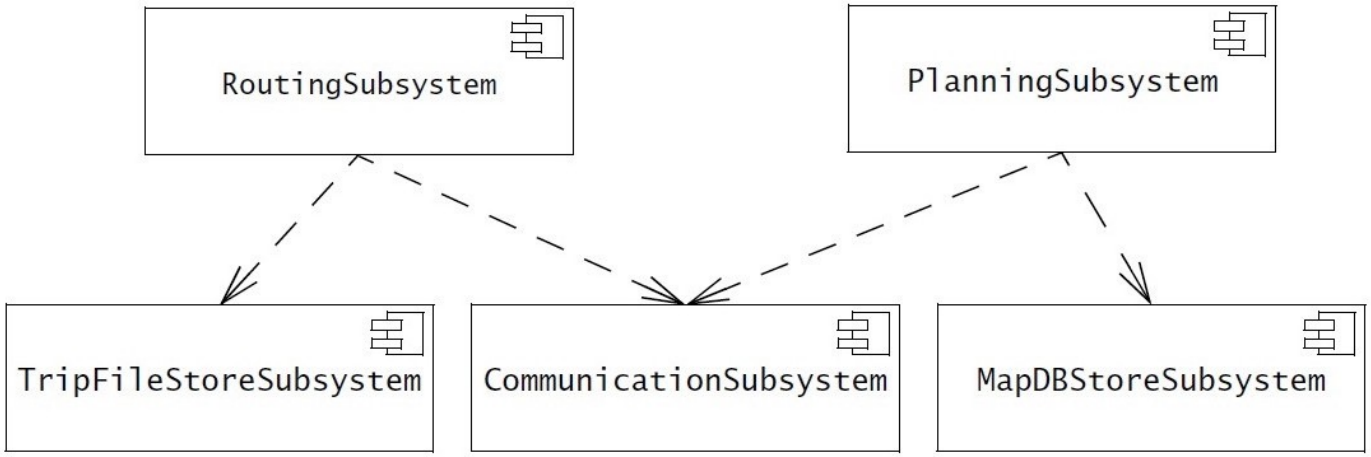
ALLOCAZIONE DI OGGETTI E SOTTOSISTEMI AI NODI

- Solo i segmenti che formano il viaggio pianificato sono memorizzati nel RoutingSubsystem
- I segmenti adiacenti che non ne fanno parte sono memorizzati nel solo PlanningSubsystem
- Se il guidatore ripianifica il viaggio, deve poter richiedere al CommunicatonSubsystem di recuperare le informazioni associate con i suoi segmenti nel PlanningSubsystem
- Per tenerne conto, abbiamo bisogno di oggetti in RoutingSubsystem che possano fungere da surrogati di Segment e Trip nel PlanningSubsytem
  - In RoutingSubsytem si introducono gli oggetti SegmentProxy e TripProxy
  - I proxy sono esempi di del design pattern Proxy
- Infine, il CommunicationSubsytem è usato per trasferire un viaggio completo da PlanningSubsystem al RouteAssistant

IDENTIFICAZIONE DATI PERSISTENTI

- Dove e quando i dati sono memorizzati nel sistema impatta la decomposizione del sistema
  - In alcuni casi, con uno stile architetturale Repository, un sottosistema può essere dedicato in toto alla memorizzazione dei dati
  - La scelta di uno specifico dbms può avere anche implicazioni sulla strategia di controllo e la gestione della concorrenza
- In MyTrip, memorizziamo il viaggio corrente in un file su disco rimovibile per consentire il recupero di Trip nel caso in cui il guidatore spenga l'auto prima di prima di raggiungere la destinazione
  - Semplice ed efficiente poiché RoutingSubsytem memorizzerà viaggi completi nel file solo prima dell'arresto per poi caricare il file all'avvio
- In PlanningSubsystem i viaggi saranno memorizzati in un database
  - Usato per gestire tutti i viaggi di molti guidatori e le mappe necessarie per generare i viaggi

IDENTIFICAZIONE DATI PERSISTENTI



TripFileStoreSubsystem The TripFileStoreSubsystem is responsible for storing trips in files on the onboard computer. Because this functionality is only used for storing trips when the car shuts down, this subsystem only supports the fast storage and loading of whole trips.

MapDBStoreSubsystem The MapDBStoreSubsystem is responsible for storing maps and trips in a database for the PlanningSubsystem. This subsystem supports multiple concurrent Drivers and planning agents.

IDENTIFICAZIONE OGGETTI PERSISTENTI

- Gli oggetti entity identificati durante l'analisi sono candidati per la persistenza
  - MyTrip: Trip, Crossing, Destination, PlanningService e Segment
  - N.B.: non tutti gli oggetti entity sono persistenti: Location e Direction sono ricalcolati costantemente al muoversi dell'auto
- Gli oggetti persistenti non sono solo oggetti entity
  - In un sistema multiutente, le informazioni sugli utenti sono persistenti (Driver) ma anche alcuni attributi di oggetti boundary (posizione finestra, preferenze interfaccia utente, stato oggetti di controllo di lunga esecuzione)
- In generale, gli oggetti persistenti si identificano esaminando tutte le classi che devono sopravvivere allo shutdown del sistema (spegnimento controllato o crash)

STRATEGIA GESTIONE

MEMORIZZAZIONE

Trade-off between flat files, relational databases, and object-oriented databases

When should you choose flat files?

- Voluminous data (e.g., images)
- Temporary data (e.g., core file)
- Low information density (e.g., archival files, history logs)

When should you choose a relational or an object-oriented database?

- Concurrent accesses
- Access at finer levels of detail
- Multiple platforms or applications for the same data

When should you choose a relational database?

- Complex queries over attributes
- Large data set

When should you choose an object-oriented database?

- Extensive use of associations to retrieve data
- Medium-sized data set
- Irregular associations among objects

CONTROLLO ACCESSI

Durante la progettazione, modelliamo l'accesso determinando

- quali oggetti sono condivisi tra gli attori e
- definendo come gli attori possono controllare l'accesso

A seconda dei requisiti di sicurezza del sistema possiamo anche definire

- come autenticare gli attori nel sistema e
- selezionare quali dati del sistema devono essere crittografati

CONTROLLO ACCESSI

- MyTrip
  - Memorizzare mappe e viaggi per molti guidatori nello stesso DB può introdurre problemi di sicurezza
  - Dobbiamo assicurare che i viaggi siano inviati al solo guidatore che li ha creati
    - Modelliamo il guidatore con la classe Driver e l'associamo con la classe Trip
      - Il driver è un guidatore autorizzato
    - Il sistema PlanningSubsytem diventa anche responsabile per l'autenticazione dei guidatori prima di inviare i viaggi
    - Decidiamo di crittografare il traffico di comunicazione tra RoutingSubsytem e PlanningSubsystem (fatto da CommunicationSubsystem)

DECISIONI SUL FLUSSO DI CONTROLLO GLOBALE

- Flusso di controllo: sequenza di azioni nel sistema
- Nei sistemi OO, la sequenza delle azioni comporta decidere quali operazioni eseguire e in quale ordine
  - Eventi esterni generati da un attore o dal trascorrere del tempo
- Tre meccanismi possibili
  - Controllo guidato da procedura
  - Controllo guidato da eventi
  - Thread

ESEMPIO DI CONTROLLO GUIDATO DA PROCEDURA

- Le operazioni attendono gli input se hanno bisogno di dati da un attore
  - Sistemi legacy e sistemi scritti in linguaggi procedurali

```
    Stream in, out;
    String userid, passwd;
/* Initialization omitted */
    out.println("Login:");
    in.readln(userid);
    out.println("Password:");
    in.readln(passwd);
    if (!security.check(userid, passwd)) {
        out.println("Login failed.");
        system.exit(-1);
    }
/* ...*/
```

Stampa messaggi e attende input da utente

CONTROLLO GUIDATO DA EVENTI

- Un loop principale attende un evento esterno
  - Quando un evento si verifica, è attribuito all'oggetto appropriato, sulla base dell'informazione associata all'evento

```
Iterator subscribers, eventStream;
Subscriber subscriber;
Event event;
EventStream eventStream;
/* ... */
while (eventStream.hasNext()) {
    event = eventStream.next();
    subscribers = dispatchInfo.getSubscribers(event);
    while (subscribers.hasNext()) {
        subscriber = subscribers.next()) {
        subscriber.process(event);
    }
}
/* ... */
```

Un evento è preso da eventstream ed inviato agli oggetti interessati

THREAD

- Variante concorrente di controllo guidato da procedura
- Il sistema può creare un numero di thread e ognuno risponde ad un evento diverso
  - Se un thread ha bisogno di dati attende tali dati da un attore specifico

```
Thread thread;
Event event;
EventHandler eventHandler;
boolean done;
/* ...*/
while (!done) {
     event = eventStream.getNextEvent();
     eventHandler = new EventHandler(event)
     thread = new Thread(eventHandler);
     thread.start();
}
/* ...*/
```

Eventhandler è un oggetto dedicato alla gestione di event

DECISIONI SUL FLUSSO DI CONTROLLO GLOBALE

- Fissato il meccanismo del flusso di controllo lo si realizza con un insieme di uno o più oggetti di controllo
- Gli oggetti di controllo
  - registrano eventi esterni
  - memorizzano stati temporanei ad essi relativi e
  - determinano l'invio dell'ordine corretto di chiamate ad operazioni su oggetti boundary e entity associate con l'evento esterno
- Localizzare il flusso di controllo per un caso d'uso in un singolo oggetto
  - codice più leggibile
  - Sistema resiliente ai cambiamenti nelle implementazione del flusso di controllo

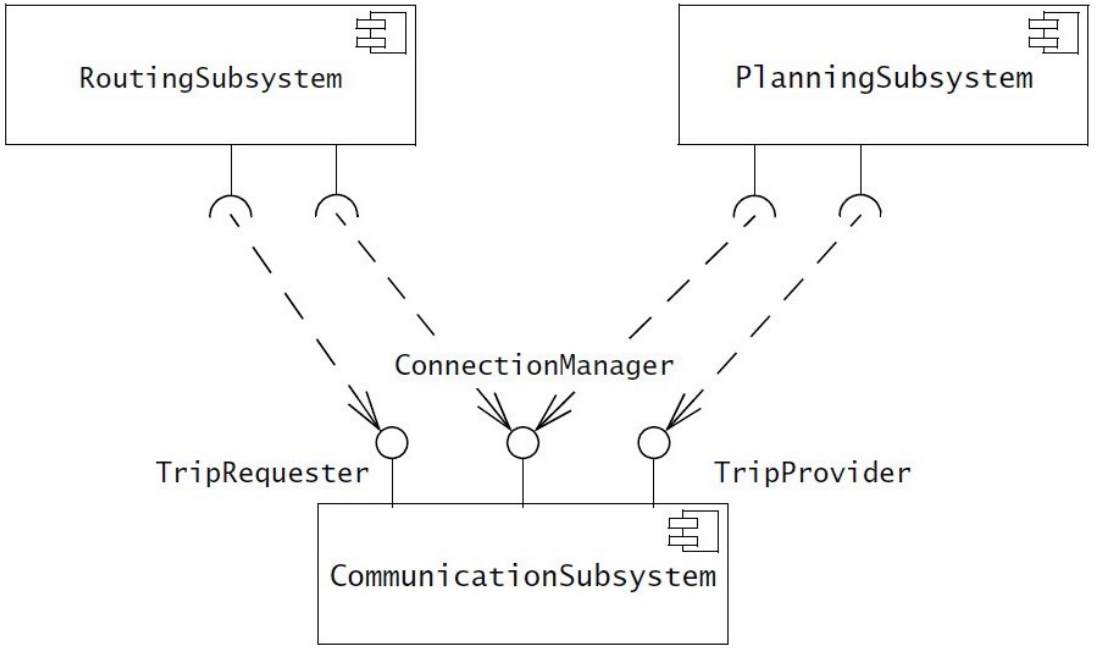
IDENTIFICAZIONE SERVIZI

- Finora abbiamo esaminato le decisioni di progettazione chiave che impattano sulla decomposizione
- Abbiamo identificato i sottosistemi ed abbiamo un'idea approssimativa di come allocare le responsabilità ai sottosistemi
- Ora dobbiamo rifinire la decomposizione del sottosistema identificando i servizi forniti da ciascuno
- I servizi forniti servono per gestire le connessioni, e fare uploading e downloading di trip
- Rivediamo le dipendenze tra i sottosistemi e definiamo l'interfaccia per ogni servizio identificato
- In tale attività diamo i nomi ai servizi identificati

- La responsabilità di CommunicationSubsystem è di trasportare i viaggi dal PlanningSubsistem a RoutingSubsytem
- RoutingSubsystem inizia la connessione poiché PlanningSubsystem è il server e sempre disponibile, mentre Routing solo quando la macchina è accesa

Identificazione servizi

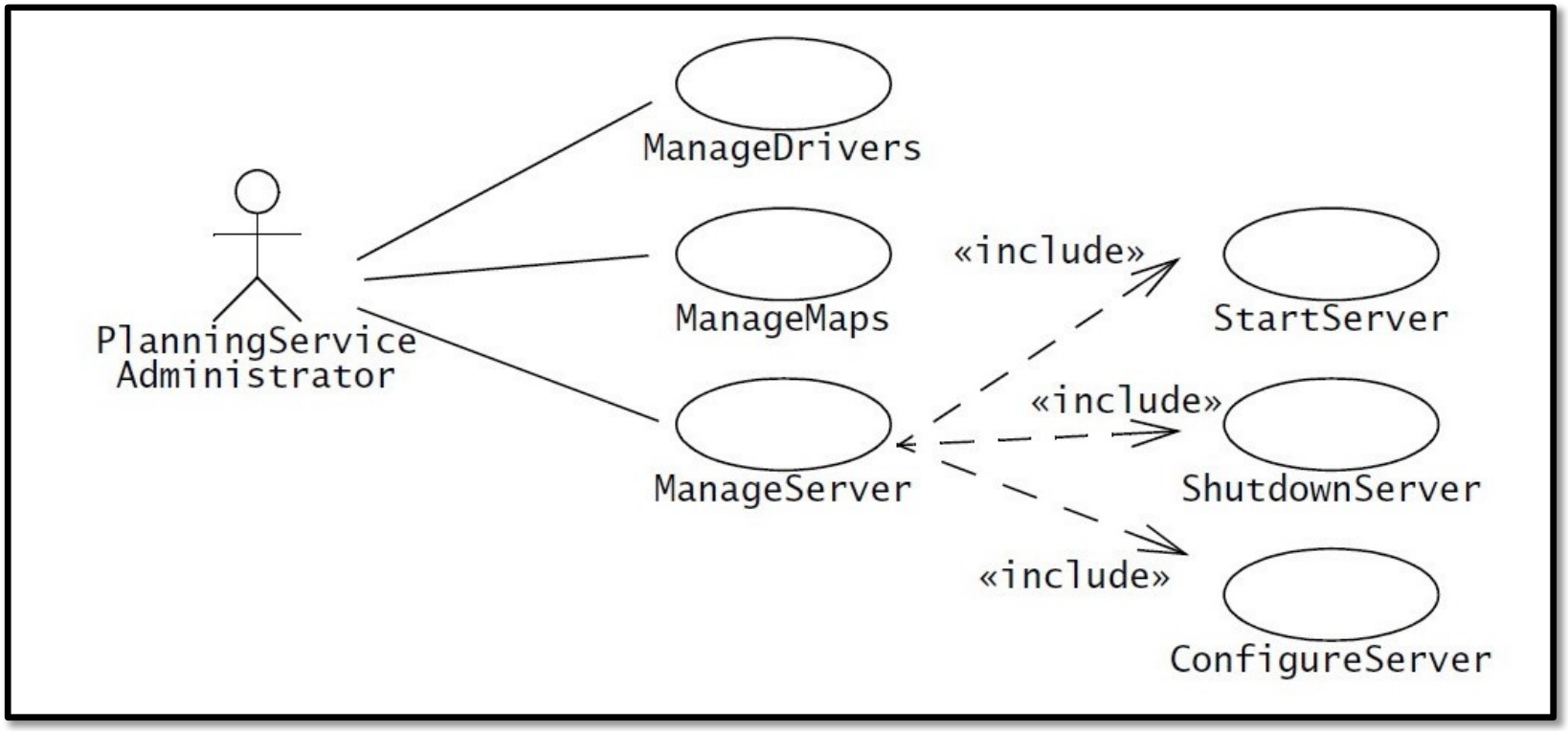
- **ConnectionManager**: permette ad un sottosistema di registrarsi con CommunicationSubsystem, autenticarsi, trovare altri nodi e iniziare e chiudere la connessione
- **TripRequester**: permette ad un sottosistema di richieder una lista di viaggi disponibili e fare il download del viaggio selezionato
- **TripProvider**: permette ad un sottosistema di fornire una lista di viaggi disponibili per lo specifico guidatore e rispondere a specifiche richieste di viaggio



IDENTIFICARE LE CONDIZIONI LIMITE

- Condizioni limite
  - Avvio
  - Inizializzazione
  - Spegnimento
  - Gestione guasti
- **Boundary use case**
- **MyTrip**
  - **Inizializzazione: caricamento mappe in PlanningService? Installazione nell'auto?**
  - **…**

Casi d’uso boundary



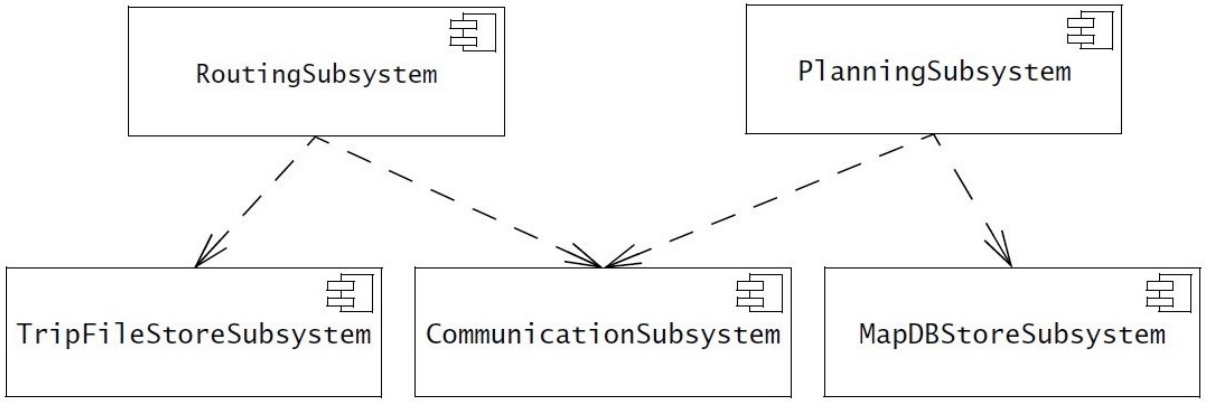
- **ManageDrivers**: per aggiungere, rimuovere e modificare i guidatori
- **ManageMaps**: per aggiungere, rimuovere e modificare le mappe per generare i viaggi
- **ManageServer**: per eseguire routine di configurazione, avvio, spegnimento

Caso d'uso Startserver

Use case name	`StartServer`
Entry condition	1. The `PlanningServiceAdministrator` logs into the server machine.
Flow of events	2. Upon successful login, the `PlanningServiceAdministrator` executes the `startPlanningService` command.
	3. If the `PlanningService` was previously shutdown normally, the server reads the list of legitimate `Drivers` and the index of active `Trips` and `Maps`. If the `PlanningService` had crashed, it notifies the `PlanningServiceAdministrator` and performs a consistency check on the `MapDBStore`.
Exit condition	4. The `PlanningService` is available and waits for connections from `RoutingAssistants`.

Revisione MapDBStoreSubsystem

- La revisione del diagramma dei casi d'uso non influenza la decomposizione del sistema. Tuttavia …
  - Abbiamo aggiunto tre casi d'uso nuovi ai sottosistemi esistenti
  - **MapDBStoreSubsystem** dovrebbe essere in grado di individuare se o no è stato arrestato correttamente e dovrebbe essere in grado di fare controlli di consistenza e riparare i dati corrotti



MapDBStoreSubsystem

The MapDBStoreSubsystem is responsible for storing maps and trips in a database for the PlanningSubsystem. This subsystem supports multiple concurrent Drivers and planning agents.

Revisione MapDBstoresubsystem

MapDBStoreSubsystem	The MapDBStoreSubsystem is responsible for storing Maps and Trips in a database for the PlanningSubsystem. This subsystem supports multiple concurrent Drivers and planning agents. *When starting up, the MapDBStoreSubsystem detects if it was properly shut down. If not, it performs a consistent check on Maps and Trips and repairs corrupted data if necessary.*

Documento di progettazione

System Design Document

1. Introduction
   - 1.1 Purpose of the system
   - 1.2 Design goals
   - 1.3 Definitions, acronyms, and abbreviations
   - 1.4 References
   - 1.5 Overview
2. Current software architecture
3. Proposed software architecture
   - 3.1 Overview
   - 3.2 Subsystem decomposition
   - 3.3 Hardware/software mapping
   - 3.4 Persistent data management
   - 3.5 Access control and security
   - 3.6 Global software control
   - 3.7 Boundary conditions
4. Subsystem services

Glossary