Ingegneria dei Requisiti

Corso tenuto dal Professor Mariano Ceccato

Università degli Studi di Verona

Alessio Gjergji

Indice

Capitolo 1

Introduzione

1.1 Terminologia di base

Per avere un'implementazione corretta e completa del software dobbiamo fare in modo che il software risolva un problema concreto del mondo reale. Dobbiamo quindi comprendere correttamente il mondo reale, ovvero come funziona, come dovrebbe funzionare, quali sono i vincoli che governano, capire il contesto in cui il software deve operare. Questo è il compito dell'ingegneria dei requisiti.

Esempio

- **Problema**: la gestione del freno a mano in un'automobile moderna potrebbe essere automatizzato.
- **Contesto**: se l'automobile è in movimento, se si sta frenando, se è intenzione del guidatore fermarsi, se il guidatore ha premuto il pedale del freno, ...

Quando parliamo di **requisiti** dobbiamo inoltre definire il **mondo**. Quando definiamo il mondo, abbiamo a che fare con elementi molto complessi che contengono di fatto elementi umani (*lo staff, l'utente, il cliente, ...*), elementi fisici (*dispositivi, software legacy, la natura, ...*) e dovrà quindi adattarsi alla situazione esistente.

Dobbiamo definire anche il mondo delle **macchine**. Parliamo quindi di tutto ciò che dovrà essere implementato e/o acquistato, come ad esempio *database, server, client, ...* e i componenti hardware e software che dovranno essere implementati.

L'ingegneria dei requisiti non si limita solamente al mondo delle macchine, ma tiene in considerazione anche gli effetti che il software ha sul mondo reale, che dovrà modellare.

Il mondo e le macchine hanno i propri fenomeni, ma ne condividono anche alcuni. Ad esempio, il mondo ha il *rilascio del freno a mano*, mentre le macchine hanno `errorCode = 013`. Nell'intersezione potremmo avere ad esempio `motor.Regime =‘up‘`.

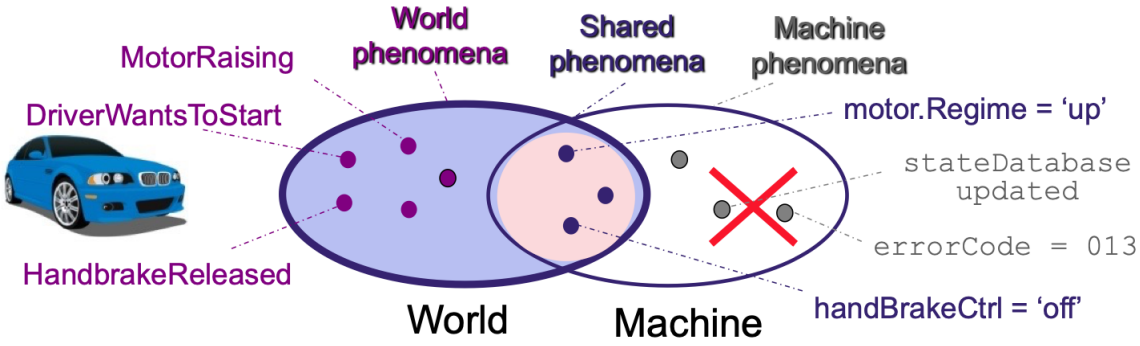


Figura 1.1.1: Il mondo e le macchine

Quando parliamo di requisiti, consideriamo solamente la parte relativa al mondo, che comprende quindi anche l'intersezione. È importante comprendere che non descrivono **nulla** riguardo alle macchine.

1.1.1 Le due versioni del mondo

Il sistema è l'insieme delle interazioni delle componenti che strutturano il mondo.

- Il sistema **as-is**: si tratta del sistema prima che il software venga implementato. Questo sistema è composto da solamente dal mondo.
- Il sistema **to-be**: si tratta del sistema come dovrebbe essere quando il software opererà. Questo sistema è composto dall'unione del mondo e delle macchine.

Definizione preliminare dell'ingegneria dei requisiti

Consiste in una serie di attività collegate tra loro che ci permettono di esplorare, valutare, documentare, consolidare, rivedere e adattare quelli che sono gli obiettivi, le capacità, i vincoli e le assunzioni in un sistema software. Basato sui problemi del sistema **as-is** e le opportunità date dalle nuove tecnologie.

1.1.2 I requisiti di sistema e i requisiti software

Il sistema (*unione del mondo reale e del sistema software*) sono i requisiti che fanno riferimento a tutto, si tratta degli statement che fanno parte del **system to-be**, mentre i requisiti software sono un sottoinsieme dei requisiti di sistema, fanno quindi riferimento solamente al **software to-be**, di fatto le funzionalità che il software che dovrà fornire per rispondere in maniera adeguata ai problemi del nostro utente.

1.2 Scope dell'ingegneria dei requisiti

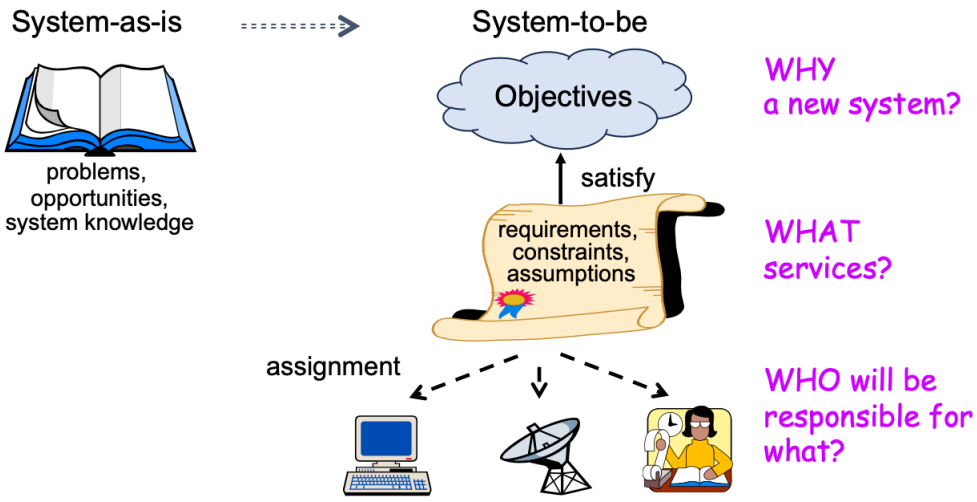


Figura 1.2.1: Scope dell'ingegneria dei requisiti

Prima di iniziare a lavorare sui requisiti, abbiamo il nostro system as-is, che comprenderà i problemi, le opportunità e le conoscenze sul sistema. Da qui mapperemo il tutto nel system to-be. Per comprendere al meglio il system to-be, dobbiamo ragionare in tre prospettive:

- Perché: quali sono gli obiettivi dell'abbinamento tra software e mondo reale. Quindi quali sono le necessità che il software dovrà soddisfare.
- Cosa: quali sono le funzionalità che il software dovrà fornire per soddisfare i bisogni del cliente, quali saranno i vincoli e le assunzioni che il software dovrà rispettare.
- Chi: chi sarà responsabile della fornitura delle funzionalità.

Questa è la chiave di lettura quando ragioniamo in termini di software: dobbiamo capire quali sono le funzionalità esposte e come queste danno una risposta agli obiettivi che ci siamo posti e chi ha la responsabilità di implementare tali funzionalità, se una persona, un componente interno al sistema, un componente esterno, ... Ci saranno quindi delle viste diverse che ci permetteranno di ragionare in questo spazio tridimensionale.

1.2.1 La dimensione del perché

Si tratta dell'identificazione degli obiettivi di cui il sistema software dovrà dare risposta. Vanno identificati, perciò l'interazione con gli stakeholder è essenziale, ma vanno inoltre analizzati. Tali obiettivi andranno poi rifiniti, perché tipicamente quello che viene raccolto in prima battuta è molto astratta e di alto livello. Vedremo in seguito come questi obiettivi verranno rifiniti e a che livello di dettaglio verranno portati.

Ci sono delle difficoltà intrinseche nel definire gli obiettivi:

- È importante capire il dominio del problema, ovvero capire il contesto in cui il software dovrà operare.
- Valutare opzioni alternative (*strade differenti per soddisfare lo stesso obiettivo*). Qui è opportuno limitare le scelte per poter fare una scelta più consapevole quando si avrà una conoscenza del dominio più approfondita per poter valutare in maniera più consapevole le opzioni alternative.
- Potremmo avere obiettivi in contrasto tra loro.

1.2.2 La dimensione del cosa

Sostanzialmente mira a identificare le funzionalità messe a disposizione all'interno del sistema software. L'obiettivo è date una risposta agli obiettivi identificati nella fase precedente in maniera adeguata. Ci saranno quindi delle metriche per misurare l'adeguatezza di tali risposte in modo che gli utenti siano soddisfatti a pieno delle funzionalità.

Le risposte dovranno essere fornite in maniera realistica, elaborando una soluzione che sia compatibile con quelli che sono i vincoli ambientali.

Anche qui ci sono delle difficoltà:

- Identificazione delle esatte funzionalità che dovranno essere fornite.
- Le funzionalità devono essere descritte in modo che siano comprensibili da tutti gli attori coinvolti, dove gli **attori** sono sia gli ingegneri del software e dei requisiti, sia gli stakeholder.
- Garantire una tracciabilità tra gli obiettivi rispetto alle funzionalità, ovvero un *link* tra gli obiettivi e le funzionalità che dovranno essere implementate. Il link è importante perché se ci accorgiamo che la motivazione cambia, probabilmente cambierà anche il "cosa".

1.2.3 La dimensione del chi

Si tratta di identificare chi sarà responsabile della fornitura delle funzionalità. Le varie responsabilità possono essere distribuite tra più attori, sia interni che esterni al sistema, possono essere quindi hardware o software o persone.

La difficoltà principale è quella di identificare il giusto grado di autonomia. Anche qui è opportuno documentare le alternative ma non adottarle subito, in modo da poter fare una scelta più consapevole in seguito. Solitamente si adotta un approccio iterativo raffinando le scelte in base alle informazioni che si acquisiscono nel tempo.

1.3 Tipologie di requisiti

Quando parliamo di requisiti, possiamo adottare due tipologie di registri:

- **Descrittivo**: raccontiamo le proprietà del sistema **indipendentemente** da come dovrebbe funzionare. Dipendendo quindi la leggi naturali, le leggi fisiche, i vincoli di sistema.

  Esempio: se le porte del treno sono chiuse, allora non sono aperte.

- **Percettivo**: raccontiamo le proprietà desiderabili nel sistema che potrebbero dipendere o meno dal comportamento del sistema.

  Esempio: le pote dovrebbero sempre rimanere chiuse quando il treno è in movimento.

La seconda tipologia di frasi sono quelle che possono essere negoziate, cambiate, modificate, mentre le prime sono inalterabili.

Gli statement formulati possono avere differenze per quanto riguarda lo scope. Quindi potrebbero riferirsi a fenomeni che riguardano solamente l'ambiente o tra l'ambiente e il sistema software.

1.3.1 Requisiti di sistema e requisiti software

I requisiti di sistema sono frasi prescrittive che si riferiscono ai fenomeni ambientali e dovranno essere soddisfatti dal sistema software. Tali frasi dovranno essere formulate con un vocabolario che sia comprensibile da tutti gli attori coinvolti. Ad esempio, `TrainMoving` $\rightarrow$ `DoorsClosed`.

I requisiti software sono frasi prescrittive che si riferiscono ai fenomeni condivisi tra il sistema software e l'ambiente, che sono supportate dal software-to-be. Tali frasi sono formulate nel vocabolario degli sviluppatori. Ad esempio, `measuredSpeed` $\geq 0 \rightarrow$ `doorState` = `'closed'`.

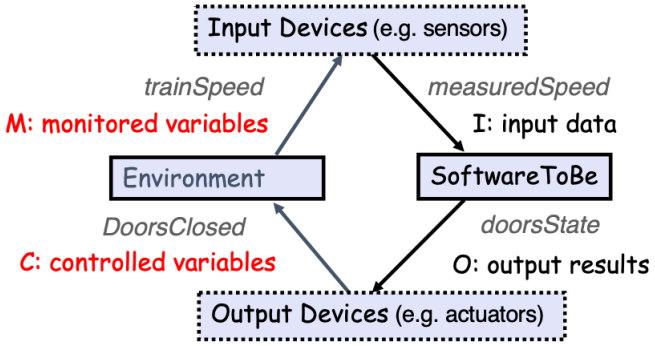
Proprietà di dominio, assunzioni e definizioni

Una proprietà di **dominio** è frase descrittiva che riguarda un fenomeno del mondo reale, che avviene indipendentemente dall'esistenza del sistema software. Ad esempio, "se l'accelerazione del treno è maggiore di 0, allora la velocità del treno è diversa da 0".

Le **assunzioni** sono frasi che devono essere soddisfatte dall'ambiente quando andiamo a considerare il software-to-be e sono formulate in termini dei fenomeni del mondo reale. Tipicamente sono prescrittive, ma non sempre. Ad esempio, "la velocità misurata è diversa da 0 se e solo se la velocità del treno è diversa da 0".

Le **definizioni** sono frasi che definiscono il significato preciso del concetto del sistema o della terminologia ausiliaria. Ad esempio, "la velocità misurata è la velocità misurata dal tachimetro del treno".

1.3.2 Relazione tra i requisiti software e i requisiti di sistema



Quando parliamo di requisiti software abbiamo questo modello quaternario. Possiamo avere degli *input device* e degli *output device* che sono i componenti che permettono di far si che il software possa interagire con il mondo reale.

Mappatura dei requisiti software e dei requisiti di sistema

Se i requisiti software sono soddisfatti, le assunzioni sono soddisfatte e le proprietà di dominio sono soddisfatte, allora i requisiti di sistema sono soddisfatti.

$$\texttt{SOFREQ}, \texttt{ASM}, \texttt{DOM} \models \texttt{SYSREQ}$$

1.4 Categorie di requisiti

I requisiti possono essere classificati in base a diverse categorie:

- **Funzionali**: sono frasi che raccontano quelle che sono le funzionalità che il software dovrà fornire per soddisfare i bisogni del cliente. Catturando il funzionamento del sistema software.
- **Non funzionali**: sono frasi che raccontano le proprietà del sistema software che non sono legate alle funzionalità. Sono quindi legati alla qualità del software. Ad esempio, la sicurezza, la performance, la scalabilità, ...

I requisiti non funzionali, essendo un po' trasversali, potrebbero non emergere direttamente dagli stakeholder, quindi è opportuno adottare delle tassonomie per poterli identificare. Possono essere raccolti mediante domande esplicite mediante interviste, questionari, ...

1.5 Il ciclo di vita dei requisiti

1.5.1 Comprensione del dominio

Prima di tutto è necessario far un passaggio di comprensione del dominio, in modo da comunicare con i committenti e gli stakeholder con la terminologia e la comprensione del dominio

stesso. Principalmente si studia il *system as-is* e si cerca di capire come funziona il mondo prima che il software venga implementato. Studiando quindi l'organizzazione, studiando i ruoli, le prassi e individuando i problemi e le opportunità. Si identificano gli stakeholder e si cerca di capire quali sono i loro bisogni e le loro aspettative.

Ciò che viene prodotto da questa prima fase è un primo glossario della terminologia del dominio.

1.5.2 Elicitazione dei requisiti

In questa fase si può iniziare ad esplorare il problema che il software dovrà risolvere. Capendo le specifiche esigenze degli stakeholder, le opportunità tecnologiche e le limitazioni.

1.5.3 Valutazione e negoziazione

A questo punto possiamo raggiungere il primo target dei requisiti concordati con gli stakeholder. Per raggiungere questo obiettivo, ci sarà, ovviamente una prima fase di negoziazione, dove emergeranno i punti di disaccordo tra le varie parti e iniziare a mediare tra essi trovando una possibile soluzione.

Tipicamente nella fase di raccolta dei requisiti, si raccolgono anche obiettivi contrastanti tra le diverse classi di stakeholder. Ad esempio alcune responsabilità potrebbero essere volute in capo a parti diverse.

Ciò che viene prodotto da questa fase è un primo documento di specifica dei requisiti.

1.5.4 Specifica e documentazione

La fase successiva è quella di documentare tali requisiti con le varie modalità per fissarli in maniera formale. Il documento dei requisiti deve essere un documento strutturato e deve essere redatto in modo che sia comprensibile da tutti gli attori coinvolti.

Ciò che viene prodotto da questa fase è un documento di specifica dei requisiti.

1.5.5 Consolidamento dei requisiti

La fase successiva è la fase di validazione e verifica, in modo da poter capire se è corretto e completo. In modo da capire se abbiamo lacune, inconsistenze e ambiguità. Per far ciò ci sono diverse tecniche per:

- Validare i requisiti: capire se i requisiti danno risposta concreta e completa a quelle che sono le necessità degli stakeholder.
- Verificare i requisiti: capire se sono presenti inconsistenze o lacune nei requisiti.
- Dovrà essere verificato rispetto a target qualitativi.

- I problemi trovati vanno risolti.

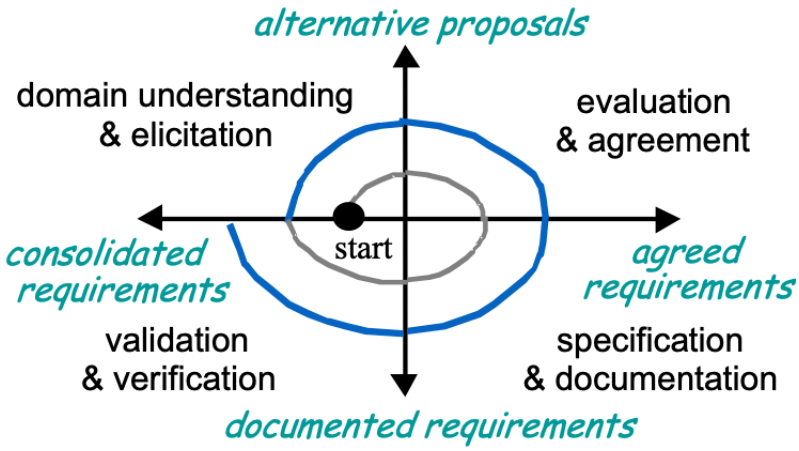
Ciò che viene prodotto da questa fase è un documento di specifica dei requisiti consolidato.

Il documento dei requisiti consolidato può fornire una base per la pianificazione del piano di test di **accettazione**. Definendo gli scenari in cui è possibile decidere se il software è completo o meno. Sono quindi **scenari** di alto livello. È possibile, inoltre, abbozzare un piano di test sviluppo.

Tale documento può essere utilizzato come base per la definizione di un contratto di fornitura del software.

La spirale dei requisiti

Nella prassi si adotta un approccio iterativo e incrementale, in modo da poter affinare i requisiti in base alle informazioni che si acquisiscono nel tempo.



Una visione evolutiva dei requisiti, che si adatta ai mutamenti dell'ambiente di lavoro, è caratteristica dello sviluppo agile del software.

1.6 Target qualitativi dei requisiti

I target di qualità, che verranno approfonditi in seguito, sono:

- **Completezza** degli obiettivi, dei requisiti e delle assunzioni
- **Coerenza** degli elementi del documento dei requisiti
- **Adeguatezza** dei requisiti, delle assunzioni e delle proprietà del dominio
- **Chiarezza** degli elementi del documento dei requisiti
- **Misurabilità** dei requisiti e delle assunzioni

- **Pertinenza** dei requisiti e delle assunzioni
- **Fattibilità** dei requisiti
- **Comprensibilità** degli elementi del documento dei requisiti
- **Buona strutturazione** del documento dei requisiti
- **Modificabilità** degli elementi del documento dei requisiti
- **Tracciabilità** degli elementi del documento dei requisiti

1.6.1 Errori comuni

Ci sono varie tipologie di errori che non devono essere commessi:

Tra questi troviamo:

- **Omissione**: l'assenza di requisiti critici rappresenta un errore grave.
- **Contraddizione**: la presenza di requisiti che si contraddicono tra loro costituisce un errore critico.
- **Inadeguatezza**: requisiti che non soddisfano le necessità o gli standard previsti sono considerati errori critici.
- **Ambiguità**: requisiti poco chiari o interpretabili in modi diversi sono errori gravi.

Altri problemi comuni includono:

- **Inmisurabilità**: requisiti che non possono essere misurati o quantificati.
- **Rumore e sovraspecificazione**: dettagli superflui o eccessivi che complicano il documento dei requisiti.
- **Infattibilità**: requisiti che non possono essere realizzati, spesso dovuti a pensieri desiderativi.
- **Inintelligibilità**: requisiti che non sono chiari o comprensibili.
- **Scarsa strutturazione**: problemi nella struttura del documento, come riferimenti anticipati o errori di organizzazione.
- **Opacità**: mancanza di trasparenza o chiarezza nei requisiti.

1.6.2 Il processo di ingegneria dei requisiti può variare a seconda del tipo di progetto

Il processo di ingegneria dei requisiti può variare significativamente in base al tipo di progetto. Alcuni dei principali fattori di variazione includono:

- **Progetti green-field** vs. **Progetti brown-field**: I progetti green-field partono da zero, mentre i progetti brown-field riguardano l'aggiornamento o l'espansione di sistemi esistenti.

- **Progetti guidati dal cliente** vs. **Progetti guidati dal mercato**: I progetti guidati dal cliente si concentrano sulle specifiche esigenze di un cliente particolare, mentre i progetti guidati dal mercato mirano a soddisfare le esigenze generali del mercato.
- **Progetti in-house** vs. **Progetti esternalizzati**: I progetti in-house sono sviluppati internamente all'organizzazione, mentre quelli esternalizzati vengono affidati a fornitori esterni.
- **Progetti single-product** vs. **Progetti product-line**: I progetti single-product riguardano un singolo prodotto, mentre i progetti product-line coinvolgono una linea di prodotti correlati.

I fattori di variazione includono:

- Il peso relativo di attività come l'elicitation, la valutazione, la documentazione, il consolidamento e l'evoluzione.
- L'intreccio tra ingegneria dei requisiti e design.
- Il peso relativo dei requisiti funzionali rispetto a quelli non funzionali.
- I tipi di stakeholder e sviluppatori coinvolti.
- Gli usi specifici del documento dei requisiti.
- L'uso di tecniche specifiche.

1.7 Ostacoli nell'ingegneria dei requisiti

Di fatto l'ingegneria dei requisiti è una disciplina molto complessa, che si trova ad affrontare diversi ostacoli. Il costo di identificare e correggere errori nei requisiti aumenta con il tempo, quindi è importante riuscire a identificarli il prima possibile. I costi in fasi avanzate possono arrivare anche a 200 volte il costo di identificazione in fase iniziale.

1.7.1 Ostacoli alla buona pratica dell'ingegneria dei requisiti

Ci sono diversi ostacoli che possono impedire una buona pratica dell'ingegneria dei requisiti. Tra questi troviamo:

- **Gli sforzi nell'ingegneria dei requisiti sono spesso spesi senza la garanzia che il contratto del progetto venga concluso**: Ciò significa che una quantità significativa di lavoro può essere svolta senza avere la certezza che il progetto andrà effettivamente avanti.
- **Pressione su scadenze strette, costi a breve termine e aggiornamenti tecnologici**: Le tempistiche ridotte, i vincoli economici e la necessità di mantenere il passo con la tecnologia possono mettere sotto pressione il processo di ingegneria dei requisiti, portando a compromessi nella qualità.
- **Scarsa disponibilità di studi sull'economia dell'ingegneria dei requisiti**: Esiste una mancanza di dati quantitativi sui benefici e sui risparmi dei costi derivanti

dall'ingegneria dei requisiti, rendendo difficile misurare il progresso rispetto alle fasi di progettazione e implementazione.

  - Mancanza di dati quantitativi sui benefici e sui risparmi dei costi dell'ingegneria dei requisiti.
  - Il progresso nel processo di ingegneria dei requisiti è più difficile da misurare rispetto alla progettazione e all'implementazione.

- **I documenti dei requisiti sono talvolta percepiti come**:
  - Grandi, complessi e rapidamente obsoleti.
  - Troppo distanti dal prodotto eseguibile che i clienti stanno pagando.

1.7.2 Sviluppo Agile e Ingegneria dei Requisiti

Lo sviluppo agile può superare alcuni degli ostacoli associati all'ingegneria dei requisiti. Alcuni dei modi in cui lo sviluppo agile riesce a farlo includono:

- **Un maggiore sviluppo agile può superare alcuni ostacoli**:
  - Fornendo in modo continuo e precoce funzionalità di valore per il cliente.
  - Riducendo la distanza tra i requisiti e il codice.
- **Cicli brevi di ingegneria dei requisiti nel processo a spirale, ciascuno seguito direttamente da un breve ciclo di implementazione**:
  - Gli incrementi funzionali utili vengono ricavati direttamente dall'utente.
  - Le fasi di valutazione, specifica e consolidamento sono spesso abbreviate (ad esempio, la specifica equivale a un caso di test sull'implementazione).
  - Gli incrementi vengono implementati e testati da un piccolo team nello stesso luogo, vicino all'utente per un feedback immediato, utilizzando regole rigorose.

1.7.3 Assunzioni forti per il successo dell'agilità

Perché l'agilità abbia successo, sono necessarie alcune assunzioni forti. Queste includono:

- **Tutti i ruoli degli stakeholder sono riducibili a un singolo ruolo**: Questo implica che le responsabilità e i compiti possono essere gestiti da un'unica figura.
- **Il progetto è sufficientemente piccolo da essere assegnabile a un singolo team di piccole dimensioni, localizzato in un unico luogo**: Questo team include programmatori, tester e manutentori.
- **L'utente può interagire prontamente ed efficacemente**: La comunicazione con l'utente deve essere rapida e produttiva.

- **La funzionalità può essere fornita rapidamente, costantemente e in modo incrementale, dalle parti essenziali a quelle meno importanti**: Non è necessaria la prioritizzazione delle funzionalità.
- **Gli aspetti non funzionali, le assunzioni sull'ambiente, gli obiettivi, le opzioni alternative e i rischi possono ricevere poca attenzione**: Questi elementi sono considerati meno critici.
- **Poca documentazione è richiesta per la coordinazione del lavoro e la manutenzione del prodotto**: La precisione dei requisiti non è essenziale; la verifica prima della codifica è meno importante di un rilascio anticipato.
- **Le modifiche ai requisiti non richiedono probabilmente un rifacimento significativo del codice**: Le modifiche ai requisiti sono gestibili senza necessità di grandi ristrutturazioni del codice.

Maggiore/minore agilità è raggiungibile con un minore/maggiore peso nelle fasi di elicitazione, valutazione, documentazione e consolidamento dei cicli di ingegneria dei requisiti.

Capitolo 2

Elicitazione dei requisiti

L'obiettivo comprendere il dominio in cui stiamo lavorando e raccogliere e capire i requisiti del software che dovrà essere implementato.

2.1 Acquisizione della conoscenza

L'acquisizione della conoscenza è una fase cruciale nel processo di ingegneria dei requisiti e include diverse attività importanti.

In primo luogo, è essenziale studiare il sistema attuale, noto come sistema *as-is*. Questo studio comprende la comprensione dell'organizzazione aziendale (*struttura, dipendenze, obiettivi strategici, politiche, flussi di lavoro e procedure operative*) e del dominio applicativo (*concetti, obiettivi, compiti, vincoli e regolamenti*). Inoltre, è necessario analizzare i problemi esistenti nel sistema attuale, identificandone sintomi, cause e conseguenze.

Un'altra attività chiave è l'analisi delle opportunità tecnologiche e delle nuove condizioni di mercato, per valutare le possibilità offerte dalle nuove tecnologie e comprendere come i cambiamenti nelle condizioni di mercato possano influenzare il sistema.

Identificare gli stakeholder del sistema è fondamentale. Gli stakeholder sono tutte le parti interessate che hanno un'influenza o sono influenzate dal sistema. Comprendere le loro esigenze e aspettative è vitale per il successo del progetto.

L'identificazione degli obiettivi di miglioramento per il sistema futuro, noto come sistema *to-be*, è un'altra attività importante. Questo include l'analisi dei vincoli organizzativi e tecnici, l'esplorazione di opzioni alternative, la definizione delle responsabilità e lo sviluppo di scenari ipotetici di interazione tra software e ambiente. Infine, è cruciale stabilire i requisiti specifici per il software e fare assunzioni sull'ambiente operativo.

Queste attività forniscono una base solida per il processo di ingegneria dei requisiti, assicurando che il sistema sviluppato soddisfi le esigenze degli stakeholder e funzioni efficacemente nel suo ambiente operativo.

Ci sono sostanzialmente due macro-approcci per l'acquisizione della conoscenza:

- **Artefact-driven**: Questo approccio si basa sull'analisi di documenti, modelli e altri artefatti esistenti per comprendere il sistema attuale e identificare i requisiti del sistema futuro. È utile quando il sistema attuale è ben documentato e i requisiti del sistema futuro sono chiari.
- **Stakeholder-driven**: Questo approccio si basa sull'interazione diretta con gli stakeholder per comprendere le loro esigenze e aspettative e identificare i requisiti del sistema futuro. È utile quando il sistema attuale è poco documentato o i requisiti del sistema futuro sono incerti.

2.1.1 Analisi degli stakeholder

Gli stakeholder sono tutte le parti interessate che hanno un'influenza o sono influenzate dal sistema.

La cooperazione degli stakeholder è essenziale per il successo dell'ingegneria dei requisiti. Il processo di elicitazione de requisiti può essere visto come un apprendimento cooperativo, dove la collaborazione tra tutte le parti coinvolte porta a una migliore comprensione dei requisiti del sistema.

Per garantire una copertura adeguata e comprensiva del mondo dei problemi, è necessario selezionare un campione rappresentativo degli stakeholder. Questa selezione deve essere dinamica, adattandosi man mano che si acquisiscono nuove conoscenze.

La selezione degli stakeholder si basa su diversi criteri, tra cui:

- La posizione rilevante nell'organizzazione.
- Il ruolo nella presa di decisioni e nel raggiungimento degli accordi.
- Il tipo di conoscenza contribuita e il livello di competenza nel dominio.
- L'esposizione ai problemi percepiti.
- Gli interessi personali e i potenziali conflitti.
- L'influenza nell'accettazione del sistema.

Selezionare gli stakeholder giusti è cruciale per assicurare che tutte le prospettive rilevanti siano considerate e che il sistema finale soddisfi le esigenze di tutti gli interessati.

2.1.2 Difficoltà nell'acquisizione della conoscenza dagli stakeholder

L'acquisizione della conoscenza dagli stakeholder presenta diverse difficoltà, tra cui fonti di informazione distribuite, punti di vista conflittuali e difficoltà di accesso alle persone chiave e ai dati. Gli stakeholder possono avere background, terminologie e culture differenti, e la conoscenza tacita o le esigenze nascoste possono complicare ulteriormente il processo. Inoltre, i dettagli irrilevanti, la politica interna, la competizione e la resistenza al cambiamento possono influire negativamente.

Il turnover del personale e i cambiamenti organizzativi e delle priorità possono creare ulteriori sfide.

Per superare queste difficoltà, sono essenziali abilità di comunicazione, costruzione di relazioni di fiducia e riformulazione continua della conoscenza attraverso riunioni di revisione.

2.2 Tecniche di Elicitazione guidate dagli Artefatti

2.2.1 Studio del background

Il processo di acquisizione della conoscenza inizia con la raccolta, la lettura e la sintesi dei documenti pertinenti. Questi documenti riguardano:

- **L'organizzazione**: include organigrammi, piani aziendali, rapporti finanziari, verbali di riunioni, ecc.
- **Il dominio**: comprende libri, indagini, articoli, regolamenti e rapporti su sistemi simili nello stesso dominio.
- **Il sistema attuale (as-is)**: include flussi di lavoro documentati, procedure, regole aziendali, documenti scambiati, rapporti di difetti/reclami, richieste di modifica, ecc.

Questo studio fornisce le basi necessarie per prepararsi prima di incontrare gli stakeholder e rappresenta un prerequisito per altre tecniche. Tuttavia, un problema comune è la gestione di una grande quantità di documentazione, dettagli irrilevanti e informazioni obsolete. La soluzione è utilizzare la meta-conoscenza per selezionare le informazioni rilevanti, sapendo cosa è necessario conoscere e cosa non lo è.

2.2.2 Raccolta dei dati

La raccolta dei dati è un'attività importante che si concentra sulla raccolta di fatti e cifre non documentati. Questi dati possono includere informazioni di marketing, statistiche di utilizzo, dati sulle prestazioni e costi. La raccolta può avvenire tramite esperimenti progettati o tramite la selezione di set di dati rappresentativi da fonti disponibili, utilizzando tecniche di campionamento statistico.

Questa attività può integrare lo studio del background, fornendo ulteriori informazioni utili per l'elicitazione dei requisiti non funzionali relativi a prestazioni, usabilità e costi.

Tuttavia, ci sono alcune difficoltà nella raccolta dei dati. Ottenere dati affidabili può richiedere tempo e i dati devono essere interpretati correttamente per essere utili.

2.2.3 Questionari

L'utilizzo dei questionari è un metodo efficace per raccogliere informazioni dagli stakeholder in modo rapido, economico e a distanza. I questionari consistono in una lista di domande inviate agli stakeholder selezionati, ognuna con una lista di possibili risposte. Possono includere:

- *Domande a scelta multipla*: dove si seleziona una risposta da una lista di opzioni.

- *Domande con pesatura*: dove si chiede di attribuire un peso a una lista di affermazioni, qualitativamente (ad esempio, "alto" o "basso") o quantitativamente (percentuali), per esprimere l'importanza percepita, le preferenze, i rischi, ecc.

I questionari sono utili per ottenere rapidamente informazioni soggettive da molte persone e possono essere d'aiuto nella preparazione di interviste più focalizzate.

Tuttavia, i questionari devono essere preparati con attenzione per evitare bias multipli (*dei destinatari, dei rispondenti, delle domande, delle risposte*) e informazioni inaffidabili dovute a fraintendimenti o risposte incoerenti.

Ecco alcune linee guida per la progettazione e la validazione dei questionari:

- Selezionare un campione rappresentativo e statisticamente significativo di persone, fornendo motivazioni per rispondere.
- Verificare la copertura delle domande e delle possibili risposte.
- Assicurarsi che le domande e le formulazioni siano imparziali e non ambigue.
- Aggiungere domande ridondanti implicitamente per rilevare risposte incoerenti.
- Far controllare il questionario da una terza parte.

Pro dei questionari

- Raccolta rapida di informazioni
- Economici e facili da distribuire
- Utili per preparare interviste focalizzate

Contro dei questionari

- Possibili bias dei destinatari e rispondenti
- Rischio di fraintendimenti nelle domande e risposte
- Difficoltà nell'assicurare risposte coerenti

2.2.4 Card sorting e repertory grids

L'obiettivo del card sorting e delle repertory grids è acquisire ulteriori informazioni sui concetti già elicitati. Nel card sorting, si chiede agli stakeholder di dividere un set di carte, ognuna rappresentante un concetto, in sottogruppi basati sui loro criteri. Per ogni sottogruppo, si indaga sulle proprietà condivise utilizzate per la classificazione. Questo processo è iterativo e può essere ripetuto per nuovi raggruppamenti e proprietà.

Ad esempio, nel contesto di un sistema di pianificazione delle riunioni, le carte "Riunione" e "Partecipante" potrebbero essere raggruppate insieme, suggerendo che *i partecipanti devono essere invitati alla riunione*. Nella successiva iterazione, lo stesso raggruppamento potrebbe indicare che *i vincoli dei partecipanti per la riunione devono essere conosciuti*.

Nel repertory grid, si chiede agli stakeholder di caratterizzare un concetto attraverso attributi e intervalli di valori, formando una griglia concetto-attributo. Ad esempio, per il concetto di "Riunione", gli attributi potrebbero essere *Data* (Lun-Ven) e *Luogo* (Europa).

Il conceptual laddering, invece, richiede agli stakeholder di classificare i concetti target lungo collegamenti di classe-sottoclasse, come ad esempio `RiunioneRegolare` e `RiunioneOccasionale` come sottoclassi di `Riunione`.

Questi metodi sono semplici, economici e facili da usare per l'elicitazione rapida di informazioni mancanti, ma i risultati possono essere soggettivi, irrilevanti o inaccurati.

Pro del card sorting e repertory grids

- Semplici ed economici
- Facili da usare
- Rapida elicitazione di informazioni

Contro del card sorting e repertory grids

- Risultati possono essere soggettivi
- Possibilità di informazioni irrilevanti
- Possibili inaccuratezze nei risultati

2.2.5 Scenari e storyboard

Gli scenari e gli storyboard aiutano ad acquisire o validare informazioni attraverso esempi concreti e narrazioni. Gli scenari illustrano sequenze tipiche di interazione tra i componenti del sistema per raggiungere un obiettivo implicito, utilizzati sia per spiegare il sistema attuale (*as-is*) che per esplorare il sistema futuro (*to-be*).

Gli storyboard raccontano una storia attraverso una sequenza di istantanee, che possono essere frasi, schizzi, diapositive o immagini, e possono includere annotazioni che spiegano chi sono i partecipanti, cosa accade loro, perché accade e cosa succede in caso di eventi alternativi.

Le storyboard possono essere inoltre *passive* o *attive*. Le passive se raccontiamo la nostra idea per ricevere un feedback, quindi sono adatte per la validazione dei requisiti. Le attive vengono elaborate o completate dagli stakeholder, quindi sono adatte per l'elicitazione dei requisiti.

Gli scenari possono essere:

- *Scenario positivo*: un comportamento che il sistema dovrebbe coprire.
- *Scenario negativo*: un comportamento che il sistema dovrebbe escludere.
- *Scenario normale*: tutto procede come previsto.
- *Scenario anomalo*: una sequenza di interazione desiderata in situazioni di eccezione.

Pro degli scenari

- Esempi concreti e contro-esempi
- Stile narrativo (*recepibili da tutti*)
- Producono sequenze di animazione, possono essere tradotti in test di accettazione

Contro degli scenari

- Inerentemente parziali (*problema di copertura del test*)
- Esplosione combinatoria (*cf. tracce di programma*)
- Sovraspecificazione potenziale: sequenze non necessarie, decisioni premature.
- Possono contenere dettagli irrilevanti, granularità incompatibili tra diversi stakeholder
- Mantengono i requisiti impliciti

Nonostante ciò, sono preziosi come veicoli iniziali per l'elicitazione dei requisiti.

2.2.6 Prototipi e mock-up

L'obiettivo dei prototipi e dei mock-up è verificare l'adeguatezza dei requisiti attraverso il feedback diretto degli utenti, mostrando uno schizzo ridotto del software futuro in azione. Questo metodo si concentra su requisiti poco chiari e difficili da formulare per elicitare ulteriori dettagli.

Un prototipo è un'implementazione rapida di alcuni aspetti del sistema:

- *Prototipo funzionale*: si focalizza su requisiti funzionali specifici, come l'avvio di una riunione o la raccolta dei vincoli dei partecipanti.
- *Prototipo dell'interfaccia utente*: si concentra sulla usabilità, mostrando moduli di input-output e pattern di dialogo.

I prototipi possono essere implementati rapidamente utilizzando linguaggi di programmazione di alto livello, linguaggi di specifica eseguibile, e servizi generici.

Prototipazione dei requisiti include due approcci principali:

- *Mock-up*: il prototipo viene scartato una volta che ha soddisfatto il suo scopo di chiarire e validare i requisiti.
- *Prototipo evolutivo*: il prototipo viene trasformato e perfezionato fino a diventare parte del prodotto finale.

Pro dei prototipi e mock-up

- Forniscono un'idea concreta di come sarà il software
- Chiariscono i requisiti, elicitano quelli nascosti, migliorano l'adeguatezza, e aiutano a comprendere le implicazioni
- Utili anche per la formazione degli utenti e come stub per test di integrazione

Contro dei prototipi e mock-up

- Non coprono tutti gli aspetti del sistema
- Possono mancare funzionalità importanti
- Ignorano requisiti non funzionali rilevanti (*prestazioni, costi, ecc.*)
- Possono creare aspettative troppo alte e fuorvianti
- Codice "quick-and-dirty" difficile da riutilizzare per lo sviluppo del software
- Potenziali incongruenze tra codice modificato e requisiti documentati

2.2.7 Riutilizzo della Conoscenza

L'obiettivo del riutilizzo della conoscenza è accelerare l'elicitazione attraverso il riutilizzo delle esperienze con sistemi simili o domini analoghi. Questo include la conoscenza di organizzazioni simili, domini, e mondi problematici, come requisiti, assunzioni, proprietà del dominio, ecc.

La trasposizione di conoscenze da un contesto all'altro avviene mediante tre passaggi:

- *Istanziazione*: quello viene fatto è un esempio di ciò che è stato visto.
- *Specializzazione*: concretizzazione dell'astrazione dell'istanziazione.
- *Riformulazione*: adattamento del concetto alla nuova situazione.

Riutilizzo della conoscenza indipendente dal dominio: tassonomie

Spesso è utile adottare una tassonomia in modo da utilizzare esperienza già maturata. Tale tassonomia può essere creata mediante l'esperienza in vari contesti in modo da facilitare il riutilizzo della conoscenza e la velocità nella raccolta dei requisiti.

Riutilizzo della conoscenza dipendente dal dominio: meta-modelli

Come informazioni di riuso è possibile riutilizzare dei meta-modelli elaborati, un meta-modello è un modello di modelli, cioè un modello che descrive come costruire altri modelli. Questo permette di avere una visione più astratta e di poter riutilizzare la conoscenza in modo più efficace.

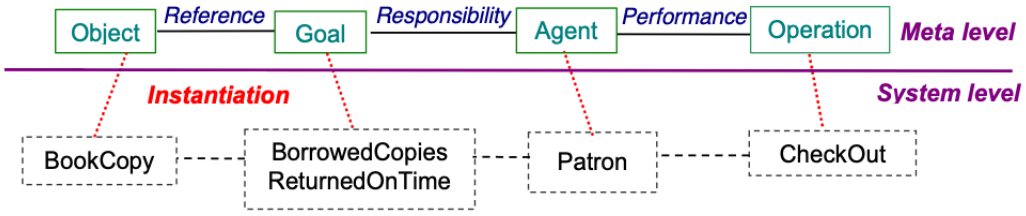


Figura 2.2.1: Esempio di meta-modello

Riutilizzo della conoscenza specifica del dominio

È possibile inoltre utilizzare informazione specifica del dominio astratto, sempre avendo come riferimento il modello, sappiamo che ci sono delle risorse che hanno vincoli di utilizzo limitato, quindi possiamo utilizzare queste informazioni per capire come si comporterà il sistema.

*Un **utente** non deve **usare** più di X **risorse** contemporaneamente*

*Un **patron** non deve **prendere in prestito** più di X **copie di libri** contemporaneamente*

Questa conoscenza piò essere personalizzata in diversi modi:

- Lo stesso dominio astratto può avere molte specializzazioni concrete.
- Lo stesso dominio con vincoli diversi può avere molte specializzazioni astratte.
- Di fatto posso riusare molteplici strutture.

Pro del riutilizzo della conoscenza

- Analisti esperti riutilizzano la conoscenza acquisita naturalmente
- Guida molto nell'elicitazione dei requisiti.
- Si eredita la struttura e uindi anche la qualità del dominio che stiamo utilizzando.
- È efficace nel completamento del documento dei requisiti.

Contro del riutilizzo della conoscenza

- Possiamo utilizzare la conoscenza astratta solo quando i domini sono simili.
- Definire i domini astratti è difficile.
- C'è effort aggiuntivo per la creazione di meta-modelli.
- Quando il dominio si avvicina molto richiede un adattamento complesso.

2.3 Tecniche di Elicitazione guidate dagli Stakeholder

2.3.1 Interviste

L'intervista è uno strumento fondamentale per l'acquisizione di conoscenza, permettendo di catturare informazioni direttamente dai portatori di conoscenza come esperti di dominio, manager e utenti finali. L'obiettivo principale dell'intervista è di elicitare informazioni che non

emergerebbero tramite altri metodi, focalizzandosi sulle percezioni personali e sulle esperienze dei partecipanti.

- *Preparazione dell'intervista*: L'intervistatore deve selezionare accuratamente i partecipanti in base alla loro esperienza e al ruolo ricoperto, preparare in anticipo le domande e organizzare incontri che facilitino una comunicazione aperta e onesta.
- *Conduzione dell'intervista*: La fase di intervista può includere sia parti strutturate, con domande specifiche, sia parti non strutturate, che permettono di esplorare nuove aree attraverso una discussione libera.

Efficacia dell'intervista dipende dalla capacità dell'intervistatore di gestire la conversazione, mantenendo il controllo dell'intervista pur permettendo all'intervistato di esprimersi liberamente. Le interviste possono svelare dettagli critici sul funzionamento interno del sistema che non sono immediatamente evidenti attraverso l'analisi documentale o osservazione diretta.

Pro delle interviste

- Forniscono intuizioni profonde e dirette dal campo
- Permettono la scoperta di requisiti impliciti o non documentati
- Aiutano a comprendere meglio le necessità e le preoccupazioni degli utenti

Contro delle interviste

- Possono essere influenzate dalla soggettività degli intervistati
- Richiedono tempo e possono essere costose in termini di organizzazione e analisi
- Le informazioni raccolte possono essere inconsistenti tra diversi intervistati

L'approccio misto, combinando elementi di interviste strutturate e non strutturate, è spesso il più efficace, permettendo di ottenere una copertura ampia delle informazioni pur mantenendo la capacità di adattarsi a nuove scoperte durante l'intervista stessa.

2.3.2 Osservazione e Studi Etnografici

L'osservazione e gli studi etnografici sono metodologie essenziali per comprendere i compiti all'interno dei sistemi esistenti, facilitando l'osservazione diretta delle interazioni umane e delle pratiche lavorative.

- *Osservazione passiva*: Consiste nel non interferire con i soggetti, registrando le attività tramite note o video e analizzando i dati raccolti senza influenzare i comportamenti.
- *Osservazione attiva*: L'osservatore partecipa direttamente alle attività, integrandosi nel gruppo per ottenere una comprensione interna delle dinamiche del compito.
- *Studi etnografici*: Effettuati su periodi prolungati, mirano a scoprire le caratteristiche emergenti di un gruppo sociale, evidenziando come i compiti vengano eseguiti e quali sono le norme culturali influenti.

Questi metodi possono offrire spunti su come le persone lavorano veramente, rispetto a come dicono di lavorare o a come si crede lavorino.

Vantaggi dell'osservazione e degli studi etnografici

- Forniscono una visione approfondita delle conoscenze tacite e dei modelli mentali.
- Identificano problemi nascosti e pratiche non documentate.
- Consentono un'analisi contestuale che è fondamentale per soluzioni adatte culturalmente.

Svantaggi dell'osservazione e degli studi etnografici

- Processi lunghi e costosi.
- Rischio di inesattezze dovute a comportamenti alterati in presenza di osservatori.
- Complessità nella gestione e interpretazione dei grandi volumi di dati.

Queste tecniche, quando abbinate ad altri metodi come le interviste, possono offrire una visione più completa del sistema studiato.

2.3.3 Sessioni di Gruppo

Le sessioni di gruppo, sia strutturate che non, sono un'efficace metodologia per la raccolta di percezioni, giudizi e idee attraverso l'interazione tra partecipanti diversi. Le sessioni possono variare da incontri focalizzati su obiettivi specifici a brainstorming aperti.

- *Sessioni di gruppo strutturate*: Si svolgono in workshop di gruppo che possono durare alcuni giorni, con ruoli ben definiti per ciascun partecipante (leader, moderatore, manager, utente, sviluppatore, ecc.). Utilizzano strumenti come supporti visivi e grafici per stimolare la discussione e documentare i risultati.
- *Sessioni di gruppo non strutturate (brainstorming)*: In questi incontri, i partecipanti hanno ruoli meno definiti e il processo si divide in generazione di idee senza censura seguita da una valutazione collettiva secondo criteri prestabiliti come valore, costo e fattibilità.

Le sessioni di gruppo possono portare alla luce aspetti nascosti del sistema attuale o futuro e favorire una più ampia esplorazione di problemi e soluzioni, stimolando l'invenzione e la collaborazione tra i partecipanti.

Vantaggi delle sessioni di gruppo

- Potenziale di esplorazione più ampia e creativa dei problemi.
- Possibilità di scoprire soluzioni innovative attraverso la sinergia del gruppo.
- Migliore risoluzione dei conflitti grazie alla collaborazione.

Svantaggi delle sessioni di gruppo

- Richiedono tempo e possono essere difficili da organizzare per persone molto impegnate.
- Rischio di dominanza di alcune personalità che possono influenzare l'intero gruppo.
- Possibili discussioni prolisse senza risultati concreti, copertura superficiale di questioni tecniche.

Capitolo 3

Validazione

L'obiettivo di questa fase è quello di prendere delle decisioni in caso di inconsistenze o quali decisioni prendere in maniera negoziata.

3.1 Decisione Basata sulla Negoziazione

La decisione basata sulla negoziazione nel processo di ingegneria dei sistemi comporta diversi passaggi per affrontare e risolvere varie problematiche che emergono durante il processo di sviluppo. Di seguito, discutiamo i principali componenti evidenziati nel processo decisionale:

- **Identificazione e Risoluzione delle inconsistenze**: Questo passaggio implica la comprensione e la risoluzione dei punti di vista degli stakeholder in conflitto e delle richieste non funzionali per raggiungere un consenso.
- **Identificazione, valutazione e risoluzione dei Rischi di sistema**: Questo passaggio è fondamentale per garantire che il sistema soddisfi tutti gli obiettivi critici di sicurezza e protezione. Include la revisione dei requisiti per sviluppare un sistema più robusto.
- **Confronto delle opzioni alternative**: Si considerano varie opzioni per raggiungere gli obiettivi, assegnare responsabilità e risolvere conflitti e rischi, il che aiuta nella selezione delle soluzioni più appropriate.
- **Prioritizzazione dei requisiti**: La prioritizzazione dei requisiti è essenziale per risolvere i conflitti, aderire ai vincoli di budget e di programma, e supportare lo sviluppo incrementale.

3.2 Gestione delle Inconsistenze

L'inconsistenza rappresenta una violazione delle regole di coerenza tra vari elementi e è un fenomeno altamente frequente nell'ingegneria dei requisiti. Queste incongruenze possono sorgere da diverse prospettive:

- **Inter-punti di vista**: Ogni stakeholder ha le proprie priorità e preoccupazioni, che possono variare significativamente. Per esempio, gli esperti di dominio possono avere requisiti che contrastano con quelli del reparto marketing.
- **Intra-punto di vista**: Possono emergere conflitti anche all'interno dello stesso punto di vista, come tra requisiti di qualità che si contraddicono (*ad esempio, sicurezza vs. usabilità*).

l'inconsistenza può rappresentare anche un valore, permettendo di avere due punti di vista differenti circa uno stesso requisito. Facendo quindi scoprire nuovi requisiti.

Le inconsistenze devono essere rilevate e risolte in modo tempestivo:

- **Non troppo presto**: perché potremmo avere una visione parziale dei requisiti.
- **Non troppo tardi**: perché va risolta prima della progettazione e la realizzazione del sistema.

3.2.1 Tipi di Inconsistenza nell'Ingegneria dei Requisiti

Nell'ambito dell'ingegneria dei requisiti, si possono identificare diversi tipi di inconsistenze, ciascuno con specifiche sfide da gestire:

Clash di Terminologia

Questo tipo di inconsistenza si verifica quando lo stesso concetto viene nominato in modi diversi nelle diverse dichiarazioni. Ad esempio, nella gestione di una biblioteca, i termini *borrower* e *patron* possono essere utilizzati per indicare la stessa figura.

Clash di Designazione

Si verifica un clash di designazione quando lo stesso nome viene utilizzato per concetti differenti. Ad esempio, il termine *user* potrebbe riferirsi sia a un *library user* che a un *library software user*, creando confusione.

Clash Strutturale

Questo tipo di inconsistenza si verifica quando lo stesso concetto è strutturato in modo diverso in dichiarazioni differenti. Un esempio è la *latest return date*, che può essere interpretata sia come un punto temporale specifico (*es. Venerdì alle 17:00*) che come un intervallo di tempo (*es. il Venerdì*).

Conflitto Forte

Un conflitto forte si verifica quando le dichiarazioni sono logicamente inconsistenti e non possono essere soddisfatte contemporaneamente, ad esempio, "le restrizioni dei partecipanti non possono essere divulgate a nessuno" contro "l'iniziatore dell'incontro dovrebbe conoscere le restrizioni dei partecipanti".

Conflitto Debole (*Divergenza*)

I conflitti deboli avvengono quando le dichiarazioni non possono essere soddisfatte insieme sotto certe condizioni limite. Questi sono molto più frequenti nell'ingegneria dei requisiti. Ad esempio, da un punto di vista del personale, "i lettori devono restituire i libri presi in prestito entro due settimane" può confliggere con la prospettiva del lettore che "i lettori possono tenere i libri presi in prestito finché necessario", se il lettore ha bisogno del libro per più di due settimane.

Questi tipi di inconsistenze richiedono attenzione e strategie specifiche per la loro risoluzione, per garantire che il processo di sviluppo dei requisiti sia efficace e che il prodotto finale soddisfi tutte le parti interessate.

3.2.2 Gestione delle Inconsistenze

La gestione efficace delle inconsistenze è fondamentale nell'ingegneria dei requisiti. Ecco alcune strategie per affrontare questi problemi:

Gestione dei Conflitti di Terminologia, Designazione e Struttura

Per mitigare i conflitti di terminologia, designazione e struttura, è utile sviluppare un **glossario** condiviso di termini durante la fase di raccolta dei requisiti. Questo glossario dovrebbe includere:

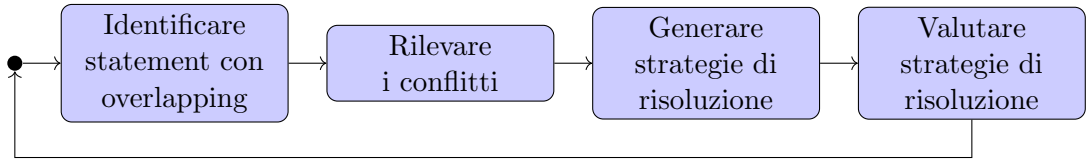
- Definizioni chiare per ogni termine rilevante.
- Sinonimi accettati, se necessario, per garantire coerenza nella comunicazione tra le parti interessate.

Conflitti Deboli e Forti

I conflitti, sia deboli che forti, possono essere radicati in obiettivi personali contrastanti degli stakeholder:

- **Radice dei Conflitti**: Spesso, questi conflitti sono profondamente radicati negli obiettivi personali degli stakeholder e devono essere affrontati a questo livello per poi essere propagati al livello dei requisiti.
- **Conflitti Funzionali e Non-Funzionali**: Sono frequenti in questioni non-funzionali come il bilanciamento tra prestazione e sicurezza, o tra riservatezza e consapevolezza. L'esplorazione di compromessi preferenziali è essenziale.

3.2.3 Come risolvere le Inconsistenze

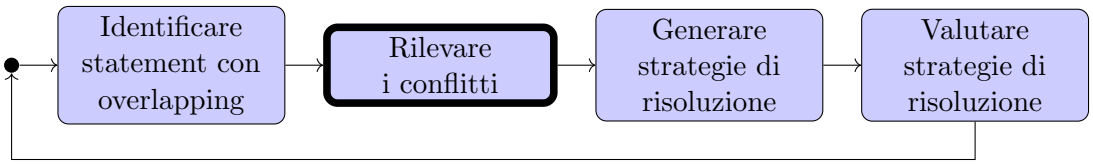


L'**Overlap** si riferisce al riferimento a termini o fenomeni comuni, che sono una precondizione per l'identificazione di dichiarazioni conflittuali. Questo include attività come la raccolta dei vincoli degli incontri e la determinazione dei programmi.

La **Rilevazione dei Conflitti** può avvenire attraverso vari metodi:

- **Informalmente**: Utilizzando conoscenze generali ed esperienza per identificare potenziali conflitti senza metodi formali.
- **Usando euristiche su categorie di requisiti in conflitto**, come la verifica dei conflitti tra "requisiti di informazione e requisiti di riservatezza su oggetti correlati" e la valutazione dei requisiti su "quantità in aumento e diminuzione correlate".
- **Usando modelli di conflitto**: Identificazione di scenari o strutture ripetuti che tipicamente portano a conflitti nei requisiti.
- **Formalmente (tecniche di dimostrazione teoremi)**: Applicazione di metodi matematici o logici per provare se certe combinazioni di requisiti possono coesistere senza conflitti.

Importanza della Documentazione dei Conflitti



Per una successiva risoluzione e analisi dell'impatto, è importante documentare i conflitti rilevati. Utilizzando strumenti di documentazione e query che sfruttano i collegamenti *Conflict* registrati nel database dei requisiti, è possibile tracciare e gestire efficacemente le dichiarazioni che presentano molteplici conflitti.

Matrice di Interazione

Una matrice di interazione è uno strumento utile per visualizzare le relazioni tra i requisiti e identificare i conflitti. Questa struttura consente di vedere facilmente quali requisiti si sovrappongono o entrano in conflitto, come illustrato nella matrice seguente:

Statement	**S1**	**S2**	**S3**	**S4**	**Totale**
S1	0	1000	1	1	1002
S2	1000	0	0	0	1000
S3	1	0	0	1	2
S4	1	0	1	0	2
Totale	1002	1000	2	2	2006

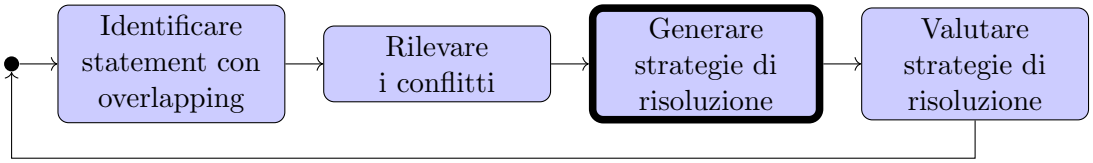
Nella matrice:

- **0** indica nessuna sovrapposizione,

- **1** indica un conflitto,
- **1000** indica l'assenza di conflitti.

La formula per calcolare il numero di conflitti per *S1* è data dal resto della divisione di 1002 per 1000, mentre il numero di sovrapposizioni non conflittuali è il quoziente di 1002 diviso 1000.

Generazione di Strategie di Risoluzione

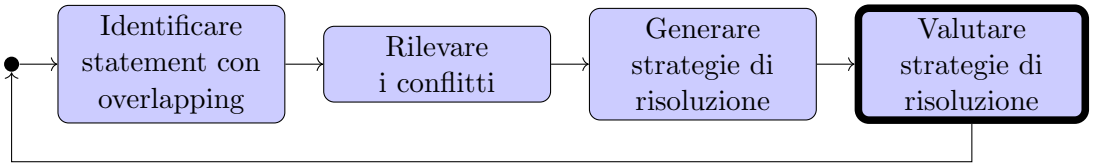


Bisogna per prima cosa identificare le possibili soluzioni che si possono avere a disposizione. Nella fase successiva cerco di confrontare le soluzioni e valutare quale sia la migliore. Per cercare dei candidati per la risoluzione dei conflitti, si possono utilizzare altre tecniche di elicitazione dei requisiti, come interviste, questionari, o altro; oppure utilizzo delle strategie di risoluzione dei conflitti tra cui:

- **Evitare le condizioni al limite:** ad esempio, "*Mantenere le copie dei libri molto richiesti non prenotabili*".
- **Ripristinare le affermazioni in conflitto:** ad esempio, "*Copia restituita entro 2 settimane e poi presa in prestito di nuovo*".
- **Indebolire le affermazioni in conflitto:** ad esempio, "*Copia restituita entro 2 settimane a meno di un permesso esplicito*".
- **Omettere le affermazioni di minore priorità:** questo implica scartare le affermazioni meno critiche per risolvere il conflitto.
- **Specializzare la fonte o il target del conflitto:** ad esempio, "*Stato del prestito del libro noto solo agli utenti dello staff*".

Queste strategie aiutano a trasformare le affermazioni in conflitto o gli oggetti coinvolti, oppure a introdurre nuovi requisiti per facilitare la risoluzione dei conflitti.

Valutazione delle Strategie di Risoluzione



La valutazione delle strategie per risolvere i conflitti si basa su criteri ben definiti che mirano a ottimizzare il processo decisionale e garantire la coerenza con gli obiettivi del progetto:

- **Contributo agli obiettivi non funzionali critici:** Le strategie di risoluzione vengono valutate in base al loro impatto sui requisiti non funzionali considerati critici per il successo del sistema.
- **Contributo alla risoluzione di altri conflitti e rischi:** Oltre a risolvere il conflitto specifico, una strategia efficace dovrebbe facilitare la gestione di altri conflitti potenziali e mitigare i rischi associati.

Questi criteri sono fondamentali per scegliere la risoluzione che non solo risolve il conflitto immediato ma promuove anche la stabilità e l'integrità del sistema nel lungo termine. Ulteriori dettagli sulla selezione e l'implementazione di queste strategie possono essere trovati nella sezione "Valutazione delle opzioni alternative".

3.3 Analisi dei rischi

Rischio

Un **rischio** è un fattore incerto il cui verificarsi può comportare la perdita di soddisfazione di un obiettivo corrispondente.

Esempi includono:

- Un passeggero che forza l'apertura delle porte mentre il treno è in movimento.
- Un partecipante a una riunione che non controlla regolarmente la posta elettronica.

Un rischio è caratterizzato da:

- Una probabilità di occorrenza.
- Una o più conseguenze indesiderate, come i passeggeri che cadono dal treno in movimento con le porte aperte.

Ogni conseguenza del rischio ha:

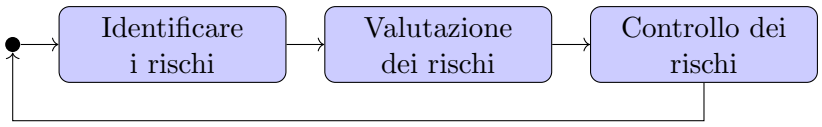
- Una probabilità di occorrenza se il rischio si verifica.
- Una gravità, che è il grado di perdita di soddisfazione dell'obiettivo.

3.3.1 Tipologie di rischi

I rischi possono essere categorizzati in:

- **Rischi legati al prodotto:** Impattano sugli obiettivi funzionali o non funzionali del sistema. Esempi includono minacce alla sicurezza e pericoli per la sicurezza.
- **Rischi legati al processo:** Hanno un impatto sugli obiettivi di sviluppo, come ritardi nella consegna o superamenti di costo. Un esempio comune è il turnover del personale.

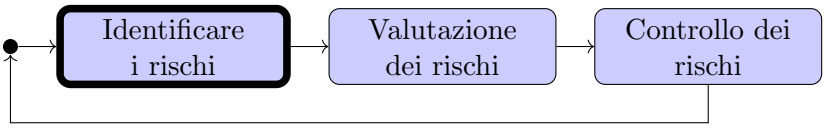
3.3.2 Come affrontare i rischi



L'identificazione del rischio è un processo iterativo, infatti le azioni che prendiamo per controllare i rischi potrebbero dettare nuovi requisiti, ponendo nuovi rischi.

La gestione superficiale è una delle principali cause di fallimento di progetti software.

Identificazione dei Rischi



L'identificazione dei rischi è un processo essenziale che si basa sull'uso di liste di controllo specifiche per ciascuna categoria di progetto, al fine di personalizzare l'approccio al contesto specifico del progetto. Queste liste sono importanti per identificare i rischi potenziali associati alle diverse categorie di requisiti.

- **Rischi Legati al Prodotto**: derivano dalla possibile insoddisfazione nei requisiti funzionali o di qualità:
  - **Inesattezze informative**, come stime errate della velocità o della posizione dei treni.
  - **Indisponibilità o inutilizzabilità** delle informazioni.
  - **Tempi di risposta scarsi** o **carenze durante picchi di carico**, che possono compromettere l'efficienza del servizio.
- **Rischi Legati al Processo**: includono quelli che possono influenzare negativamente il raggiungimento degli obiettivi di sviluppo:
  - **Volatilità dei requisiti**, che può portare a frequenti cambiamenti e riallineamenti.
  - **Carenze di personale** o **dipendenze da fonti esterne**, che possono ritardare il progresso del progetto.
  - **Programmazioni o budget non realistici**, aumentando il rischio di superamenti di costi.
  - **Gestione inefficace dei rischi**, come può avvenire con un team di sviluppatori inesperto.

Questi rischi devono essere sistematicamente identificati e gestiti attraverso l'uso di strategie di mitigazione adeguate, affinché il progetto possa procedere senza intoppi e con il minor numero di sorprese possibili.

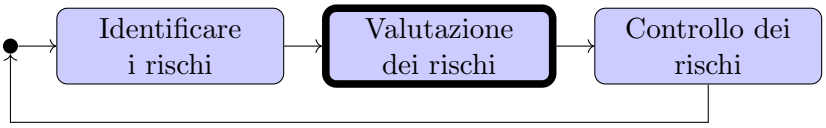
Per i rischi legati al prodotto, tipicamente analizziamo il *system to-be* e ci focalizziamo sulle sotto-componenti per capire in quanti modi ogni singolo componente potrebbe fallire, comprendendo inoltre come ogni componente possa fallire. Possiamo procedere spezzando ulteriormente i componenti in sotto-componenti, fino a quando non siamo in grado di identificare i rischi.

Per identificare i rischi è possibile inoltre utilizzare il **risk tree**, che è un albero che parte da un nodo radice e si dirama in nodi figli, che rappresentano i sotto-rischi. Questo albero può essere utilizzato per identificare i rischi e per capire come questi si propagano.

Quando ho un risk tree, posso analizzare il *cut set*, ovvero la minima decomposizione delle foglie dell'albero che mi permette di analizzare il rischio ed essi stessi possono essere organizzati in un albero.

Per identificare i rischi, oltre a queste analisi, posso dotarmi di scenari e chiedendo agli stakeholder, sessioni di gruppo e il riuso della conoscenza sulla base di esperienze passate.

Valutazione dei Rischi



La valutazione dei rischi è importante per determinare la probabilità e la gravità delle conseguenze dei rischi identificati. Questo processo è guidato da:

- **Stima Qualitativa**: Utilizzo di stime qualitative per la probabilità e la gravità:
  - *Probabilità*: ad esempio, molto probabile, probabile, possibile, improbabile.
  - *Gravità*: ad esempio, catastrofica, grave, alta, moderata.
- **Tavola di Consequenzialità dei Rischi**: Ogni rischio viene valutato in base a una tabella che incrocia la probabilità del suo verificarsi con la gravità delle sue conseguenze.
- **Confronto e Prioritizzazione dei Rischi**: I rischi vengono confrontati e prioritizzati in base ai loro livelli di severità per focalizzarsi sul controllo dei rischi ad alta priorità.

Questi passaggi sono essenziali per sviluppare strategie efficaci di gestione e mitigazione dei rischi, assicurando così che i rischi ad alta priorità siano gestiti con le risorse adeguate e nei tempi appropriati.

Per la valutazione dei rischi possiamo dotarci di una tabella di valutazione dei rischi:

Conseguenze	Probabile	Possibile	Improbabile
Perdita di vite	Catastrofico	Catastrofico	Grave
Ferite gravi	Catastrofico	Grave	Alta
Danni al treno	Alto	Moderato	Basso
Riduzione passeggeri	Alto	Alto	Basso
Danno reputazionale	Moderato	Basso	Basso

Pro

Questo approccio è **facile da utilizzare** e fornisce stime rapide.

Contro

Le conclusioni sono **limitate**: le stime sono grossolane e soggettive, e non considerano la probabilità complessiva delle conseguenze.

Una valutazione del rischio più quantitativo può essere fatto mediante l'ausilio della probabilità e per la scala di severità sempre con un range di valori. Questo permette di associare dei valori metrici quantitativi per calcolare delle metriche derivate.

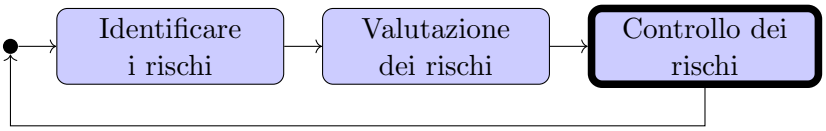
Pro

Questo approccio è **più preciso** e **più accurato**.

Contro

Tale approccio rimane soggettivo, di fatto questi valori dovrebbero essere negoziati con gli stakeholder, che conoscono meglio il contesto.

Controllo dei Rischi



L'obiettivo principale nella gestione dei rischi è ridurre l'esposizione ai rischi più critici attraverso l'implementazione di contromisure efficaci. Queste contromisure non solo mitigano i rischi esistenti ma possono anche portare all'emergere di nuovi requisiti per affrontare condizioni o sfide non precedentemente identificate.

L'adozione di tali misure richiede un'analisi accurata e una valutazione continua per assicurare che i rischi vengano gestiti in modo proattivo e che le soluzioni implementate migliorino effettivamente la robustezza del sistema o del processo.

3.3.3 Documentazione dei Rischi

La documentazione accurata dei rischi è fondamentale per supportare l'evoluzione del sistema e garantire che le contromisure siano adeguatamente pianificate e implementate. Per ogni rischio identificato, è essenziale documentare:

- **Condizioni o Eventi di Occorrenza:** Specificare le circostanze sotto le quali il rischio può manifestarsi.
- **Probabilità Stimata:** Valutare la frequenza con cui il rischio può verificarsi.
- **Cause e Conseguenze Possibili:** Analizzare e registrare le potenziali cause dei rischi e le loro possibili conseguenze.
- **Probabilità e Gravità di Ogni Conseguenza:** Stabilire quanto spesso le conseguenze possono verificarsi e quanto gravi possono essere
- **Contromisure Identificate e Leverage di Riduzione del Rischio:** Elencare le contromisure proposte e valutare la loro efficacia nel ridurre la probabilità o l'impatto del rischio.
- **Contromisure Selezionate:** Indicare quali contromisure sono state effettivamente scelte per l'implementazione.

Questa documentazione dettagliata forma la base per un **albero dei rischi annotato**, che serve come strumento visivo e funzionale per comprendere e gestire i rischi durante tutto il ciclo di vita del progetto.

3.3.4 Defect Detection Prevention

Il **Defect Detection Prevention** (`DDP`) è una tecnica e uno strumento sviluppati presso la `NASA` per fornire supporto quantitativo ai cicli di *Identificazione-Valutazione-Controlla* dei rischi. Questo approccio si articola in tre fasi principali:

Il `DDP` è un metodo per identificare i rischi e le opportunità, valutarli e controllarli. Questo metodo si basa su tre fasi principali:

1. **Elaborazione della Matrice di Impatto del Rischio:** Questa fase prevede la creazione di una matrice che valuta l'impatto di ciascun rischio identificato, facilitando una comprensione quantitativa delle potenziali conseguenze.
2. **Elaborazione della Matrice di Efficacia delle Contromisure:** Successivamente, viene sviluppata una matrice che valuta l'efficacia delle varie contromisure proposte contro i rischi identificati.
3. **Determinazione del Bilanciamento Ottimale tra Riduzione del Rischio e Costo delle Contromisure:** L'ultima fase del processo mira a trovare un equilibrio ottimale tra il grado di riduzione del rischio ottenuto e il costo delle contromisure implementate.

Questo metodo permette di approcciare la gestione dei rischi in maniera sistematica e quantificabile, assicurando che le decisioni prese siano basate su dati solidi e analisi approfondite.

3.4 Confronto delle opzioni alternative

l processo di Ingegneria dei Requisiti solleva diverse opzioni alternative per soddisfare gli obiettivi di un sistema, risolvere i conflitti e ridurre i rischi. La selezione delle alternative preferite richiede una negoziazione basata su criteri ben definiti.

3.4.1 Valutazione Qualitativa e Quantitativa

- **Valutazione Qualitativa:** Si determina il contributo di ciascuna opzione ai requisiti non funzionali (NFRs), usando etichette qualitative come molto positivo (++), positivo (+), negativo (-) e molto negativo (–). Ad esempio, nella programmazione di riunioni, le opzioni possono variare nella loro capacità di fornire risposte rapide o minimizzare gli inconvenienti.
- **Valutazione Quantitativa:** Si costruisce una matrice ponderata per valutare ogni opzione secondo diversi criteri, basati sull'importanza relativa di ciascun criterio. Il punteggio complessivo di ogni opzione è calcolato come:

$$\text{Punteggio Totale}(opt) = \sum_{\text{crit}} (\text{Punteggio}(opt, \text{crit}) \times \text{Peso}(\text{crit}))$$

  dove i punteggi vanno da 0 a 1, indicando la percentuale di soddisfacimento del criterio.

3.4.2 Criteri di Valutazione

Per selezionare le alternative preferenziali, si concorda sui seguenti criteri:

- Contributo ai requisiti non funzionali critici.
- Contributo alla risoluzione di altri rischi.
- Costo-efficacia, misurato attraverso il leverage di riduzione del rischio.

Questo approccio consente di prendere decisioni informate durante il processo ingegneria dei requisiti, assicurando che le soluzioni scelte offrano il miglior equilibrio tra benefici e costi.

3.5 Prioritizzazione dei requisiti

La prioritizzazione dei requisiti è essenziale per affrontare conflitti di risorse, sviluppo incrementale e per adattarsi a problemi imprevisti durante lo sviluppo del progetto. Qui sono illustrati i principi e le tecniche per una prioritizzazione efficace dei requisiti.

3.5.1 Principi di Prioritizzazione dei Requisiti

- Prioritizzazione ordinata per livelli di priorità.

- Uso di livelli qualitativi e relativi (es. “superiore a”).
- Requisiti di granularità comparabile e stesso livello di astrazione.
- Requisiti non mutuamente dipendenti per permettere eliminazioni selettive.
- Convergenza sui requisiti accettati dai principali stakeholder.

3.5.2 Tecnica di Valutazione Valore-Costo

- **Stima del Contributo Relativo**: Ogni requisito è valutato per il suo contributo al valore e al costo del progetto.
- **Diagramma Valore-Costo**: Viene utilizzato per visualizzare la trade-off tra valore e costo, categorizzando i requisiti in alta, media e bassa priorità.
- **Uso della Tecnica AHP**: L’Analytic Hierarchy Process aiuta a determinare quanto ogni requisito contribuisca al valore o al costo del progetto relativamente agli altri requisiti.

3.5.3 Implementazione della Prioritizzazione

I requisiti vengono quindi mappati in una matrice valore-costo per determinare le priorità basate su una valutazione sistematica e quantificata. Questo metodo fornisce una base oggettiva per decisioni di sviluppo e riallocazione delle risorse.

Capitolo 4

Specifica e documentazione

La specifica e la documentazione dei requisiti sono passaggi fondamentali nel processo di sviluppo di sistemi, assicurando che tutte le parti interessate abbiano una comprensione chiara delle caratteristiche concordate del sistema.

Dettagli della Specifica

- **Definizione precisa di tutte le caratteristiche**: Comprende gli obiettivi, i concetti, le proprietà del dominio rilevanti, i requisiti del sistema, le ipotesi e le responsabilità.
- **Razionale delle opzioni adottate**: Motivazione delle scelte effettuate e argomentazioni per la soddisfazione dei requisiti.
- **Evoluzioni e varianti del sistema**: Discussione sulle possibili evoluzioni e varianti del sistema previste nel tempo.

Organizzazione e Documentazione

- **Struttura Coerente**: Organizzazione delle informazioni in una struttura coerente che faciliti la comprensione e l'utilizzo del documento.
- **Forma di Documentazione**: Preparazione della documentazione in una forma comprensibile per tutte le parti coinvolte. Spesso include allegati con costi, piani di lavoro e calendari di consegna.

Documento dei Requisiti

Il risultato di questo processo è un documento formale che dettaglia tutti gli aspetti del sistema concordato, servendo come riferimento principale per lo sviluppo e la manutenzione del sistema.

4.1 Documentazione libera in linguaggio naturale

La documentazione libera in linguaggio naturale offre diversi vantaggi e svantaggi.

Pro della documentazione libera

- **Espressività illimitata**: Il linguaggio naturale permette una vasta gamma di espressioni, rendendo la documentazione estremamente versatile.
- **Facilità di comunicazione**: Essendo in linguaggio naturale, la documentazione è immediatamente comprensibile senza necessità di formazione specifica.

Contro della documentazione libera

- **Propensione agli errori e difetti specifici**: La natura illimitata del linguaggio naturale lo rende suscettibile a errori di specifica e difetti.
- **Ambiguità intrinseca**: Il linguaggio naturale è naturalmente ambiguo, il che può essere dannoso in contesti tecnici. Esempio: interpretazioni errate di connettivi logici possono portare a conclusioni sbagliate.

4.2 Documentazione strutturata nel linguaggio naturale

Tipicamente, un documento dei requisiti non è composto esclusivamente da testo libero, ma utilizza delle linee guida per strutturare il contenuto in modo coerente e comprensibile. Qui di seguito sono elencate alcune regole locali per la scrittura efficace di specifiche in linguaggio naturale.

- **Identifica il pubblico**: Comprendi chi leggerà il documento e scrivi di conseguenza per assicurare la chiarezza e la pertinenza del contenuto.
- **Struttura l'informazione**: Dichiara ciò che intendi fare prima di farlo, motivando le scelte all'inizio e riassumendo i punti chiave alla fine.
- **Definizione dei concetti**: Assicurati che ogni concetto venga definito prima del suo utilizzo nel documento per evitare ambiguità.
- **Chiarimenti continui**: Poni a te stesso domande come "È comprensibile?", "È sufficiente?" e "È rilevante?" per mantenere la pertinenza e la qualità del testo.
- **Controllo della complessità**: Evita di includere più di un requisito, assunzione o proprietà del dominio in una singola frase. Mantieni le frasi brevi e al punto.
- **Linguaggio prescrittivo**: Utilizza "deve" per indicazioni obbligatorie e "dovrebbe" per quelle desiderabili.
- **Evita gergo tecnico**: Minimizza l'uso di gergo tecnico e acronimi a meno che non siano chiaramente definiti o universalmente comprensibili.
- **Esempi illustrativi**: Fornisci esempi suggeriti per chiarire dichiarazioni astratte e complesse.

- **Uso di diagrammi**: Include diagrammi per rappresentare relazioni complesse tra elementi, facilitando la comprensione visiva delle informazioni.

Queste regole sono pensate per migliorare l'efficacia della documentazione mantenendola accessibile e comprensibile per tutte le parti interessate.

4.2.1 Regole locali

La documentazione dei requisiti di sistema deve seguire regole locali ben definite per garantire coerenza e comprensibilità. Queste regole sono fondamentali per strutturare il contenuto in modo che sia accessibile e tracciabile.

Uso delle Tabelle Decisionali

- Le tabelle decisionali aiutano a gestire combinazioni complesse di condizioni, offrendo un supporto sistematico e chiarezza nella documentazione dei requisiti.
- Questo approccio assicura una completa verifica e un'integrazione logica delle condizioni di input e output.

Modelli di Dichiarazione Standardizzati

Le tabelle di decisione sono strumenti utilizzati per mappare e implementare le logiche decisionali che coinvolgono condizioni multiple in maniera sistematica e strutturata. Nelle tabelle di decisione, ogni riga rappresenta un insieme di condizioni di ingresso mentre ogni colonna corrisponde a un risultato decisionale specifico. Questo strumento permette agli ingegneri di vedere chiaramente come combinazioni diverse di condizioni d'ingresso influenzino le azioni da intraprendere.

Le condizioni di ingresso possono includere vari fattori come comandi di accelerazione, posizioni relative dei treni, o velocità di ingresso in una stazione, e le azioni corrispondenti possono essere l'attivazione dei sistemi di frenata, l'emissione di allarmi, o altre misure di sicurezza.

Vantaggi delle Tabelle di Decisione

- **Chiarezza e sistematicità:** Facilitano la comprensione delle relazioni tra cause ed effetti e aiutano a garantire che tutte le possibili situazioni siano considerate.
- **Facilità di manutenzione:** Le modifiche ai criteri o ai procedimenti possono essere implementate aggiornando semplicemente la tabella.
- **Verifica di completezza:** Ogni possibile combinazione di condizioni è rappresentata, permettendo un controllo completo su ogni scenario decisionale.
- **Forniscono test di accettazione:** Le tabelle di decisione possono essere utilizzate per definire test di accettazione per verificare che il sistema soddisfi i requisiti specificati.

Svantaggi delle Tabelle di Decisione

- **Scalabilità:** Man mano che il numero di condizioni aumenta, le tabelle diventano esponenzialmente grandi e complesse da gestire.
- **Rigidità:** Possono non adattarsi bene a scenari non previsti e possono richiedere frequenti aggiornamenti per rimanere efficaci in ambienti dinamici.

Consigli per i template

I template standardizzati per la formulazione delle affermazioni sono fondamentali per standardizzare e organizzare la documentazione dei requisiti nei progetti. Di seguito è riportata un'analisi dettagliata dei componenti essenziali di questi template:

Le componenti principali di un template standard per le affermazioni includono:

- **Identificatore:** Serve a nominare l'affermazione in modo intuitivo e, se necessario, gerarchico, facilitando la ricerca e il riferimento incrociato tra affermazioni correlate.
- **Categoria:** Classifica l'affermazione in categorie quali requisiti funzionali o di qualità, assunzioni, proprietà del dominio, definizioni o esempi di scenario. Questa classificazione aiuta ad organizzare le affermazioni in modo logico.
- **Specifica:** Dettaglia la formulazione dell'affermazione seguendo regole stilistiche precise per garantire chiarezza e precisione.
- **Criterio di Adeguazione:** Fornisce criteri per misurare o verificare l'affermazione, essenziali per la validazione e il controllo della qualità.
- **Fonte:** Indica la fonte dell'affermazione per la tracciabilità, collegandola alle evidenze o ai requisiti originali.
- **Razionale:** Spiega il motivo dietro l'affermazione, migliorando la comprensione e facilitando la tracciabilità.
- **Interazione:** Descrive come l'affermazione interagisce con altre affermazioni, inclusi potenziali contributi o conflitti, essenziale per la gestione delle dipendenze.
- **Livello di Priorità:** Stabilisce un livello di priorità per l'affermazione, aiutando nella gestione delle risorse e nella pianificazione del progetto.
- **Stabilità, Livelli di Comunalità:** Aiutano nella gestione del cambiamento e nella valutazione dell'applicabilità dell'affermazione attraverso diverse applicazioni o progetti.

Criteri di Adeguatezza

I criteri di adeguatamente rendono misurabili le dichiarazioni, quantificando l'estensione con cui devono essere soddisfatte, essenziali specialmente per la misurabilità dei requisiti non funzionali (`NFRs`). Per esempio, le date di riunioni programmate dovrebbero essere comode per il 90% dei partecipanti in almeno l'80% dei casi.

Documentazione Disciplinata in linguaggio naturale Strutturato

La documentazione deve seguire regole globali per la strutturazione dei documenti di requisiti, raggruppando elementi simili come obiettivi del sistema, componenti, e caratteristiche del software. Questo aiuta a mantenere la documentazione organizzata e facilmente navigabile.

Template `IEEE Std-830` per l'Organizzazione del documento dei Requisiti

Il template `IEEE Std-830` fornisce una struttura standardizzata per organizzare il documento dei requisiti, includendo specifiche per scopo, dominio e altri elementi critici. Questo standard aiuta le organizzazioni a mantenere una documentazione chiara e conforme alle migliori pratiche del settore.

Uso delle Notazioni Diagrammatiche

Le notazioni diagrammatiche sono utilizzate per completare o sostituire la prosa in linguaggio naturale. Sono dedicate ad aspetti specifici del sistema (*come è o come sarà*) e sono grafiche per facilitare la comunicazione e fornire una panoramica. Queste notazioni possono essere semi-formali, includendo:

- Dichiarazione di elementi in linguaggio formale (*sintassi, semantica*), che permette controlli superficiali sugli elementi del documento dei requisiti e la loro elaborazione automatica.
- Specifiche informali delle proprietà degli elementi in linguaggio naturale.

4.3 Documentazione strutturata con l'ausilio di diagrammi

4.3.1 Scope del sistema

Context Diagram

Il diagramma di contesto è uno strumento utile per rappresentare il sistema mediante le componenti principali e le relazioni tra di esse. I nodi sono le componenti del sistema, mentre gli archi rappresentano le relazioni tra di esse.

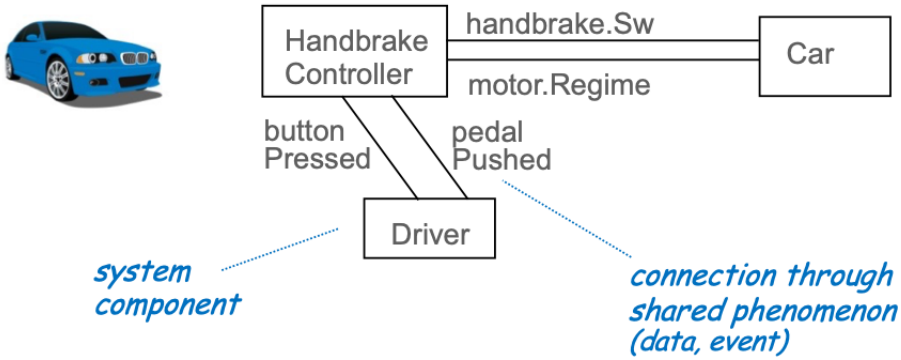


Figura 4.3.1: Esempio di Context Diagram

Problem Diagram

I diagrammi dei problemi sono una forma dettagliata di diagrammi di contesto che evidenziano i componenti del sistema e le loro interazioni. Questi diagrammi mostrano chi controlla e monitora i fenomeni condivisi e indicano i requisiti e i componenti influenzati da essi.

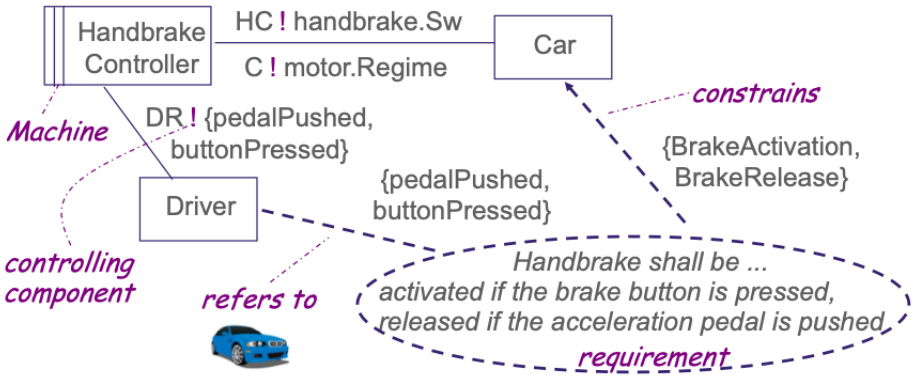


Figura 4.3.2: Esempio di Problem Diagram

Frame Diagram

I frame diagram sono utilizzati per catturare pattern di problemi frequenti all'interno di un sistema, evidenziando fenomeni tipizzati e componenti tipizzati. Questi diagrammi aiutano a comprendere le interazioni tra i vari elementi del sistema e i requisiti correlati.

- **Fenomeni Tipizzati:**
  - **Causale (C):** Fenomeni che causano effetti diretti.
  - **Evento (E):** Fenomeni che rappresentano eventi specifici.
  - **Simbolico (Y):** Fenomeni che rappresentano simboli o dati.
- **Componenti Tipizzati:**
  - **Causale (C):** Componenti che causano effetti diretti.
  - **Biddable (B):** Componenti che possono essere controllati o influenzati.
  - **Lessicale (X):** Componenti che rappresentano dati o informazioni.

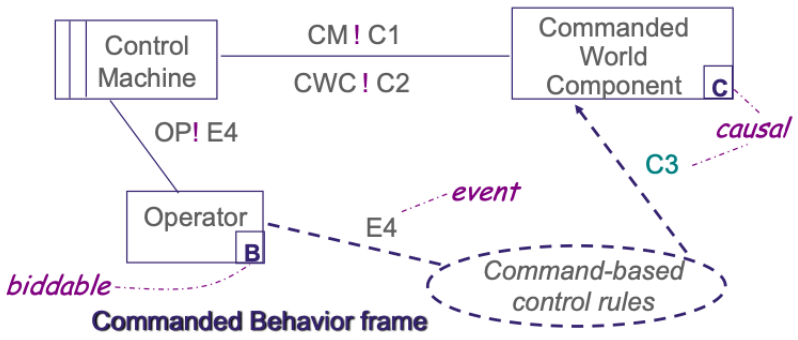


Figura 4.3.3: Esempio di Frame Diagram

Un esempio di istanziazione di un frame diagram è il seguente:

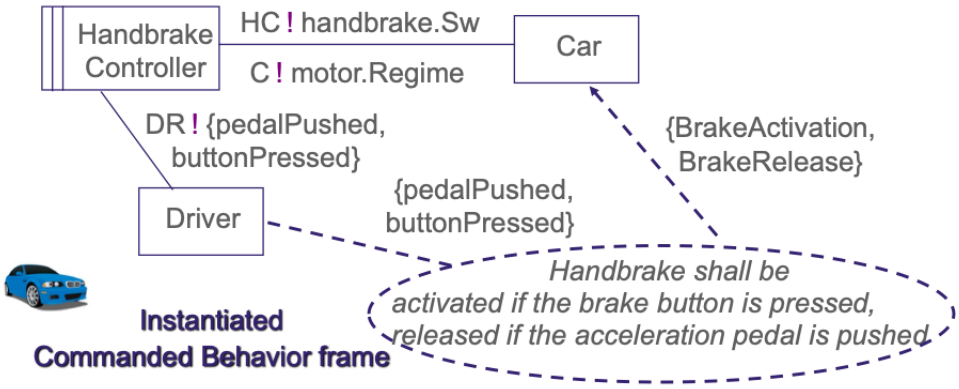


Figura 4.3.4: Esempio di Frame Diagram istanziato

4.3.2 Diagrammi Entità-Relazione

I diagrammi Entità-Relazione sono utilizzati per rappresentare le entità e le relazioni tra di esse in un sistema. Questi diagrammi sono utilizzati per modellare i dati e le loro interazioni all'interno del sistema, aiutando a definire i requisiti relativi alla gestione dei dati.

- **Specializzazione delle Entità:** Le entità possono essere specializzate in sottoclassi con caratteristiche specifiche (attributi, relazioni). Ad esempio, un *ImportantParticipant* eredita attributi e relazioni dalla superclass *Participant*, ma può avere attributi aggiuntivi come *Preferences*.
- **Annotazioni dei Diagrammi:** Essenziali per definire con precisione gli elementi e evitare errori di specifica. Ad esempio, l'annotazione per *Participant* può specificare che è una persona attesa alla riunione in un ruolo specifico.

Svantaggi dei Diagrammi Entità-Relazione

Non si riesce a distinguere tra prescrittivi e percettivi.

4.3.3 Diagrammi SADT (Structured Analytics Design Technique)

I diagrammi SADT sono utilizzati per catturare attività e dati nel sistema, sia nello stato attuale che in quello futuro. Questi diagrammi si dividono in due categorie principali:

- **Actigram:** Relaziona le attività tramite collegamenti di dipendenza dai dati (*input/output*). Mostra come le attività sono correlate attraverso i dati che utilizzano e producono.
- **Datagram:** Relaziona i dati tramite collegamenti di dipendenza dal controllo (*produttori/consumatori*). Indica come i dati vengono prodotti, consumati, validati e le risorse necessarie.
  - **Est:** attività di produzione.
  - **Nord:** attività di validazione.
  - **Ovest:** attività di consumo.
  - **Sud:** risorse necessarie.

Nei diagrammi SADT, esiste una dualità dati-attività:

- I dati in un actigram devono apparire in un datagram.
- Le attività in un datagram devono apparire in un actigram.

Vantaggi dei Diagrammi SADT

- Forniscono una chiara visualizzazione delle dipendenze tra attività e dati.
- Aiutano a identificare le risorse necessarie e le relazioni di controllo.

Svantaggi dei Diagrammi SADT

- Possono diventare complessi e difficili da gestire per sistemi di grandi dimensioni.
- Richiedono una comprensione approfondita delle attività e dei dati coinvolti.

Un esempio di diagramma SADT mostra come le attività di gestione dei vincoli (*handling constraints*) siano correlate attraverso richieste di meeting, data range e risorse necessarie. Ogni attività e dato è chiaramente definito e le loro interazioni sono mappate per garantire la coerenza e la completezza del sistema.

4.3.4 Diagrammi di Flusso dei Dati (*Dataflow Diagrams*)

I diagrammi di flusso dei dati (DFD) sono utilizzati per catturare le operazioni del sistema collegate dalle dipendenze dei dati. Questi diagrammi sono più semplici ma meno espressivi rispetto agli actigram.

- **Operazione:** Attività di trasformazione dei dati.
- **Link di Input e Output:** Flussi di dati. Le operazioni richiedono dati in entrata per produrre dati in uscita (non flusso di controllo).
- **Regole di Trasformazione dei Dati:** Devono essere specificate nelle annotazioni (linguaggio naturale strutturato) o in un altro DFD (raffinamento delle operazioni, cf. SADT).
- **Componenti del Sistema e Repositori di Dati:** Origini e destinazioni del flusso.

Vantaggi dei Diagrammi di Flusso dei Dati

- Semplificano la rappresentazione delle operazioni di trasformazione dei dati.
- Facilitano la comprensione delle dipendenze dei dati nel sistema.

Svantaggi dei Diagrammi di Flusso dei Dati

- Meno espressivi rispetto agli actigram per rappresentare le operazioni complesse.
- Richiedono annotazioni dettagliate per specificare le regole di trasformazione dei dati.

I diagrammi di flusso dei dati sono essenziali per garantire la coerenza e la completezza del sistema, con ogni attività che deve avere un input e un output, e tutti i dati devono avere un produttore e un consumatore. Le regole di consistenza/completa possono essere verificate da strumenti, simili ai diagrammi SADT.

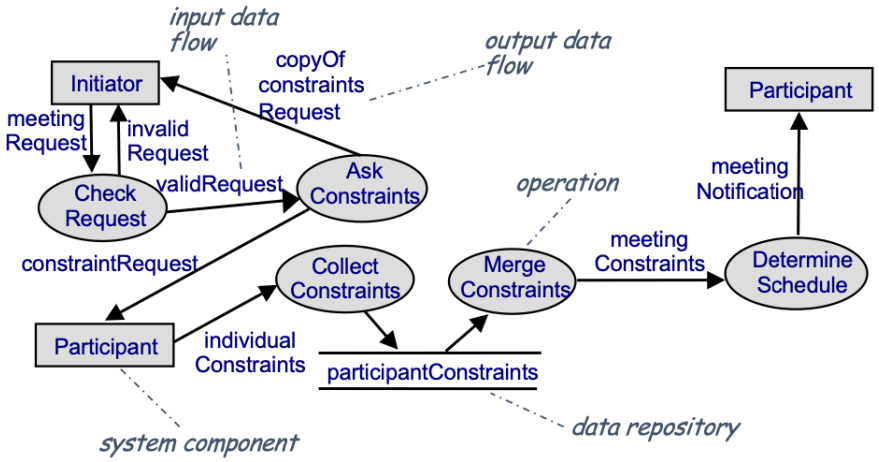


Figura 4.3.5: Esempio di Diagramma di Flusso dei Dati

4.3.5 Use Case Diagram

I diagrammi dei casi d'uso sono utilizzati per catturare le operazioni che devono essere eseguite da un componente del sistema e le interazioni con altri componenti. Questi diagrammi forniscono una vista di insieme semplice ma a volte vaga delle operazioni del sistema.

- **Operazioni del Sistema:** Catturano le operazioni da eseguire e le interazioni con altri componenti. Sono introdotti in `UML` per sostituire i `DFD`.
- **Annotazioni e Scenari di Interazione:** Necessari per rendere precisi i diagrammi dei casi d'uso.
- **Meccanismi di Strutturazione:**
  - `<<include>>`: Specifica una "sotto-operazione".
  - `<<extend>>` + precondizione: Specifica un'operazione "variante" in casi eccezionali.

Vantaggi dei Diagrammi dei Casi d'Uso

- Forniscono una vista di insieme delle operazioni del sistema.
- Facilmente comprensibili anche da stakeholder non tecnici.

Svantaggi dei Diagrammi dei Casi d'Uso

- Possono essere troppo vaghi senza annotazioni dettagliate.
- Richiedono scenari di interazione per essere completamente utili.

I diagrammi dei casi d'uso sono strumenti potenti per la modellazione delle operazioni del sistema, ma devono essere integrati con annotazioni e scenari di interazione per garantire una comprensione completa e precisa delle funzionalità del sistema.

4.3.6 Event trace diagrams

I diagrammi delle tracce degli eventi catturano scenari positivi attraverso sequenze di interazioni tra istanze dei componenti del sistema. Questi diagrammi mostrano come le interazioni si svolgono nel tempo.

Vantaggi dei Diagrammi delle Tracce degli Eventi

- Permettono di visualizzare chiaramente le interazioni tra i componenti del sistema.
- Forniscono una sequenza temporale dettagliata degli eventi di interazione.

Svantaggi dei Diagrammi delle Tracce degli Eventi

- Possono diventare complessi con un numero elevato di componenti e interazioni.
- Richiedono una comprensione dettagliata delle sequenze di interazione.

4.3.7 State Machine Diagram

I diagrammi delle macchine a stati catturano i comportamenti ammissibili dei componenti del sistema. Essi rappresentano la sequenza delle transizioni di stato per gli elementi controllati.

- **Comportamento di un'istanza di componente:** Sequenza di transizioni di stato per gli elementi che controlla.
- **Stato della Macchina a Stati:** Insieme di situazioni in cui una variabile che caratterizza un elemento controllato ha sempre lo stesso valore. Ad esempio, lo stato "MeetingScheduled" ha sempre lo stesso valore per "Date" e "Location".
- **Stati Iniziali e Finali:** Stati in cui l'elemento appare o scompare. Gli stati possono avere una certa durata.
- **Transizione di Stato della Macchina a Stati:** Causata da un evento associato. Se l'elemento è nello stato di origine e l'evento si verifica, allora passa allo stato di destinazione. Gli eventi sono fenomeni istantanei.

Vantaggi dei Diagrammi delle Macchine a Stati

- Permettono di modellare chiaramente i comportamenti ammissibili dei componenti del sistema.
- Forniscono una rappresentazione dettagliata delle transizioni di stato.

Svantaggi dei Diagrammi delle Macchine a Stati

- Possono diventare complessi con un numero elevato di stati e transizioni.
- Richiedono una comprensione dettagliata dei comportamenti dei componenti.

Per evitare l'esplosione combinatoria di stati è possibile utilizzare gli statecharts, che permettono di modellare comportamenti complessi in modo più chiaro e conciso.

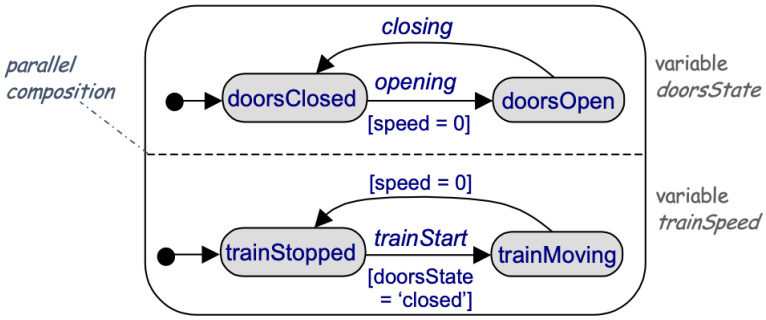


Figura 4.3.6: Esempio di State Machine Diagram

4.3.8 `R-net` Diagram

Se vogliamo ragionare in termini di stimolo e risposta agli stimoli, una versione alternativa è il diagramma `R-net`, che cattura le relazioni tra gli stimoli provenienti da un'entità esterna e le risposte generate dal sistema.

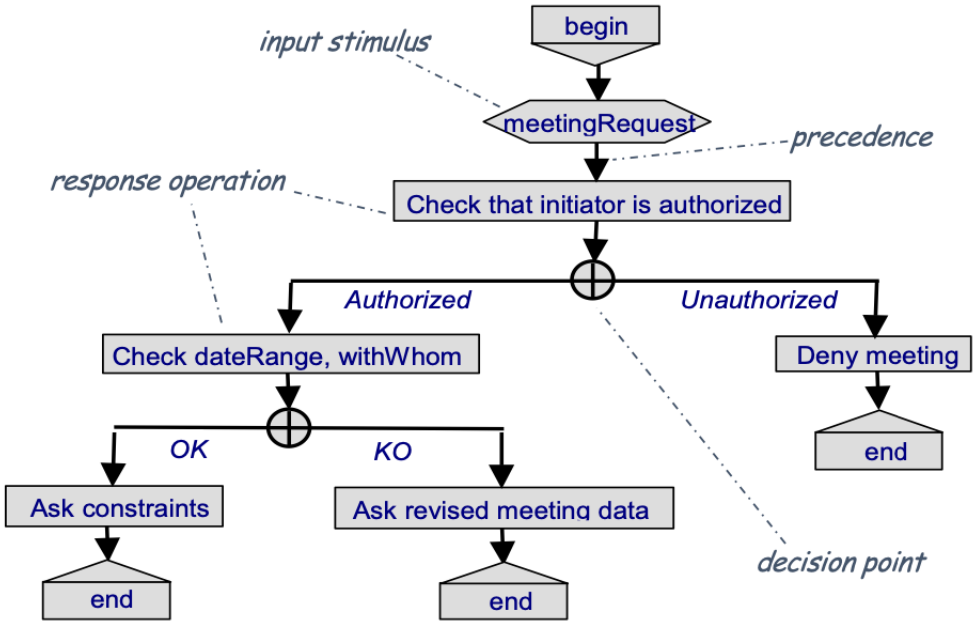


Figura 4.3.7: Esempio di R-net Diagram

4.4 Integrare i Diagrammi in un sistema multi-vista

Avendo a disposizione una varietà di diagrammi per rappresentare i requisiti del sistema, possiamo integrare i diversi diagrammi dato che ciascuno fornisce una prospettiva unica sul sistema. Una prima verifica, data dai sistemi automatici, può essere fatta per verificare la consistenza tra i diagrammi.

Le inconsistenze che si possono verificare possono essere diverse, tra cui:

- tutti i componenti di un problem diagram devono essere presenti in un diagramma entità-relazione.
- tutti i fenomeni di un problem diagram devono essere presenti in un diagramma entità-relazione come un'entità, un attributo o una relazione.

Ci sono viste alternative in `UML` che ci permetto di ragionare, non tanto sull'architettura, ma supportano nella fasi di progettazione e validazione dei requisiti, come ad esempio:

- **Diagrammi delle Classi**: utili per la vista strutturale del sistema.
- **Use Case Diagram**: utile per la vista operazionale del sistema.
- **Sequence Diagram**: utile per la visione degli scenari.
- **State Diagram**: utile per la visione delle state machine.

Vantaggi dell'ausilio di Diagrammi

- Sono diversi dalla presentazione in linguaggio naturale, permettendo quindi di essere guidati.
- Hanno una semantica precisa, non ambigua.
- Mezzo di informazione più funzionale per il dialogo con gli stakeholder.
- Hanno una struttura grafica che permette di avere una panoramica di alto livello.
- Sono facili da capire e da comunicare.
- Permettono di fare un'analisi di primo livello e sono supportati da tool.

Svantaggi dell'ausilio di Diagrammi

- Il linguaggio semantico può essere vago (*con interpretazioni differenti*).
- Spesso vengono formalizzati solo gli aspetti più evidenti e superficiali del sistema, lasciando non formalizzate le proprietà dettagliate degli elementi, il che può limitare la profondità dell'analisi.
- L'analisi automatizzata è limitata.
- Si modellano solo aspetti funzionali e strutturali.

Le specifiche più formali sono richieste per sistemi *mission critical*.

Capitolo 5

Validazione e verifica

L'obiettivo di tale fare è quella di partire da un documento dei requisiti consolidato e di verificare che esso sia corretto e completo.

Alcuni errori siamo stati in grado di evitarli in partenza, grazie alle fasi precedenti. In questa fase ci concentriamo nell'individuazione di errori che non sono stati individuati in precedenza.

Il primo obiettivo è la **ricerca** che si divide in tre fasi. La prima fase da seguire è la **validazione**, che consiste nel verificare che il documento dei requisiti soddisfi le esigenze del cliente. La seconda fase è la **verifica**, che consiste nel verificare che il documento sia corretto e completo. La terza fase è la **controllo**, che riguarda il raggiungimento tei target di qualità.

Il secondo obiettivo è la **documentazione** dei difetti, che include anche l'**analisi** delle cause e la loro correzione.

Output della fase

L'output di tale fase è la versione finale del documento dei requisiti.
Oltre a tale documento possiamo considerare anche i test di accettazione consolidati, eventualmente un prototipo, un piano di sviluppo e potenzialmente un contratto.

5.1 Come si affronta la revisione

5.1.1 Ispezione e revisione

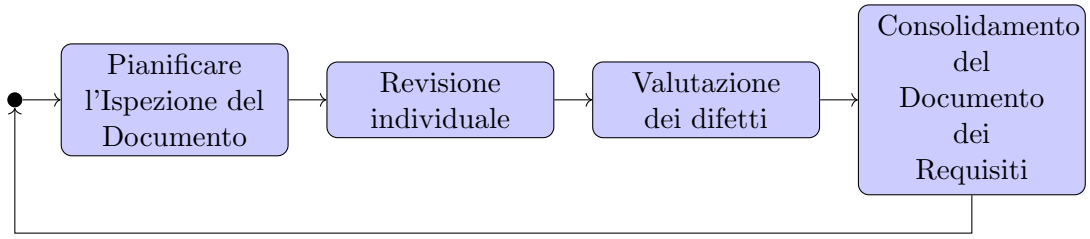
L'idea è quella di avere uno o più esperti che prendono il documento dei requisiti che prendono il documento dei requisiti e lo analizzano oppure utilizzare dei meeting di gruppo per discutere il documento.

Principalmente si utilizzano due tecniche:

- **Walkthrough**: si tratta di una lettura del documento, con l'obiettivo di trovare errori.

- **Più strutturato**: utilizzando degli ispettori esterni, dei meeting ben strutturati, report o altro.

Un processo che mira ad assicurare la qualità del documento dei requisiti segue tali fasi:



- **Pianificare l'ispezione del documento**: si tratta di definire chi sono gli ispettori, come si svolgerà l'ispezione, quali sono gli obiettivi e quali sono i criteri di accettazione.
- **Revisione individuale**: ogni ispettore legge il documento e cerca di individuare errori. Tale fase può seguire diverse modalità:
  - Liberamente: ogni ispettore legge il documento e cerca di individuare errori.
  - Guidata mediante checklist: ogni ispettore segue una lista di controllo basata sul dominio.
  - Basato sul processo.
- **Valutazione dei difetti**: si tratta di valutare i difetti capendo come sono stati generati e come possono essere corretti.
- **Consolidamento del documento dei requisiti**: si tratta della fase in cui viene consolidato il documento dei requisiti.

Tale processo può essere ripetuto più volte, fino a quando non si è soddisfatti della qualità del documento.

5.1.2 Linee guida per la revisione

- **Rapporto di Ispezione:**
  - Deve essere *informativo*, *accurato* e *costruttivo*.
  - Non deve contenere opinioni personali o commenti che possano risultare offensivi.
  - Deve seguire una *struttura standard* che permetta l'aggiunta di commenti liberi e sia di facile lettura.
  - Può servire da guida per la revisione individuale.
- **Gli Ispettori:**
  - Devono essere *indipendenti* dagli autori del documento sotto revisione.
  - Devono rappresentare tutti gli *stakeholder* e provenire da *background diversi*.

- **Tempistica dell'Ispezione:**
  - L'ispezione non deve avvenire né *troppo presto* né *troppo tardi*.
  - Incontri *brevi* e *ripetuti* risultano più efficaci.
- **Focus Maggiore su Parti Critiche:**
  - Maggiore è il numero di difetti in una parte, maggiore deve essere lo *scrutinio* su quella parte.

5.1.3 Checklist

L'obiettivo principale delle liste di controllo per l'ispezione è di dirigere la ricerca dei difetti verso specifiche problematiche, migliorando così l'efficienza e l'efficacia del processo di revisione. Le liste di controllo si suddividono in diverse categorie, ognuna con un focus particolare:

- **Liste basate sui difetti:** Queste liste comprendono domande generiche che sono strutturate in base al tipo di difetto. L'intento è di coprire un ampio spettro di possibili anomalie in modo organizzato.
- **Liste specifiche per qualità:** Tali liste affinano le domande delle liste basate sui difetti per concentrarsi su specifiche categorie di requisiti non funzionali (NFR), come la sicurezza, la performance e l'usabilità. Queste liste possono anche essere basate su parole guida e sono frequentemente orientate a identificare omissioni, migliorando così la completezza del software o del prodotto analizzato.
- **Liste specifiche per dominio:** Queste liste rappresentano un'ulteriore specializzazione e si concentrano sui concetti e le operazioni specifici di un dominio. Sono particolarmente utili per offrire una guida dettagliata nella ricerca di difetti, assicurando che le peculiarità del dominio siano adeguatamente considerate.
- **Liste basate sul linguaggio:** Queste liste adattano le liste basate sui difetti ai costrutti di linguaggi di specifica particolari, supportando processi di automazione. La ricchezza dei linguaggi di specifica consente implementazioni di controlli più sofisticati e mirati.

Vantaggi delle Checklist

- **Maggiore efficacia rispetto all'ispezione del codice:** Utilizzo di un modello basato su processi che combina checklist basate sui difetti, specifiche per qualità e specifiche per linguaggio, risultando estremamente efficace.
- **Ampia applicabilità:** Adatte per ogni tipo di difetto e formato di specifica, rendendole strumenti versatili.

Limitazioni delle Checklist

- **Onere e costo del processo di ispezione:** La dimensione del materiale di ispezione e il tempo/costo necessari per gli ispettori esterni e le riunioni di revisione possono essere significativi.
- **Nessuna garanzia di rilevare tutti i difetti importanti:** Nonostante l'efficacia, le checklist non possono garantire il rilevamento di tutti i difetti critici.

5.2 Approccio guidato da Query

La quality assurance può essere vista come delle particolari interrogazioni a delle strutture dati. Tipicamente le query possono essere formulate in presenza di una documentazione strutturata dei requisti. Per strutturata intendiamo una documentazione che si basa su diagrammi.

Quando si formulano tali query utilizzando i `CASE` tools, si possono ottenere delle risposte automatiche.

5.3 Validazione dei requisiti mediante animazioni

L'obiettivo principale di questa metodologia è verificare l'adeguatezza dei requisiti rispetto alle reali necessità degli utenti. Questo processo può essere attuato attraverso due approcci principali:

Approccio 1: Visualizzazione di Scenari di Interazione

Questo approccio consiste nel mostrare scenari di interazione concreti in azione, utilizzando strumenti di attuazione sui diagrammi di Evento Tempo (`ET`). Tuttavia, questo metodo presenta sfide legate alla copertura completa degli scenari, che può risultare incompleta.

Approccio 2: Utilizzo di Strumenti di Animazione delle Specifiche

L'animazione delle specifiche si svolge in quattro fasi principali:

1. **Generazione del Modello Esecutivo:** Si estrae o si genera un modello eseguibile direttamente dalle specifiche tecniche.
2. **Simulazione del Comportamento:** Il sistema viene simulato basandosi su questo modello, dove eventi stimolo vengono inviati per imitare il comportamento dell'ambiente circostante e si osserva la risposta del modello.
3. **Visualizzazione della Simulazione:** La simulazione viene visualizzata mentre è in corso, permettendo una valutazione immediata e dinamica del comportamento del sistema.
4. **Feedback degli Utenti:** I feedback degli utenti vengono raccolti e analizzati per valutare l'efficacia della simulazione e del modello proposto.

Punti di Forza

- **Verifica dell'Adeguazione:** Metodo efficace per controllare l'adeguatezza dei requisiti rispetto alle necessità reali e all'ambiente attuale.
- **Coinvolgimento degli Stakeholder:** Il principio "WYSIWYC" (What You See Is What You Check) permette ai stakeholder di visualizzare e verificare i requisiti direttamente.
- **Estensione a Controesempi:** Possibilità di animare controesempi generati da altri strumenti come i verificatori di modelli, migliorando così la robustezza del sistema.
- **Riutilizzo degli Scenari:** Gli scenari di animazione possono essere conservati per replay futuri e usati come dati di test di accettazione.

Limitazioni

- **Problema della Copertura:** La progettazione degli scenari deve essere attentamente gestita per assicurare che non vengano tralasciati problemi significativi. Non vi è garanzia di identificare tutti i difetti importanti.
- È necessaria una specifica formale dettagliata per eseguire un'animazione efficace dei requisiti, richiedendo un investimento considerevole in termini di tempo e risorse.

Capitolo 6

Orientamento ai goal

Il goal, nell'ingegneria dei requisiti, è un concetto intrinseco quando si parla di requisiti. Di fatto abbiamo sempre avuto a che fare con gli obiettivi, e la metodologia *goal-oriented* definisce il concetto di goal come astrazione chiave che ci guida nell'ingegneria dei requisiti.

6.1 Definizione di goal

6.1.1 Soddisfacibilità dei goal e la cooperazione degli agenti

Goal

Un goal è uno statement **prescrittivo** che esprime un desiderio, un obiettivo o che il sistema debba raggiungere attraverso la cooperazione dei suoi agenti.

Lo statement prescrittivo perché può essere esprimibile con "deve", "dovrebbe", "può", "potrebbe", e così via.

Agente

Un agente è un componete **attivo** del sistema *as-is* o *to-be* che è **responsabile** del raggiungimento di un goal.

Capiamo quindi che gli agenti possono essere umani, software o hardware. L'agente è quindi un **ruolo** che prende decisioni, rispetto che un individuo, in modo che il goal assegnato sia raggiunto. Per farlo dovrà *monitorare* e *controllare* delle grandezze.

Esempio di goal soddisfatto dalla cooperazione di più agenti

Il raggiungimento del'obiettivo "*Restituzione di un libro in biblioteca*" può essere soddisfatto dalla cooperazione di: colui che restituisce il libro, il bibliotecario e il software che gestisce il prestito.

6.1.2 Goal rispetto alle proprietà di dominio

Proprietà di Dominio

Le proprietà di dominio sono degli *statement* descrittivi rispetto l'ambiente.

Lo statement descrittivo si esprime con "è", "sono" e così via.

Se i goal possono essere negoziati, indeboliti o prioritizzati, le proprietà di dominio sono **vincoli** che non possono essere negoziati, ma sono comunque necessari nel documento dei requisiti.

I goal vanno **decisi**, mentre le proprietà di dominio vanno **identificate**.

Esempio di goal rispetto alle proprietà di dominio

Consideriamo "*Se le porte del treno sono aperte, allora non sono chiuse*". Sono quindi proprietà legate a vincoli fisici o dell'ambiente nel quale il sistema opera.

6.2 Granularità dei goal e le relazioni con i requisiti e le assunzioni

I goal possono essere espressi a diversi livelli di granularità:

- **Goal di alto livello**: sono statement strategici, che hanno quindi una granularità alta, come ad esempio "*Migliorare la qualità del servizio del 50%*".
- **Goal di basso livello**: sono statement tecnici, che hanno quindi una granularità bassa, come ad esempio "*Il sistema invia un reminder per il rinnovo del prestito*".

Se ragioniamo in termini di agenti, per raggiungere un goal di alto livello servirà la cooperazione di più agenti, mentre per raggiungere un goal di basso livello servirà la cooperazione di un minor numero di agenti.

I goal prendono un nome distinto a seconda della relazione che hanno con gli agenti.

Requisto

Il requisto è un goal che deve essere soddisfatto da un **unico** agente nel *software to-be* (*come per esempio il meccanismo di frenata*).

Prospettiva

La prospettiva è un goal che deve essere soddisfatto da un **unico** agente nel *environment* (*come per esempio le persone*).

6.3 Tipologie e categorie di goal

I goal possono essere classificati in diverse tipologie:

- **Behavioral goal**: goal prescrittivi che esprimono il comportamento. Tali goal si classificano in goal che dobbiamo **raggiungere** o **mantenere**.
- **Soft goal**: esprimono una preferenza tra funzionamenti distinti.

6.3.1 Behavioral goal (prescrittivi)

Rappresentano dei vincoli che rappresentano un sottoinsieme dei funzionamenti che il nostro sistema deve accettare a cui è possibile attribuire una risposta univoca, o *sì* o *no*.

Distinguiamo tra:

- `Achieve[targetCondition]`: rappresentano una condizione target da raggiungere
  - `[if CurrentCondition then] sooner-or-later targetCondition`
- `Maintain[targetCondition]`: rappresentano una condizione target da mantenere
  - `[if CurrentCondition then] always targetCondition`
- `Avoid[targetCondition]`: rappresentano una condizione target da evitare
  - `[if CurrentCondition then] never targetCondition`

6.3.2 Soft goal

Rappresentano il concetto di preferenza del sistema, questi quindi non possono essere soddisfatti in maniera binaria come nei behavioral goal, ma possono essere più o meno soddisfatti.

Tipicamente hanno la forma di massimizzazione o minimizzazione di una certa grandezza, come ad esempio "*La condizione di stress degli operatori deve essere minimizzata*".

6.3.3 Categorie di goal

I goal possono essere classificati in diverse categorie, ma tale divisione non è sempre netta, in quanto un goal può appartenere a più categorie.

- **Funzionali**: sono goal che esprimono le funzionalità e i servizi che il sistema dovrà offrire, usati per costruire un modello operativo del sistema.
- **Qualitativi**: sono goal che esprimono le qualità che il sistema dovrà avere, come ad esempio la sicurezza, l'usabilità, l'efficienza, ecc.
- **Di sviluppo**: sono goal che esprimono le attività che dovranno essere svolte per lo sviluppo del sistema, come ad esempio la documentazione, la formazione del personale, ecc.

6.4 Il ruolo centrale dei goal nel processo di ingegneria dei requisiti

Il ruolo dei goal è quello di supportare un meccanismo di **raffinamento e astrazione** dei goal, raggiungendo goal di più alto livello attraverso la cooperazione di goal in `AND` e `OR` di più basso livello.

Possiamo rappresentare formalmente il concetto di goal come:

$$\{\texttt{REQ}, \texttt{Exp}, \texttt{Dom}\} \models \texttt{G}$$

Che possiamo leggere come "*Considerate le proprietà di dominio, i requisiti/sottogoal che assicurano che il goal G sia soddisfatto sotto l'ipotesi delle proprietà di dominio*".

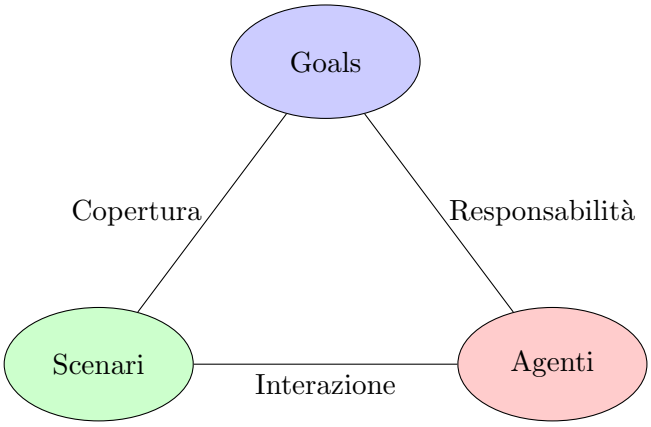
Tale rappresentazione formale ci permette di raggiungere due obiettivi:

- **Completezza**: assicurare che tutti i goal siano raggiunti.
- **Rilevanza**: se un qualche requisito è utilizzato per ottenere un goal, allora tale requisito è rilevante.

Ragionando in termini di evoluzione è importante decidere, quali sono i requisiti più stabili e quelli meno stabili in modo da osservarli con maggiore attenzione. Tipicamente i goal di alto livello sono più stabili, mentre i goal di basso livello sono più instabili perché potremmo avere alternative, anche data la presenza di `OR` decomposizione.

Bisogna evitare ti ragionare *top-down*, tipicamente si utilizzano entrambe le strategie, *top-down* e *bottom-up*. La metodologia *bottom-up* viene chiamata **goal-abstraction**, dove si cerca di estrarre le motivazioni per cui un determinato goal è stato definito.

Tipicamente, però, si ragiona in termini di *scenario-oriented* e *agent-oriented*.



Gli scenari devono coprire tutti i goal, e abbiamo delle interazioni tra agenti e scenari. Sono quindi un modo concreto in cui identifichiamo o validiamo i goal.

Capitolo 7

Goal Diagram

I goal rappresentano degli *statement* prescrittivi delle proprietà che il sistema deve avere attraverso la cooperazione di più agenti. Tali *statement* sono realizzati attraverso la terminologia e il dominio nella quale il sistema opera e sono a vari livelli di granularità.

Per capire se un ci troviamo di fronte ad un goal dobbiamo capire se questa proprietà può essere negoziata, indebolita o rafforzata. Altrimenti ci troviamo di fronte ad una proprietà di dominio.

Si tratta di una vista **intenzionale del sistema**.

7.1 Rappresentazione dei Goal

All'interno del goal diagram i goal vengono rappresentati con un parallelogramma che può o meno avere il bordo in grassetto.

Figura 7.1.1: Rappresentazione di un goal

Possono avere delle annotazioni per specificare ulteriori attributi di questo goal, e possono avere delle precise definizioni a seconda del progetto software.

7.2 Raffinamento dei goal

Il raffinamento permette di suddividere un goal in sotto-goal per poterlo realizzare. Questo raffinamento può essere di due tipi:

- `AND`-refinement.
- `OR`-refinement.

7.3 AND-refinement

Nell'AND-refinement il goal viene suddiviso in sotto-goal che devono essere soddisfatti tutti per poter soddisfare il goal principale.

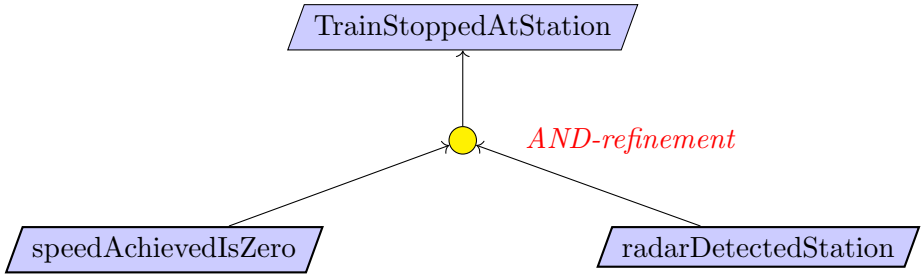


Figura 7.3.1: Rappresentazione di una raffinamento di tipo AND

Il raffinamento di tipo AND deve essere **completo**, ciò significa che il raggiungimento degli obiettivi in AND è una condizione sufficiente per il raggiungimento del goal di più alto livello.

$$\{G_1, G_2, \ldots, G_n\} \models G$$

7.3.1 Proprietà di dominio

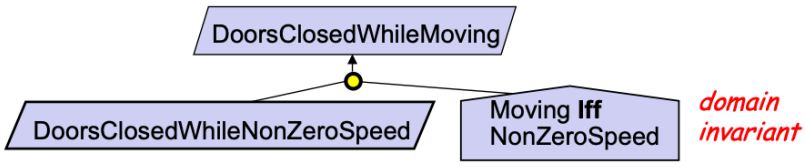


Figura 7.3.2: Rappresentazione di una proprietà di dominio

- **Invariante di dominio**: proprietà di dominio che non varia.
- **Ipotesi di dominio**: proprietà di dominio che assume una condizione.

7.3.2 Altre proprietà

- **Consistenza**: non ci sono contraddizioni tra il goal e i suoi raffinamenti. Quindi non accade che:
$$\{G_1, G_2, \ldots, G_n, Dom\} \models \texttt{false}$$
- **Minimalità**: se viene tolto anche solo un sotto-goal il goal principale non può essere raggiunto.
$$\{G_1, G_2, \ldots, G_{j-1}, G_{j+1}, \ldots, G_n\} \not\models G$$

7.3.3 Alberi di raffinamento

Requisito

Un requisito è un goal che non può essere raffinato ulteriormente.

Quando ho un goal che è responsabilità di un solo agente, allora posso fermare il raffinamento.

7.3.4 Potenziali conflitti

La presenza di contraddizioni potrebbe far emergere nuovi requisiti, rappresentando quindi delle temporanee lacune nella modellazione, non essendo quindi un pattern da evitare. È piuttosto importante documentare questi conflitti per poterli risolvere in seguito, attraverso un artefatto.

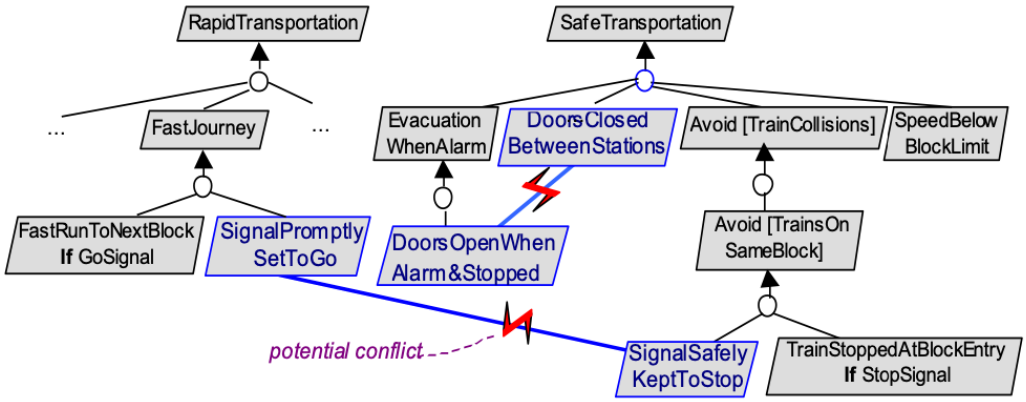


Figura 7.3.3: Rappresentazione di un conflitto

7.4 OR-decomposition

Nella **OR-refinement** il goal viene suddiviso in sotto-goal che devono essere soddisfatti almeno uno per poter soddisfare il goal principale.

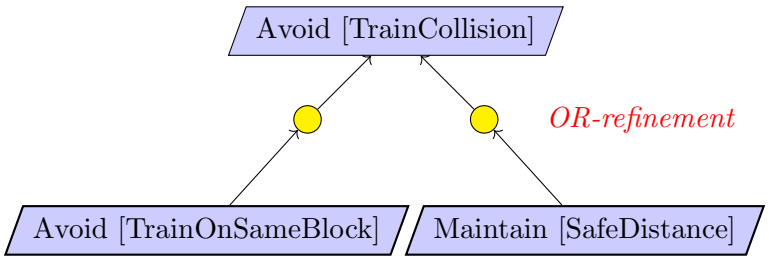


Figura 7.4.1: Rappresentazione di una raffinamento di tipo OR

7.4.1 Assegnamento di responsabilità alternativo

Nel caso in cui un goal possa essere raggiunto da più agenti diversi, in cui ognuno di essi può raggiungere il goal in modo indipendente, allora si può utilizzare un raffinamento di tipo `OR`.

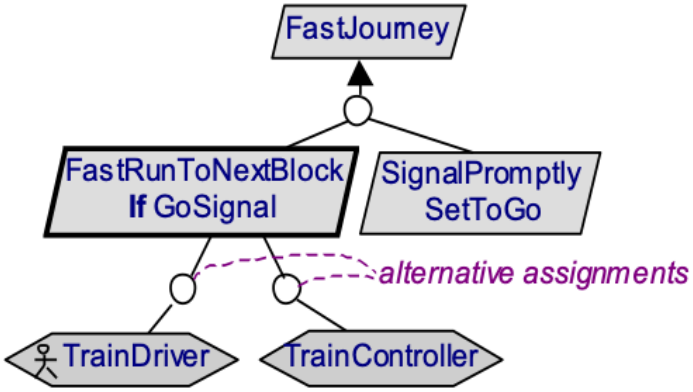


Figura 7.4.2: Rappresentazione di un assegnamento di responsabilità alternativo

È opportuno in fase di modellazione segnalare queste alternative che verranno discusse in futuro sulla base di costi, benefici e rischi.

7.5 Annotazioni

Possiamo aggiungere annotazioni e commenti per specificare ulteriori dettagli liberamente in modo da enfatizzare o cristallizzare il significato.

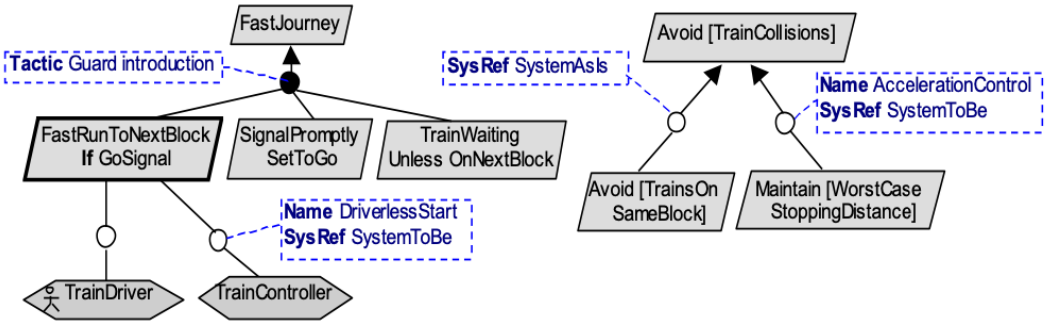


Figura 7.5.1: Rappresentazione di un'annotazione

7.6 Euristiche per l'individualizzazione dei goal

- Elicitazione dei goal preliminare.
  - Analizzare gli obiettivi correnti del sistema *as-is*.

  - Cercare keyword all'interno del materiale elicitato.
  - Cercare all'interno delle tassonomie di goal.
- Identificare i goal lungo i branch di raffinamento.
  - Farsi le domande "Perché?" (*astrazione*), "Come?" (*raffinamento*).
  - Utilizzare gli scenari.
  - Dividere le responsabilità tra gli agenti.
  - Identificare i goal da i pro e contro.
  - Dagli *Achive goal* posso identificare i *Maintain goal*.
- Non confondere i goal con le operazioni.
- Non confondere `AND`-refinement con `OR`-refinement.
- Evitare ambiguità.

Capitolo 8

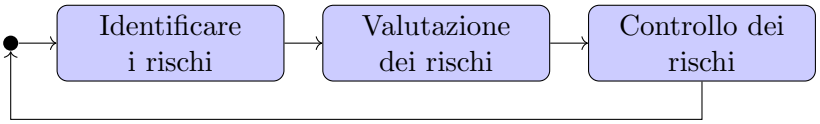
Analisi dei rischi

Rischio

Un rischio è una probabilità che un goal non venga raggiunto.

La funzionalità del sistema software viene a mancare, facendo si che il sistema raggiunga dei risultati parziali. I rischi vanno modellati già nella fase di modellazione dei goal, in modo da individuare i goal che possono essere compromessi e come mitigare questi rischi.

Già nella fase di modellazione dei requisiti abbiamo affrontato la risk analysis, identificando le fasi relative ai rischi.



La mancata individuazione dei rischi è una delle principali cause di fallimento dei progetti software.

I rischi possono essere rappresentati attraverso un parallelogramma, inclinato nella direzione opposta rispetto ai goal.

Figura 8.0.1: Rappresentazione di un rischio

8.1 Goal ostruiti dai rischi

Quando modelliamo i goal rischiamo di modellare solamente il funzionamento corretto del sistema, senza considerare i rischi che possono compromettere il raggiungimento di questi goal.

Un **ostacolo** insieme ad una proprietà di dominio implica che il goal non può essere raggiunto.

$$\{G, Dom\} \models \neg G$$

L'insieme ideale di ostacoli di G dovrebbe essere la negazione di tutti i goal che permettono di raggiungere G.

Anche gli ostacoli, come i goal possono essere mappati in una tassonomia per poter identificare i rischi in modo più semplice.

8.2 Modellazione degli ostacoli

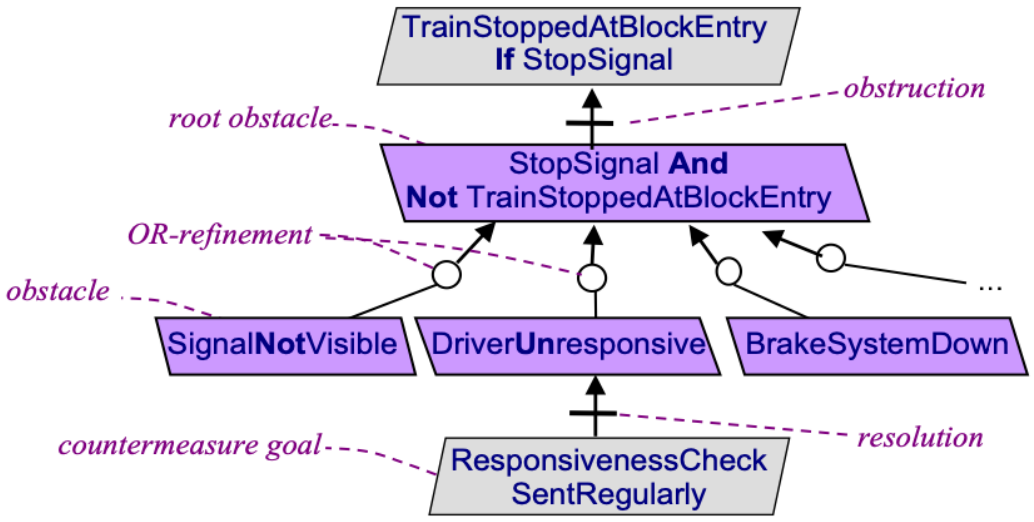


Figura 8.2.1: Rappresentazione di un ostacolo

La modellazione è simile a quella dei goal, dove per rappresentare un ostacolo che potrebbe compromettere il raggiungimento di un goal, si utilizza un *link di ostruzione*. Anche qui possiamo usare la `AND`-decomposition e la `OR`-decomposition per raffinare gli ostacoli.

Per rappresentare una possibile soluzione ad un ostacolo si può usare un *link di risoluzione* che collega un goal ad un ostacolo.

8.2.1 Raffinamento dei rischi

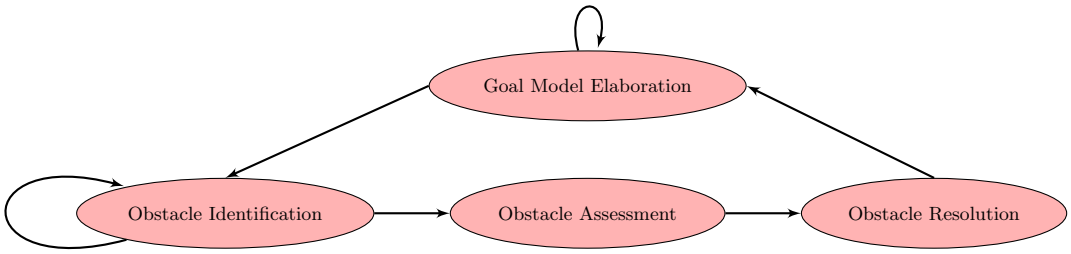
La struttura dei goal è molto simile alla struttura dei rischi, quindi, molto spesso, si può raffinare un rischio in modo simile a come si raffina un goal.

Anche per gli ostacoli possiamo identificare delle annotazioni per specificare ulteriori dettagli.

8.3 Analisi degli ostacoli per migliorare la robustezza del sistema

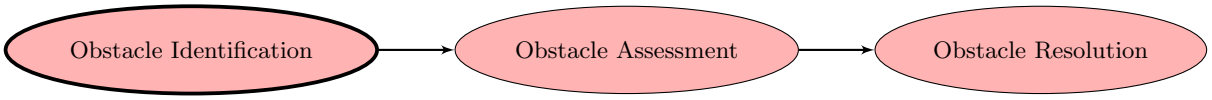
I nostri obiettivi principali sono:

- Identificare quanti più ostacoli possibili.
- Comprendere la probabilità di occorrenza di questi ostacoli.
- Comprendere l'impatto di questi ostacoli.



Di fatto prima di tutto si identificano gli ostacoli, si esegue un assessment dove si capisce la probabilità di occorrenza e l'impatto di questi ostacoli, e infine si risolvono questi ostacoli, procedendo poi con la modellazione dei goal.

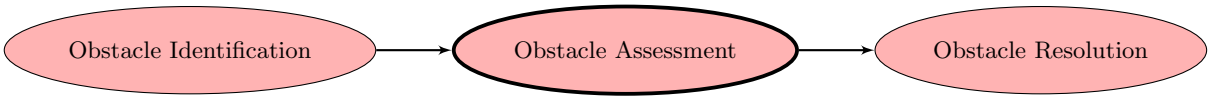
8.3.1 Identificare gli ostacoli



per prima cosa si parte dai goal di alto livello, il primo step è identificare i **root obstacle**, ovvero gli ostacoli che impediscono il raggiungimento di un goal. Si procede poi con la `AND`-decomposition e la `OR`-decomposition per identificare tutte le possibili cause che possono impedire il raggiungimento di un goal, fin tanto che non si raggiungono ostacoli atomici.

Per negare un goal *top level* utilizziamo la negazione con la legge di De Morgan.

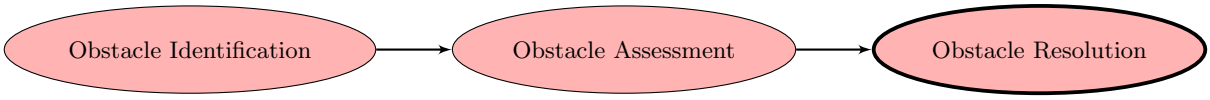
8.3.2 Valutare gli ostacoli



Per affrontare questa fase, solitamente, si coinvolgono degli esperti di dominio in cui si lavora, in modo da capire la probabilità di occorrenza e l'impatto di questi ostacoli.

per stimare la probabilità di un ostacolo di stima la probabilità di occorrenza di ogni sotto-ostacolo e si calcola la probabilità di occorrenza dell'ostacolo come il massimo tra i sotto ostacoli nel caso di `OR`-decomposition, mentre nel caso di `AND`-decomposition si calcola la probabilità di occorrenza minima tra i sotto-ostacoli.

8.3.3 Risolvere gli ostacoli



Per risolvere un ostacolo si adottano delle contromisure identificando dei goal che permettono di superare l'ostacolo. Questi goal vengono collegati all'ostacolo tramite un link di risoluzione. Per ogni ostacolo si può avere più di una soluzione, in questo caso occorrerà considerare le alternative e selezionare la migliore basandosi su costi, benefici e rischi.

Esplorare alternative

- **Sostituzione del goal**: nel caso in cui nel goal top-level ci siano diverse alternative per raggiungere il goal, si possono esplorare queste alternative per capire quale sia la migliore anche in termini di rischi. Se una delle alternative contiene un rischio rispetto ad un'altra, allora si può considerare di scartare questa alternativa.
- **Sostituzione di agente**: in certe situazioni potrei trovarmi di fronte alla necessità di rivedere l'agente utilizzato per raggiungere il goal, in quanto potrebbe rimuovere la presenza di un ostacolo.
- **Indebolire il goal**: se il goal, così come è stato definito, è troppo difficile da raggiungere, si può considerare di indebolire il goal, in modo da renderlo più facile da gestire e da raggiungere.

  `Maintain [TrafficControllerOnDutyOnSector]`

  Tale goal viene ostruito da `NoSectorControllerOnDuty`, quindi si può considerare di indebolire il goal in modo da renderlo più facile da raggiungere.

  `TrafficControllerOnDutyOnSector or WarningToNextSector`

- **Prevenzione di ostacoli**: definire un goal che cerca di porre rimedio ad un ostacolo.

  `AccelerationCommandCorrupted` $\rightarrow$ `Avoid [AccelerationCommandCorrupted]`

- **Ripristino di goal**: rafforzando la condizione target come occorrenza di un ostacolo.

  `if O then sooner-or-later TargetCondition`

- **Riduzione di ostacoli**: definiamo tecnicamente un nuovo goal che cerca di rimediare un singolo ostacolo di basso livello.

- **Mitigazione di ostacoli**: introduciamo nuovi goal per mitigare le conseguenze di un ostacolo. Distinguiamo quindi due diverse strategie di mitigazione di ostacoli:
  - **Mitigazione debole**: accettare l'ostacolo e ridefinire il goal in modo tale che una versione più debole del goal venga realizzata.
  - **Mitigazione forte**: si raggiunge il goal di alto livello anche se ostruito.

Capitolo 9

Modellazione degli oggetti concettuali con i Diagrammi delle Classi

Sostanzialmente, questo modello rappresenta una vista **strutturale** del sistema che vogliamo modellare (*to-be*) o il sistema attuale che vogliamo analizzare (*as-is*).

L'obiettivo è modellare i concetti, le strutture e i collegamenti tra i vari oggetti che rappresentano i requisiti del sistema.

La rappresentazione avviene tramite i **Diagrammi delle Classi** `UML`, declinandolo verso l'uso di una terminologia differente. Questo perché nella terminologia *goal oriented*, quando parliamo di oggetti e classi, non ci riferiamo alla metodologia orientata agli oggetti, bensì modelliamo concetti del mondo reale.

L'oggetto *treno*, o *controllore* non si riferiscono al software, ma al mondo dei requisiti che stiamo modellando e, dato che non modelliamo il software, **le classi non hanno operazioni**, ma solo **attributi** per modellare le proprietà degli oggetti del mondo che stiamo modellando.

9.1 Oggetto concettuale

Oggetto concettuale

Un oggetto concettuale è un insieme di istanze che godono di queste proprietà:

- **sono identificabili in maniera distinta**
- **possono essere enumerati in ogni stato del sistema**
- **condividono caratteristiche comuni**
- **possono essere differire da uno stato all'altro**

Rappresentiamo lo stato di un oggetto di rappresenta con la lista di tuple $x_i \to v_i$, dove:

- x_i è l'attributo dell'oggetto, associazione;
- v_i è il valore dell'istanza.

Esempio

Per l'istanza `tr` dell'oggetto `Treno`:

$$(tr.Speed \to 100, tr.Loc \to 9.25, tr.Dest \to MI, On \to (tr, block13), At \to (tr, st1))$$

La relazione tra gli oggetti e le istanze di oggetti è che un object instance è un'istanza di un oggetto, rappresentato come `instanceOf(o, Ob)`. A differenza dell'object oriented, nel corso dell'evoluzione del sistema, un oggetto può diventare un'istanza di un altro oggetto. Un'istanza più essere membro di più oggetti (**tipizzazione multipla**).

9.1.1 Classi e istanze correnti

Nella rappresentazione diagrammatica, per ogni oggetto bisogna specificare obbligatoriamente la motivazione per cui un'istanza è istanza di un oggetto.

9.2 Entità

Entità

Un'entità è un oggetto **autonomo** e **passivo**, non controllano il funzionamento e lo stato di altri oggetti. L'esistenza di tali oggetti è indipendente dall'esistenza di altri oggetti. Tali oggetti vengono rappresentati attraverso le **classi** `UML`.

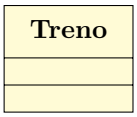


Figura 9.2.1: Esempio di entità

Necessitano della annotazione **Def**, che specifica la definizione dell'oggetto.

9.3 Associazioni e molteplicità

Associazioni

Un'associazione è un oggetto concettuale che collega altri oggetti, ed ogni oggetto ha un ruolo specifico. Tali oggetti si rappresentano mediante le associazioni `UML`.

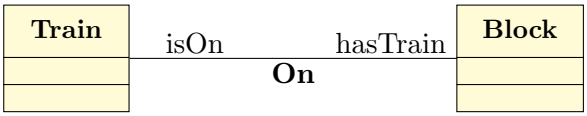


Figura 9.3.1: Esempio di associazione

Istanza di associazione

Un'istanza di associazione che collega due istanze di oggetti.

In linea di principio possono riferirsi ad un numero arbitrario di oggetti, avendo un'arietà maggiore di due oggetti.

9.3.1 Molteplicità

La molteplicità è un vincolo che specifica il numero di oggetti che possono essere coinvolti in un'associazione. La molteplicità può essere:

- **0..1** - 0 o 1 oggetti
- **0..*** - 0 o più oggetti
- **1..*** - 1 o più oggetti
- **1..1** - 1 oggetto

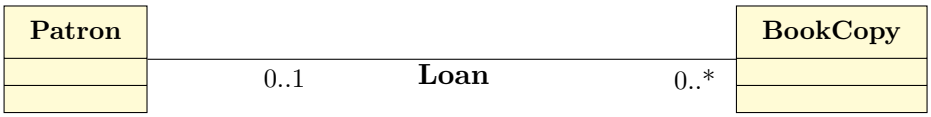


Figura 9.3.2: Esempio di associazione con molteplicità

Possono anche essere riflessive, partendo da un oggetto e ritornando allo stesso oggetto.

9.4 Eventi

Eventi

Un evento è un oggetto istantaneo, gli oggetti che sono istanze di eventi soddisfano la proprietà `Occurs(Ev)`. Gli eventi sono rappresentati tramite gli le classi `UML`.

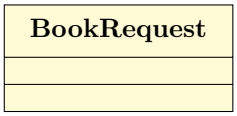


Figura 9.4.1: Esempio di evento

9.5 Agenti

Agenti

Un agente è un oggetto attivo e autonomo, che hanno funzionamento autonomo e possono **controllare** o **modificare** lo stato di altri oggetti, causando transizioni di stato. Gli agenti sono rappresentati tramite le classi `UML`.

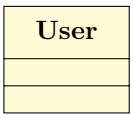


Figura 9.5.1: Esempio di agente

9.6 Attributi

Attributi

Gli attributi sono proprietà degli oggetti che modelliamo, e sono rappresentati mediante gli attributi delle classi `UML`.

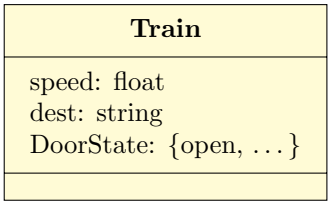


Figura 9.6.1: Esempio di attributi

Anche le associazioni, in quanto oggetti, possono avere attributi.

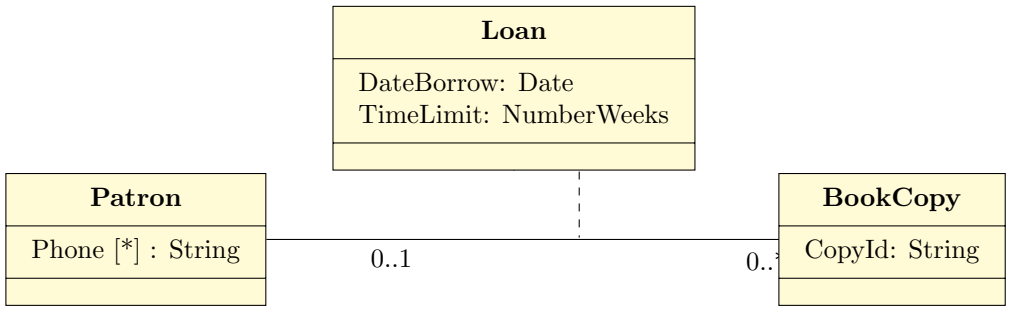


Figura 9.6.2: Esempio di associazione con molteplicità

9.7 Specializzazione

Specializzazione

La specializzazione è un'associazione tra una classe generale e una classe specializzata, dove la classe specializzata eredita gli attributi della classe generale. La specializzazione è rappresentata tramite l'ereditarietà delle classi `UML`.

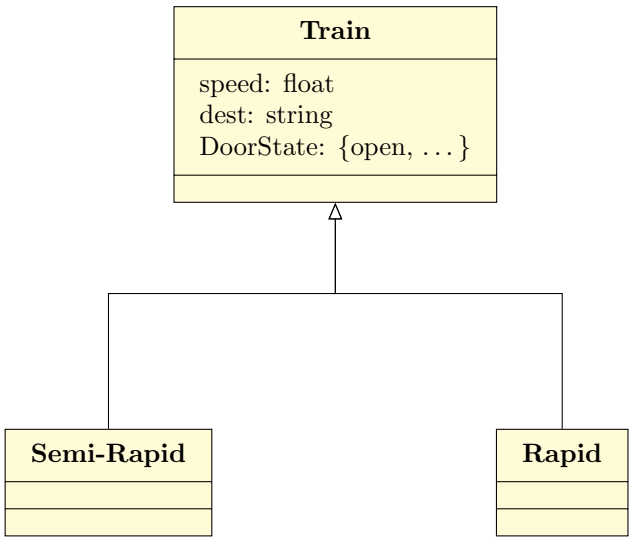


Figura 9.7.1: Esempio di specializzazione

9.7.1 Inibizione dell'ereditarietà

In questo modo specializzare un oggetto, definendo un attributo in maniera più specifica rispetto all'attributo generale (*si tratta di una ridefinizione*).

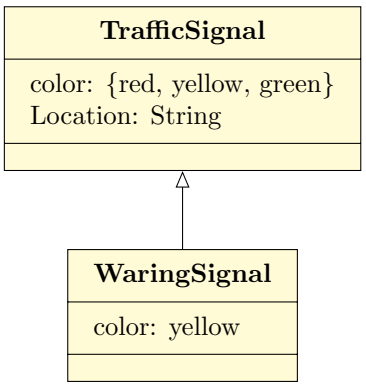


Figura 9.7.2: Esempio di specializzazione

9.7.2 Ereditarietà multipla

Un oggetto può essere specializzato in più oggetti, ereditando gli attributi di più classi.

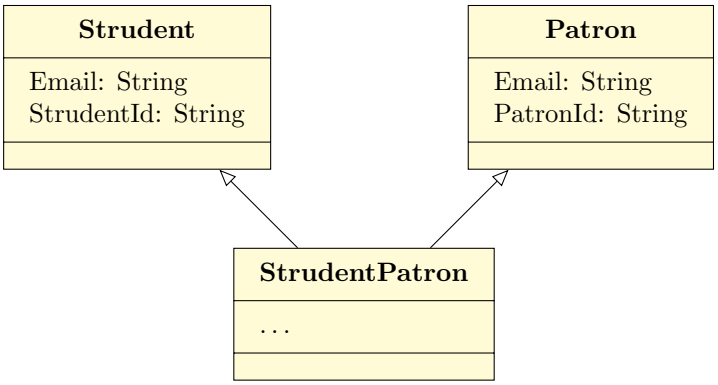


Figura 9.7.3: Esempio di specializzazione

Possiamo avere dei conflitti, in questo caso, avendo il campo `Email` in entrambe le classi, dobbiamo rinominare il campo in uno dei due oggetti.

Benefici della specializzazione

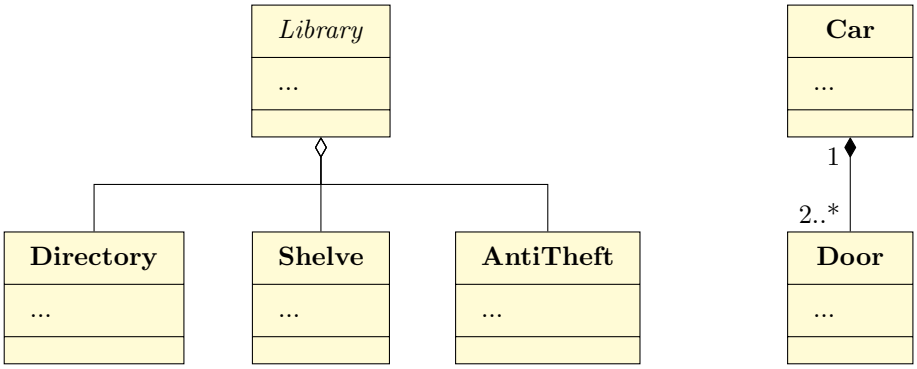
Una struttura basata su specializzazione permette di:
- Avere un modello più semplice evitando la duplicazione di attributi.
- L'eliminazione di attributi è possibile su uniche entità evitando disallineamenti.

9.8 Aggregazione

Aggregazione

L'aggregazione è una composizione di diversi oggetti, dove ogni oggetto rappresenta una parte. In questa relazione, l'oggetto aggregato può essere considerato un contenitore dei suoi componenti, ma i componenti possono esistere indipendentemente dall'aggregato.

L'aggregazione è applicabile alle entità, agenti, eventi e associazioni. Nelle relazioni di aggregazione, le molteplicità possono essere attaccate ai collegamenti parte-aggregazione, suggerendo che una parte può essere condivisa tra più aggregati o esistere in molteplicità variabili.



9.8.1 Differenze tra Aggregazione e Composizione

Mentre l'aggregazione permette che le parti esistano indipendentemente dall'aggregato, la composizione è una forma più forte di aggregazione dove le parti non hanno un'esistenza indipendente dall'aggregato. Se un aggregato viene distrutto, tutte le sue parti vengono distrutte con esso. Questo legame è utile per rappresentare relazioni vita o morte tra componenti, come tra un corpo e gli organi interni.

9.8.2 Esempi di Aggregazione

Un esempio comune di aggregazione potrebbe essere una libreria e i libri che contiene. La libreria è l'aggregato, e i libri sono le parti. I libri possono esistere al di fuori della libreria, suggerendo che la libreria è solo un contenitore per i libri, non una necessità per l'esistenza dei libri.

Un altro esempio può essere un'azienda (*aggregato*) e i suoi dipartimenti (*parti*). Mentre i dipartimenti sono funzionalmente parte dell'azienda, possono tecnicamente operare in qualche misura indipendentemente dall'azienda stessa.

9.9 Euristiche per la Modellazione degli Oggetti

Nel processo di modellazione degli oggetti, è fondamentale seguire determinate euristiche per assicurare una progettazione chiara e funzionale. Di seguito sono elencate le principali linee guida da considerare:

- **Revisione delle Specifiche**: È cruciale rivedere tutte le specifiche degli obiettivi e delle proprietà del dominio contenute nel modello degli obiettivi per garantire che tutte le necessità siano adeguatamente rappresentate e comprese.
- **Assegnazione degli Obiettivi**: Per l'implementazione degli obiettivi nel software da realizzare, è essenziale introdurre rappresentazioni condivise degli oggetti dell'ambiente riferiti dagli obiettivi.
- **Definizione degli Attributi**: Un elemento è da considerare un attributo se è una funzione che restituisce un valore unico, le sue istanze non necessitano distinzione, non si

prevede di aggiungervi attributi o associazioni, e non si intende specializzarne l'intervallo di valori.

- **Oggetti Concettuali negli Obiettivi**:
  - Gli oggetti concettuali definiti da uno stato unico (esempio: *StartTrain*) implicano eventi.
  - Gli oggetti che attivano e controllano comportamenti di altre istanze (esempio: *DoorsActuator*) agiscono come agenti.
- **Attributi nelle Associazioni**: Un attributo deve essere collegato a un'associazione se caratterizza tutti gli oggetti partecipanti, sia esplicitamente che implicitamente.
- **Link Strutturali e Associativi**: Considerare la creazione di un'associazione se esiste un link strutturale tra oggetti compositi e componenti che soddisfa specifiche condizioni operative o strutturali.
- **Specializzazioni e Generalizzazioni**: È importante identificare specializzazioni e generalizzazioni attraverso l'analisi di attributi comuni, associazioni e invarianti di dominio, contribuendo alla creazione di un modello più organizzato e scalabile.
- **Uso di Associazioni invece di Puntatori**: Evitare l'utilizzo di "puntatori" agli altri oggetti come attributi e preferire l'uso di associazioni binarie.
- **Nomi e Definizioni Chiare**: Utilizzare nomi suggeriti e chiari per oggetti e attributi, evitando ambiguità e assicurando una definizione precisa e dettagliata.

Capitolo 10

Modellare gli agenti di sistema e le loro responsabilità

Avendo parlato ampiamente dei goal, parliamo degli agenti, ovvero chi ha la responsabilità che i goal vengano raggiunti. Non si parla solo di **responsabilità**, ma anche di **capacità**, ovvero quali sono gli agenti che sono in grado di raggiungere i goal, chi ha interesse a raggiungerli e quelli a cui verrà assegnata la responsabilità del raggiungimento del goal.

10.1 Caratteristiche degli agenti di sistema

10.1.1 Capacità degli agenti

Gli agenti possono *monitorare* o *controllare* delle grandezze del sistema, ovvero possono avere la capacità di *osservare* o *modificare* lo stato del sistema.

Figura 10.1.1: Esempio di agenti che monitorano e controllano delle grandezze del sistema.

- In input all'agente abbiamo grandezze che l'agente monitora.
- In output all'agente abbiamo grandezze che l'agente controlla.

Di fatto, quello che gli agenti possono monitorare non sono solo dati, ma ogni tipologia di oggetto, ad esempio un agente potrebbe monitorare un campo di un oggetto, la presenza di associazioni tra oggetti, ecc.

10.1.2 Responsabilità degli agenti

È importante che una variabile sia controllata da al più un agente, altrimenti si rischiano conflitti di responsabilità.

Responsabilità di un agente

La responsabilità di un agente viene rappresentata con un **responsibility link**, ovvero un collegamento tra l'agente e il goal che l'agente deve raggiungere.

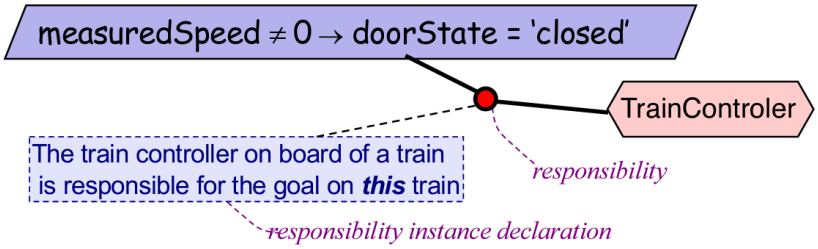


Figura 10.1.2: Esempio di responsabilità di un agente.

Assegnamento di responsabilità e capacità degli agenti

L'assegnamento delle responsabilità deve essere subordinato al fatto che l'agente possieda le capacità di monitorare e controllare le grandezze necessarie per il raggiungimento del goal. Per farlo si definisce una lista di transizioni di stato che l'agente può monitorare e controllare in modo che il goal sia soddisfatto.

Un goal potrebbe essere non realizzabile per l'agente quando delle variabili importanti non possono essere monitorate o controllate dall'agente stesso. Se ci sono variabili di stato che vengono valutate in stati futuri, quando il goal esprime delle condizioni istantanee, allora il goal non è realizzabile. Un'altra condizione si verifica quando il goal non è soddisfacibile sotto determinate condizioni, ovvero il goal non è realizzabile in alcune situazioni.

10.1.3 Operazioni degli agenti

Per raggiungere gli obiettivi, gli agenti devono prendere delle decisioni intraprendendo delle operazioni. Fare un'operazione vuol dire che l'agente deve impostare o leggere delle variabili negli oggetti che controlla o monitora. Deve farlo nelle condizioni in cui il sistema opera.

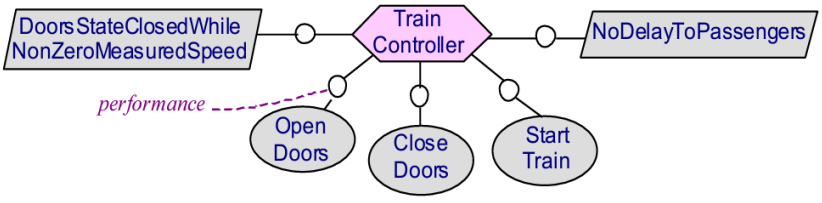


Figura 10.1.3: Esempio di operazioni di un agente.

Il collegamento tra l'agente e l'operazione che l'agente deve compiere viene rappresentato con un **performance link**.

10.1.4 Desideri dell'agente

Un agente umano potrebbe avere dei desideri, ovvero degli obiettivi personali che non sono necessariamente in linea con i goal del sistema.

Un esempio di desiderio di un agente umano potrebbe essere "*Un Patron vuole che il periodo di prestito sia lungo*".

Ci forniscono un modo per guidare l'assegnamento delle responsabilità, in quanto un agente umano potrebbe avere un desiderio che non è in linea con i goal del sistema. Pensare ai desideri degli agenti può quindi guidare già in fase di assegnamento delle responsabilità.

10.1.5 Conoscenza e credenze degli agenti

È possibile modellare sia la conoscenza che le credenze degli agenti rispetto una **memoria locale** che l'agente possiede rispetto all'ambiente reale.

- Un fatto F è creduto dall'agente se F è nella memoria locale dell'agente.
- Un fatto F è conosciuto dall'agente se F è nella memoria locale dell'agente e F è vero, proprietà posseduta dall'ambiente o dal sistema.

Tale considerazione è importante perché ci permette di fare delle considerazioni sulla **sicurezza**.

10.1.6 Dipendenze tra agenti

Una dipendenza tra due agenti si verifica quando un agente è responsabile del raggiungimento di un goal, ma un altro agente, per aggiungere un determinato goal deve aspettare che il primo agente abbia raggiunto il suo goal.

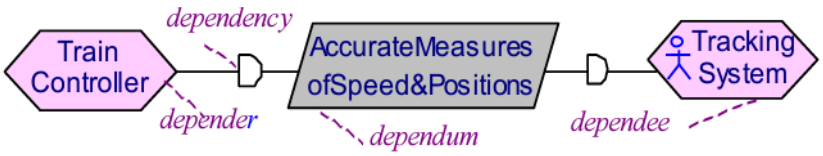


Figura 10.1.4: Esempio di dipendenze tra agenti.

Tali dipendenze possono propagarsi a catena. In sistemi critici è opportuno non avere catene troppo lunghe.

10.2 Rappresentazione della modellazione degli agenti

10.2.1 Agent diagram

Agent diagram

L'agent diagram è un diagramma che ha come prospettiva principale gli agenti e permette di rappresentare le *responsabilità*, le *capacità* e le *operazioni* degli agenti.

Possiamo, come nel caso dei goal, possiamo avere degli `OR`-assignments, ovvero un goal può essere assegnato a più agenti e il goal è soddisfatto se almeno uno degli agenti ha raggiunto il goal. Ricordiamoci che ogni goal deve essere assegnato ad un solo agente, per questo motivo non possiamo avere `AND`-assignments.

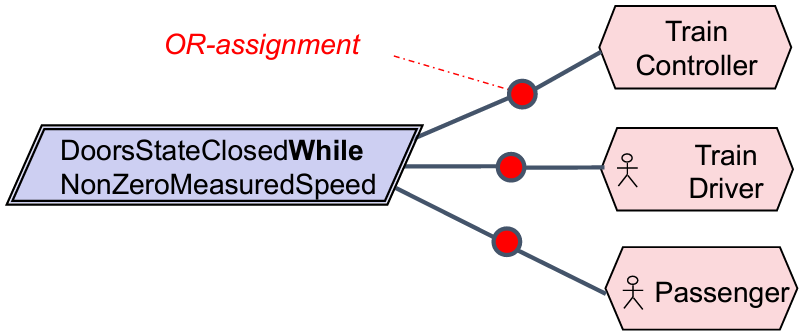


Figura 10.2.1: Esempio di `OR`-assignment in un agent diagram.

10.2.2 Context diagram

Context diagram

Il context diagram è una vista che rappresenta solamente la relazione tra variabili *controllate* e *monitorate* dagli agenti.

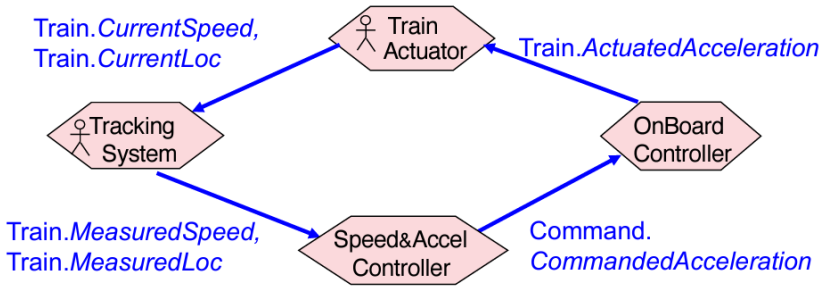


Figura 10.2.2: Esempio di context diagram.

Un agente potrà prendere decisioni in relazione alle informazioni in termine di variabili controllate e monitorate.

10.2.3 Dependency diagram

Dependency diagram

Il dependency diagram è una vista che rappresenta le *dipendenze* tra agenti.

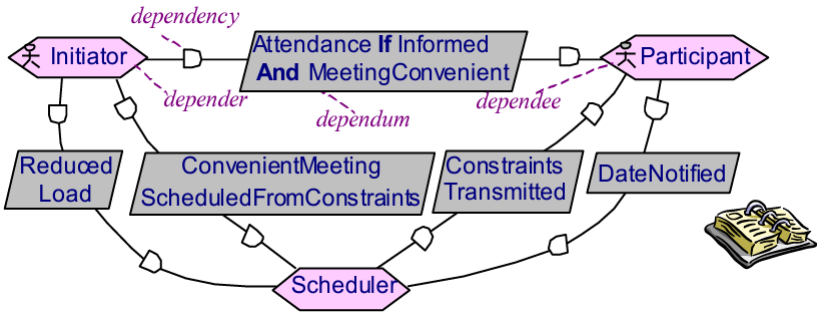


Figura 10.2.3: Esempio di dependency diagram.

10.3 Raffinamento e astrazione degli agenti

Gli agenti possono essere raffinati in sottoagenti, ovvero possiamo avere una decomposizione gerarchica degli agenti. Questo permette di avere una visione più dettagliata degli agenti e delle loro responsabilità. Laddove un goal di alto livello è assegnato ad un agente di alto livello, questo può essere raffinato in sottoagenti che si occupano di raggiungere i goal di basso livello rispetto ad una mappatura 1 a 1.

È importante ricordare che gli agenti sono ruoli e non persone o entità fisiche.

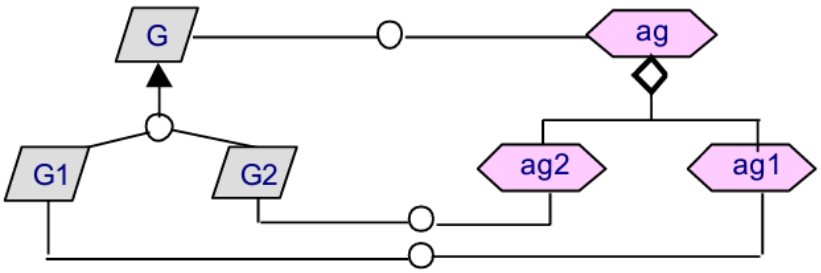


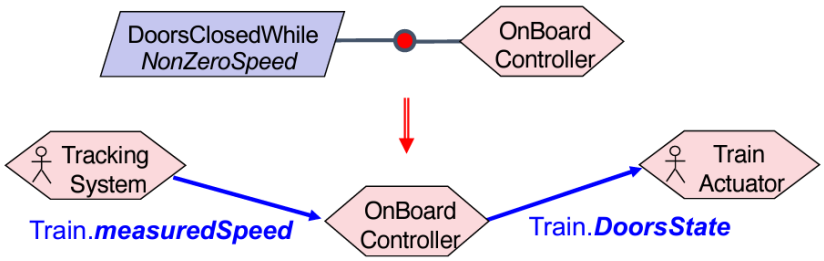
Figura 10.3.1: Esempio di raffinamento di agenti.

10.4 Euristiche per la modellazione degli agenti

- **Identificazione degli agenti**: quando si legge la definizione di un goal, implicitamente nella definizione c'è il concetto di variabile controllata o monitorata.
- Quando leggiamo le **definizione** possiamo vedere le azioni degli agenti.
- Quando assegniamo una responsabilità ad un agente umano bisogna valutare anche i **desideri** dell'utente (*i goal di sicurezza informatica devono essere assegnati a persone che hanno a cuore la sicurezza del sistema, quindi in contrasto con il desiderio dell'utente finale*).
- Non confondere agenti a livello di prodotto con gli stakeholder a livello di processo. Gli agenti sono dei **ruoli**, mentre gli stakeholder sono delle persone che intervisto, di fatto la stessa persona potrebbe avere ruoli diversi ed essere modellato in agenti diversi.
- **Assegnare responsabilità a componenti software**: per raggiungere un livello di automazione maggiore (*dipende e va valutato in base ai desideri*).
- **Raffinare gli agenti in linea con i goal**: in modo da raggiungere una granularità tale da poter assegnare responsabilità in modo chiaro.

10.5 Derivare i context diagram dai goal

Implicitamente i goal possono far riferimento a variabili controllate e monitorate, quindi possiamo creare i link che monitora la variabile monitorata e controlla la variabile controllata, iterando agente per agente.



Capitolo 11

Modellare le Operazioni

Per operazione intendiamo cosa fa il sistema per raggiungere i goal e come vengono rappresentate le operazioni.

Le operazioni riguardano le **operazioni attive**, ovvero le azioni che il sistema deve compiere per raggiungere i goal.

11.1 Operation model

Si tratta di una vista **funzionale del sistema** che rappresenta **quali** servizi saranno resi disponibili all'interno del sistema e **sotto quali condizioni** per il soddisfacimento dei goal.

La rappresentazione avviene sempre mediante l'ausilio dei diagrammi `UML` declinati ina maniera funzionale alla modellazione dei requisiti.

Gli obiettivi di tale diagramma sono molteplici, tra cui:

- Fornire un input agli sviluppatori per la specifica del software.
- Descrivere le procedure e i task all'interno dell'ambiente.
- Può essere un punto di partenza per identificare con quali dati interagirà il sistema o essere l'input per i tool di prototipazione o animazione dei requisiti.
- Possiamo utilizzarli per stimare i *function point*, ovvero capire quanto lavoro ci sarà da fare per l'implementazione delle varie componenti.
- Permette di creare un link tra tutte le funzionalità del sistema e i goal che si vogliono raggiungere.

11.2 Le operazioni

Operazione

Una **operazione** è una coppia di stati del sistema, di *input* e *output*. Un'operazione ci permette di cambiare lo stato del sistema.

Ricordiamo che uno stato è una tupla $x_i \to v_i$, dove x_i rappresenta l'attributo di una istanza di oggetto concettuale, mentre v_i rappresenta il valore che assume tale attributo.

Abbiamo diverse tipologie di variabili coinvolte nelle operazioni:

- Variabili di **input**: rappresentano i dati che il sistema riceve in input per poter eseguire un'operazione.
- Variabili di **output**: rappresentano i dati che il sistema restituisce in output dopo aver eseguito un'operazione.

Anche le operazioni possono avere **attributi** che ne specificano le caratteristiche.

Esempio

Consideriamo l'operazione *OpenDoors(tr)*, lo stato iniziale del sistema è

$$(tr.Speed \to 0, tr.DoorState \to' Closed')$$

mentre lo stato finale diventa:

$$(tr.Speed \to 0, tr.DoorState \to' Open')$$

L'obiettivo è di **operazionalizzare** i goal del goal model, ovvero permettere in maniera operativa di raggiungere i goal.

Le operazioni sono:

- **Deterministiche**: non hanno quindi un comportamento probabilistico.
- **Atomiche**: non possono essere suddivise in operazioni più piccole.
- **Possono essere applicate in modo parallelo**: possono essere eseguite in modo parallelo.

11.3 Rappresentare le operazioni

Le operazioni vengono rappresentate mediate l'ellisse come nel **Use Case Diagram**.

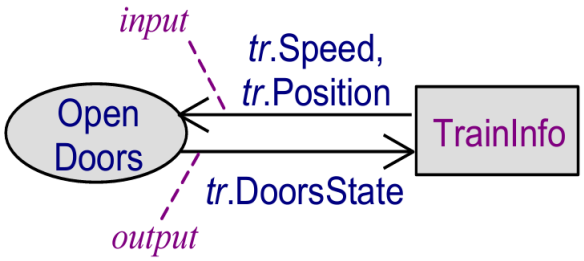


Figura 11.3.1: Rappresentazione di un'operazione

Potrebbero avere anche delle precondizioni e delle postcondizioni, all'interno di un commento UML, insieme a **input** e **output**.

Esempio

- **Operation**: TrainInfo
- **Input**: tr: TrainInfo / {Speed, Position}
- **Output**: tr: TrainInfo / {DoorState}

11.3.1 Pre e post condizioni di Dominio

Precondizioni

Le **precondizioni** sono le condizioni che devono essere verificate affinché l'operazione possa essere eseguita.

Postcondizioni

Le **postcondizioni** sono le condizioni che devono essere verificate affinché l'operazione sia stata eseguita correttamente.

Rispettivamente vengono descritte con **DomPre** e **DomPost**. Entrambe sono degli statement **descrittivi**.

11.3.2 Esecuzione di un'operazione

Sostanzialmente il link tra le operazioni e gli agenti è un link di **performing**.

Gli agenti attuano le operazioni per raggiungere i goal di cui è responsabile.

Ci sono delle regole di consistenza che devono però essere rispettare:

- Le variabili di input e output devono essere grandezze *monitorabili* e *controllabili* dagli agenti.

- Ogni operazione può essere intrapresa da un solo agente (*problematiche di responsabilità*).

11.4 Operalizzazione dei goal

La metodologia goal oriented permette di **operazionalizzare** i leaf goal, e questo è il motivo primario per cui abbiamo la destrutturazione dei goal di alto livello in goal di basso livello.

11.4.1 Condizioni Pre, Post, Trigger per il soddisfacimento dei goal

Abbiamo un link di operalizzazione che collega un'operazione a un requisito (*leaf goal*). Anche in questo caso tale link può avere degli attributi, tra cui:

- **ReqPre**: le condizioni *necessarie* affinché l'operazione possa soddisfare il goal (**deve**).
- **ReqTrig**: le condizioni *sufficienti* affinché l'operazione possa soddisfare il goal (**può**).
- **ReqPost**: le condizioni che devono essere verificate affinché l'operazione abbia soddisfatto il goal.

Esempio completo

- **Operazione**: OpenDoors
  - **Def**: Operazione che controlla l'apertura di tutte le porte di un treno.
  - **Input**: `tr`: TrainInfo / {Speed, Position}
  - **Output**: `tr`: TrainInfo / {DoorState}
  - **DomPre**: Le porte del treno `tr` sono chiuse.
  - **DomPost**: Le porte del treno `tr` sono aperte.
  - **ReqPre**: Il velocità del treni `tr` è zero.
  - **ReqPre**: il treno `tr` è in stazione.
  - **ReqTrig**: Il treno `tr` si è appena fermato in stazione.

> Generalmente un leaf goal è operazionalizzato da più operazioni, e delle operazioni possono operazionalizzare più leaf goal.

Quando succede, tutte le precondizioni e le postcondizioni devono essere implicitamente congiunte. Se una precondizione di dominio è vera, deve essere applicata appena la condizione di trigger diventa vera. Se una precondizione di dominio può essere applicata quando tutte le condizioni di operalizzazione sono vere.

Consistenza

Se una precondizione di dominio è vera e almeno una delle condizioni di trigger è vera, allora tutte le preconditioni di operalizzazione devono essere vere.

11.4.2 Impegno degli agenti

Per ogni goal sotto la responsabilità di un agente, e per ogni operazione che operalizza un goal, abbiamo che l'agente deve garantire che l'operazione sia applicata quando:

- Quando le operazioni **DomPre** sono vere.
- Appena le operazioni **ReqTrig** diventano vere.
- Solo se le operazioni **ReqPre** sono vere.
- In Modo tale che le operazioni **DomPost** e **ReqPost** siano vere.

Di fronte agli agenti ci troviamo di fronte a un **non determinismo**, poiché potremmo trovarci di fronte a due categorie di agenti:

- **Eager**: l'agente fa partire l'operazione non appena le condizioni **ReqPre** sono vere.
- **Lazy**: l'agente aspetta che le condizioni **ReqTrig** siano vere, quando si trova costretto.

Ci troviamo quindi di fronte a un certo grado di libertà per gli agenti, che possono decidere quando far partire le operazioni e una sorta di concorrenza tra le gli agenti stessi.

11.4.3 Operalizzazione dei goal e soddisfacimento

Consideriamo l'operalizzazione di un goal in concomitanza con la soddisfacibilità dei goal, l'operation model e il goal model possono argomentare sugli obiettivi di alto livello che vengono raggiunti. Di fatto possono essere percorsi:

- All'indietro: capire perché un goal è stato raggiunto.
- In avanti: capire come un goal può essere raggiunto.

11.5 Rappresentazione semantica

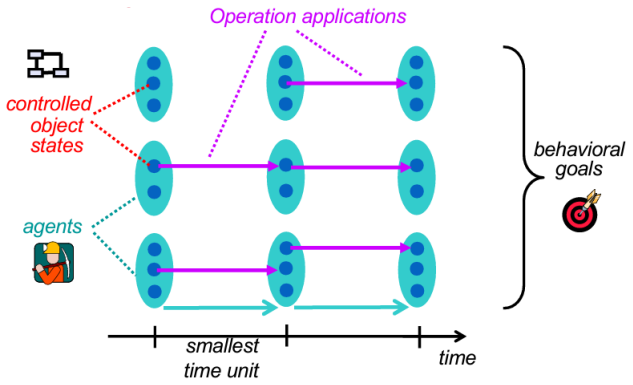


Figura 11.5.1: Rappresentazione semantica delle operazioni

Le operazioni si applicano nel corso del tempo attuano cambiamenti di stato all'interno del sistema, controllando lo stato degli oggetti.

11.5.1 Use Case Diagram

Nel tempo possono non far nulla o modificare lo stato di uno o più agenti che può controllare.

Le operazioni possono essere rappresentati mediante **use case diagram** rappresentando le operazioni e gli agenti.

Ho delle interazioni con:

- L'agente che controlla le operazioni in input
- L'agente che monitora operazioni in output

Ci sono dei link tra le operazioni di **extend** e **include**.

- **Include**: l'operazione include un'altra operazione.
- **Extend**: l'operazione estende un'altra operazione.

Dobbiamo però ricordarci che le operazioni sono atomiche, non decomponibili in sotto operazioni, perciò questa tipologia di link non sarebbe da utilizzare.

Esempio

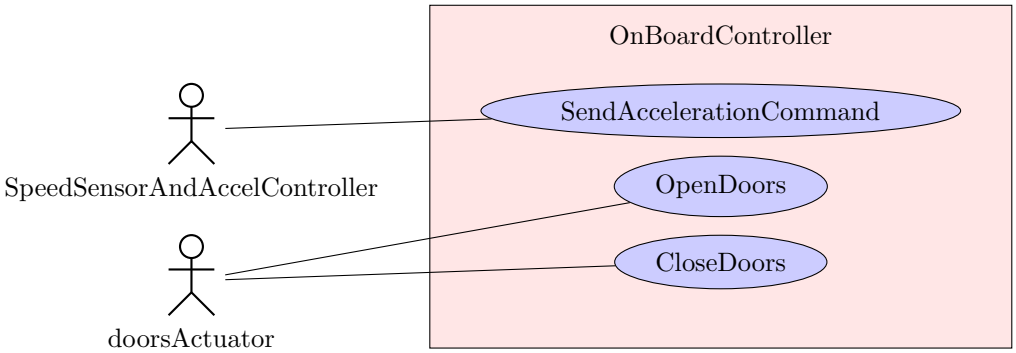
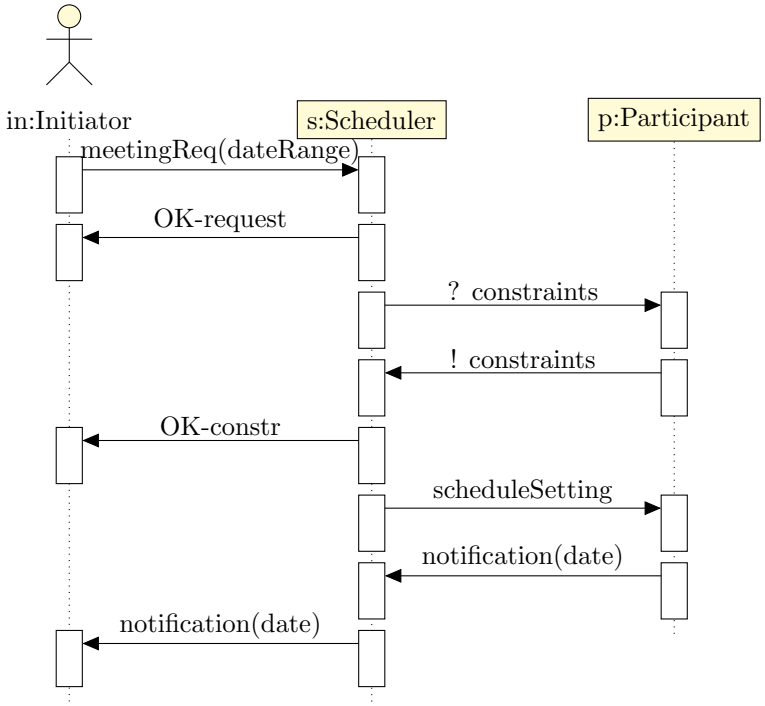


Figura 11.5.2: Rappresentazione semantica delle operazioni

11.6 Derivare operazioni dai goal in maniera fluida

- La prima modalità consiste nell'utilizzare i *fluents*, i fluents non sono altro che condizioni che definiscono l'avvio o il termine di una qualche operazione. Consideriamo *Le porte del treno sono chiuse se il treno è in movimento.* In questo caso distinguiamo `Moving` → `Closed`.
- Possiamo utilizzare le interazioni all'interno degli scenari. Gli scenari possono essere metodologie per l'espressione dei requisiti.



- Rafforzamento delle precondizioni e postcondizioni con condizioni richieste dagli obiettivi
  - **Identificare i permessi richiesti**: se l'effetto DomPost di un'operazione può violare un obiettivo G sotto condizione C
    - `ReqPre per` G: `non` C
    - **es.** OpenDoors: `ReqPre per "Moving → Closed": non Moving`
  - **Identificare le obbligazioni richieste**: se l'effetto DomPost di un'operazione è prescritto da un obiettivo G da mantenere ogni volta che la condizione C diventa `true`
    - `ReqTrig per` G: C
    - **es.** OpenDoors: `ReqTrig per "StoppedAtPlatform → Open": StoppedAtPlatform`
  - **Identificare gli effetti aggiuntivi richiesti**: se il DomPost di un'operazione non è sufficiente per garantire la condizione target T di un obiettivo G
    - `ReqPost per` G: `condizione mancante da` T
    - **es.** PlanMeeting: `ReqPost per ConvenientMeeting: data non in date escluse`

Capitolo 12

Comportamento del sistema

Rappresenta una **vista dinamica** del sistema, in cui si descrive il comportamento degli agenti in termini della sequenza temporale di stati e le variabili che essi controllano.

Sostanzialmente vogliamo rappresentare come il sistema, sotto il controllo degli agenti verrà guidato in due possibili modalità:

- Mediante una singola istanza di esecuzione
- Mediante classi di esecuzione

Entrambe le viste possono rappresentare il *system as-is* o il *system to-be*.

Tali viste possono avere molteplici utilizzi, infatti gli scenari possono essere utilizzati in fase di elicitazione dei requisiti, per definire i casi d'uso, per definire i test di sistema e per definire i test di accettazione. Gli scenari sono un veicolo per la fase di elicitazione dei requisiti, in quanto infatti possono far emergere domande sul perché determinate azioni devono essere intraprese.

Le state machine possono essere usate per supportare strumenti avanzati come le animazioni, model checking e generazione di codice.

12.1 Modellare singole istanze di comportamenti

Scenario

Lo scenario rappresenta una successione temporale di eventi tra le varie istanze di agenti che abbiamo disponibili nel nostro sistema.

È possibile considerare tali sequenze fra differenti agenti, o diverse istanze di agenti dello stesso tipo.

Le interazioni possono avere diverse direzioni:

- **Da** un'istanza di agente, che controlla l'evento.

- **Verso** un'istanza di agente, che monitora l'evento.

Gli scenari si contraddistinguono in:

- **Scenari positivi**: rappresentano il raggiungimento di un goal all'interno del sistema. Se di dividono a loro volta in:
  - **Scenari normali**: rappresentano il raggiungimento di un goal senza particolari ostacoli.
  - **Scenari anomali**: rappresentano il raggiungimento di un goal nel caso in cui si verifichino particolari ostacoli (*rimedio all'ostacolo*).
- **Scenari negativi**: rappresentano il verificarsi di un ostacolo all'interno del sistema, ovvero quello che non vogliamo che accada.

12.1.1 Gli scenari come diagrammi di sequenza UML

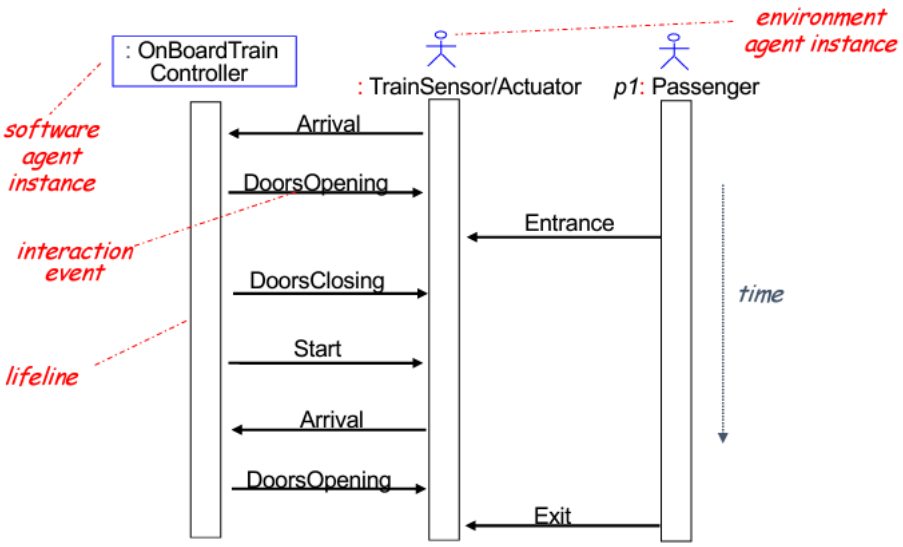


Figura 12.1.1: Esempio di diagramma di sequenza UML

Eventi

Gli eventi sono dei messaggi **istantanei** che vengono scambiati tra le varie istanze di agenti. Possono avere degli attributi che ne specificano il contenuto.

Di fatto un evento corrisponde all'applicazione di un'operazione e sono tipicamente emessi da un'istanza di agente e ricevuti da una o più istanze di agente.

Nel sequence diagram abbiamo due tipologie di ordinamenti:

- **Totale**: data una coppia di eventi possiamo dire quale evento è stato ricevuto prima dell'altro.

- **Parziale**: data una coppia di eventi non possiamo dire quale evento è stato ricevuto prima dell'altro.

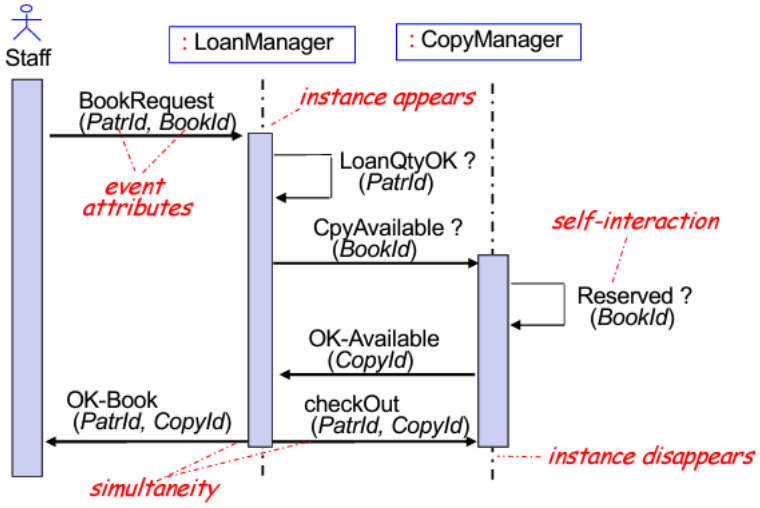
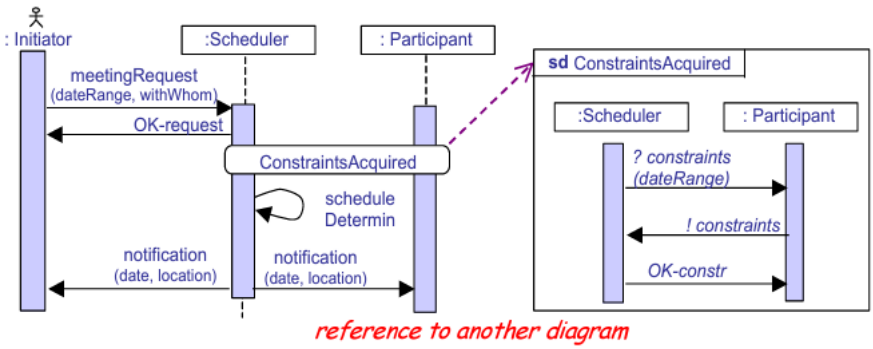


Figura 12.1.2: Esempio di diagramma di sequenza

12.1.2 Raffinamento dei diagrammi di sequenza

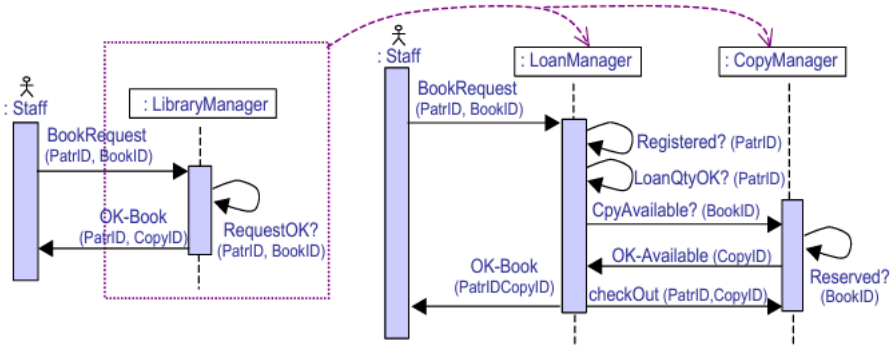
Episodi

Uno scenario che rappresentiamo potrebbe essere composto da episodi, ovvero sotto-scenari.



Decomposizione degli agenti

È possibile decomporre una responsabilità complicata su più agenti, in modo da ridurre le granularità delle responsabilità.



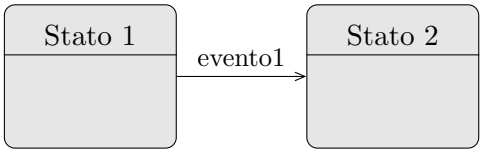
12.2 Modellare classi di comportamenti

Una **macchina a stati** cerca di fornire tutti i possibili funzionamenti di un sistema, per farlo si utilizza una macchina a stati.

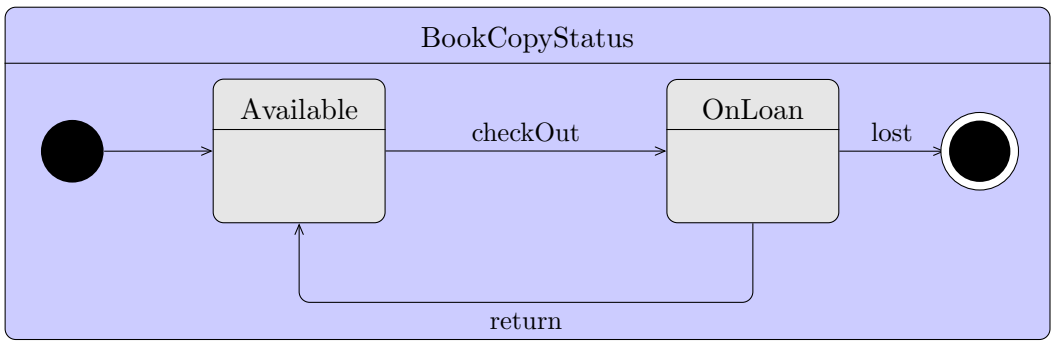
Un macchina a stati rappresenta una serie di transizioni tra stati

$$S \times E \to S$$

- Gli stati sono finiti.
- Gli eventi sono finiti.
- Le state machine sono deterministiche.



Esempio pratico



Dal punto di vista semantico, una transizione ha luogo quando il sistema è in un particolare stato e occorre l'evento rappresentato.

Nei capitoli precedenti è stato definito il concetto di **stato** (9.1).

Quando parliamo delle state machine siamo più generici rispetto alla definizione precedente, non ci riferiamo quindi a una particolare istanza di un oggetto, ma a una **classe di istanze oggetti**.

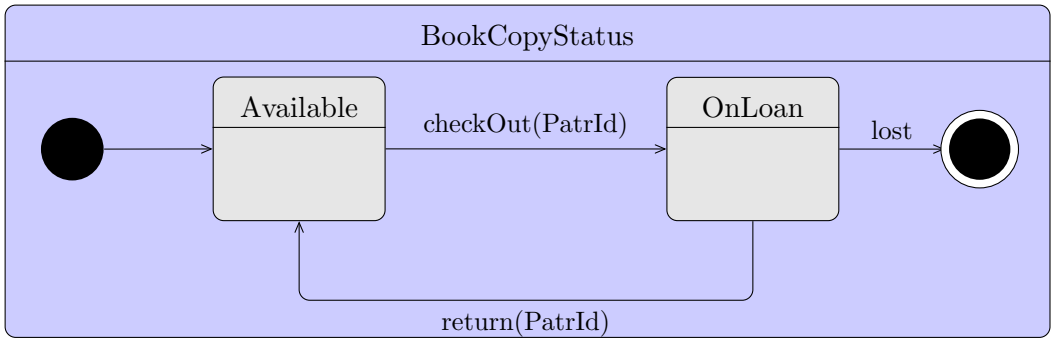
12.2.1 Gli scenari come macchine a stati UML

Tipicamente abbiamo una state machine per ogni variabile **controllata** da un agente.

Tipicamente le transizioni sono **istantanee**, mentre gli stati hanno una certa durata.

- Possiamo rappresentare lo stato iniziale con un pallino pieno, che tipicamente rappresenta l'istanziazione dello stato.
- Possiamo rappresentare lo stato finale con un pallino pieno con un cerchio attorno, che tipicamente rappresenta la terminazione dello stato.

Gli eventi possono avere degli attributi.



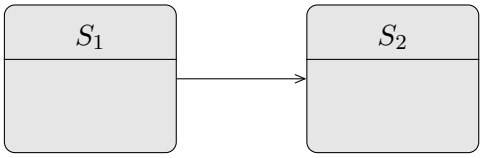
Gli eventi si contraddistinguono in diverse varianti:

- **Eventi esterni**: gli eventi esterni arrivano da un'altra state machine, che si suddividono in:
  - **Eventi temporali**: all'arrivo di un certo orario succede qualcosa.
  - **Eventi di durata**: un evento che dura per un certo periodo di tempo.
  - **Stimoli**: un evento che arriva dall'esterno, come per esempio da un agente. (*Provenienti da un'altra state machine*).
- **Eventi interni**: generati che è associato alla state machine. *DoorsClosing* è un'applicazione dell'operazione *closeDoors*.

> Usiamo il participio presente per gli eventi, mentre per le operazioni usiamo il verbo all'infinito.

Transizione automatiche

Nel caso in cui non ci fosse alcuna etichetta che contraddistingue una transizione, allora si tratta di una transizione automatica. Non sarà necessario che un evento scateni la transizione.



Guardie per gli eventi

Le guardie sono condizioni necessari per far si che un evento possa scatenare una transizione. Tipicamente si tratta di una condizione booleana.

È importante non confondere le condizioni necessarie con le condizioni sufficienti per far scattare una transizione. La condizione necessarie è che la guardia sia vera, la condizione sufficiente è che l'evento occorra.

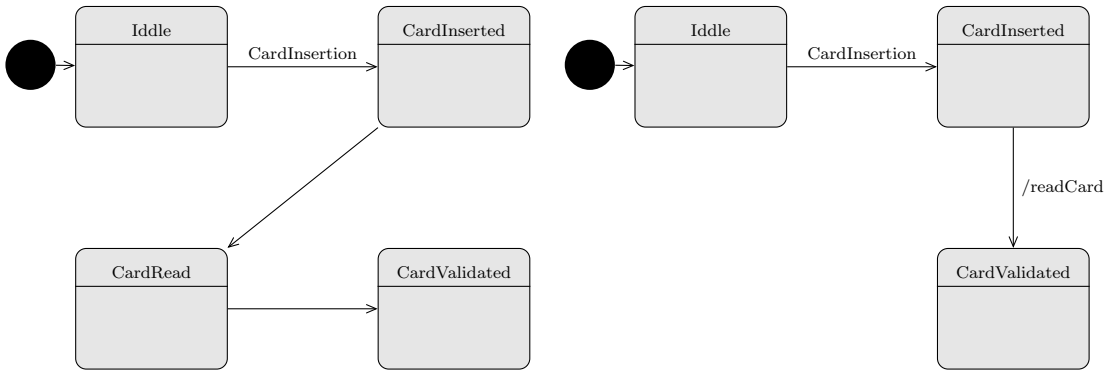
Un esempio può essere `copyBorrowing[LatCopy]`.

Azioni ausiliarie

Le azioni ausiliarie sono condizioni che devono essere intraprese con una transizione va da uno stato ad un altro. Tipicamente si tratta di operazioni che vengono eseguite un evento possa essere scatenato.

Un esempio può essere `DoorsOpening[AtPlatform and Speed = 0] / Display terminalInfo`. Tali operazioni sono atomiche e possono essere notifiche o azioni intraprese dall'ambiente quando la transizione avviene.

Evitare transizioni inutili



Invece di rappresentare uno stato intermedio utilizzo un'azione ausiliaria per rappresentare la lettura della carta.

Notifica degli eventi

Nel contesto del goal model potrebbe aver senso il meccanismo delle notifiche, in quanto potremmo avere una relazione tra due state machine distinte, avendo che alcuni eventi possono essere notificati da una state machine all'altra.

Ad esempio `send BookInfo.copyReturn`. Dove BooInfo è la state machine che riceverà l'evento, mentre copyReturn è l'evento che verrà notificato.

Azioni di entrata e uscita

Gli eventi di entrata e uscita etichettano eventi che avvengono quando si entra o si esce da un particolare stato. Permettendoci di evitare duplicati.

Invece di etichettare tutti gli stati di ingresso etichettiamo lo stato con `entry/ <azione>`, mentre per gli stati di uscita etichettiamo lo stato con `exit/ <azione>` per evitare di duplicare le etichette in uscita.

12.2.2 Raffinamento delle macchine a stati

Decomposizione sequeziale degli stati

È possibile rappresentare una state machine all'interno di un'altra state machine, esplodendo lo stato in una serie di sotto-stati. Allo stesso modo possiamo rappresentare le due state machine in due diagrammi separati. rappresentando la relazione tra le due state machine.

Decomposizione parallela

È possibile rappresentare due state machine in parallelo, in modo da rappresentare come lo stato del sistema evolva in parallelo a seconda di due proprietà distinte.

La rappresentazione d'esempio piò essere la transizione da treno fermo a treno in movimento e parallelamente da porte chiuse a porte aperte, che avvengono in parallelo.

Fork e join

Dopo aver rappresentato due state machine in parallelo, possiamo rappresentare il punto in cui le due state machine si incontrano, ovvero il punto in cui le due state machine si riuniscono. Allo stesso modo possiamo rappresentare il punto in cui le due state machine si dividono.

Per farlo utilizziamo i nodi di `fork` e `join`.

12.3 Costruire modelli di comportamento

12.3.1 Goal, scenari e state machine sono complementari

Goal model

Vantaggi del goal model

- Hanno una rappresentazione dichiarativa, aiutano quindi a capire *quando* e da *chi* viene soddisfatto un goal.
- Rappresentazione di aspetti funzionali e non funzionali.
- Rappresentano tanti funzionamenti.
- Può essere usato in analisi preliminari.

Limiti del goal model

- Molte cose rimangono implicite, non spiego *come* il goal viene soddisfatto.
- Troppo astratto.
- Difficile da elicitare.

Scenari

Vantaggi degli scenari

- Esempi concreti e narrativi.
- Facili da elicitare.
- Rappresentano comportamenti espliciti.
- Utili per i test di accettazione.

Limiti degli scenari

- La copertura completa è difficile, ogni sequence diagram rappresenta una sola esecuzione.
- Ragionare in termini potrebbe introdurre dei bias, rappresentando come il sistema funzionerà, con scelte premature (*potrebbero introdurre scelte architetturali premature*).

State machine

Vantaggi delle state machine

- Rappresentazione visuale.
- Rappresentazione esplicita di comportamenti del sistema (*classi*).
- Verificabile e eseguibile.
- Supporta la generazione di codice.

Limiti delle state machine

- Requisiti impliciti, quindi troppo operazionale.
- Difficile da costruire e da capire.