→ *Quarto Blocco*

Il processo di ingegneria dei requisiti (*requirements engineering*) varia in base al dominio applicativo, alle persone coinvolte ed all'organizzazione che sviluppa il sistema software.

Si può però individuare un insieme di attività generiche comuni a tutti i processi. Seguono in ordine di esecuzione:

- studio di fattibilità (feasibility study)
- identificazione e analisi dei requisiti (requisiti elicitation and analysis)
- specifica dei requisiti (requisiti specification)
- convalida dei requisiti (requisiti validation)
- gestione dei requisiti (requisiti management)

Vediamo nel dettaglio le fasi elencate precedentemente:

- *Studio di fattibilità:* è una fase iniziale (o anche fase che precede i requisiti) del processo di ingegneria dei requisiti. Si basa su una descrizione sommaria del sistema software e delle necessità utente. Le informazioni necessarie per lo studio di fattibilità vengono raccolte da colloqui con:

- client manager
- ingegneri del software con esperienza nello specifico
- dominio applicativo
- esperti delle tecnologie da utilizzare
- utenti finali del sistema

Lo scopo di questi colloqui è rispondere ad una serie di domande, del tipo:

- In che termini il sistema software contribuisce al raggiungimento degli obiettivi strategici del cliente?
- Può il sistema software essere sviluppato usando le tecnologie correnti e rispettando i vincoli di durata e costo complessivo?
- Può il sistema software essere integrato con altri sistemi già in uso?

Lo studio di fattibilità produce come risultato un report/documento che stabilisce l'opportunità o meno di procedere allo sviluppo del sistema software.

- *Identificazione e analisi dei requisiti:* In questa è fondamentale interagire, infatti il team di sviluppo incontra il cliente e gli utenti finali al fine di identificare l'insieme dei requisiti utente, dalla cui analisi si generano i requisiti di sistema (specifiche). L'identificazione dei requisiti può coinvolgere personale che copre vari ruoli sia all'interno dell'organizzazione del cliente che in altre organizzazioni o tra gli utenti finali. A questo proposito viene introdotto il termine *stakeholder*, il quale viene usato

per identificare tutti coloro che hanno un interesse diretto o indiretto sui requisiti del sistema software da sviluppare.

Esistono dei task principali per fare questa identificazione e analisi:

- *Comprensione del dominio:* l'analista deve acquisire conoscenze sul dominio applicativo.

  Esempio: se il sistema software deve supportare il lavoro di un ufficio postale, l'analista deve comprendere il funzionamento di un ufficio postale)

- *Raccolta dei requisiti:* mediante interazione con gli stakeholder si identificano i requisiti utente.

- *Classificazione:* l'insieme dei requisiti raccolti viene suddiviso in sottoinsiemi coerenti di requisiti

- *Risoluzione dei conflitti:* eventuali contraddizioni e/o conflitti tra requisiti vanno identificati e risolti

- *Assegnazione delle priorità:* mediante interazione con gli stakeholder, ad ogni requisito o sottoinsiemi di requisiti va assegnata una classe di priorità

- *Verifica dei requisiti:* i requisiti vengono controllati per verificarne completezza e consistenza, in accordo a quanto richiesto dagli stakeholder

Per quanto riguarda le tecniche di identificazione dei requisiti sono 3:

- Ethnography, basata sull'osservazione: quando l'utente non è in grado o non può farlo, quello che fa l'analista software è osservare e descrivere i requisiti
- Casi d'uso o basati su scenari: definire dei possibili scenari e a partire da questi l'analista software dovrebbe capire quali sono i requisiti.
- Prototipazione (già vista precedentemente)

Esistono pure tecniche di analisi (e specifica) dei requisiti, ovvero:

- Tecniche semi-formali, basate su modelli del sistema e usate dai metodi di analisi strutturata o analisi orientata agli oggetti [vedremo in dettaglio più avanti]
- Tecniche formali, basate su Petri Net, FSM, Z, etc.

- *Convalida dei requisiti*

  È finalizzata ad accertare se il documento dei requisiti, ottenuto come risultato della fase di analisi, descrive realmente il sistema software che il cliente si aspetta. La scoperta di errori in questa fase è fondamentale per evitare costosi rework in fasi più avanzate del ciclo di vita. I controlli da effettuare includono:

  - Validità
  - Consistenza
  - Completezza
  - Realizzabilità
  - Verificabilità

  Per la convalida dei requisiti esistono delle tecniche di convalida. Di seguito elencate dal più informale al più formale:

  - revisioni informali
  - revisioni formali: possono essere di vario tipo:
    - *walkthrough:* letteralmente "camminare dentro" il documento. Durante un meeting organizzato, i partecipanti ricevono in anticipo il documento da esaminare. Nel corso della riunione, una persona legge il documento ad alta voce e, in caso si riscontrino problemi, la lettura viene interrotta per discuterne insieme.
    - *ispezioni:* richiedono ruoli specifici tra cui un moderatore, uno sviluppatore, un progettista, un analista, un tester e uno "scribe" (segretario). Il processo segue passaggi formali: si inizia con un'analisi preliminare (overview), poi con la fase di preparazione durante la quale il documento viene distribuito ai partecipanti. Successivamente si svolge l'ispezione vera e propria, durante la quale si redige un report che raccoglie eventuali errori. Dopo aver apportato le correzioni, si procede con un "rewalk" per verificare la soluzione. È importante che la riunione di ispezione non superi le 3 ore, per evitare cali di concentrazione. Sebbene questa tecnica sia molto efficace, comporta costi elevati.
  - Prototipazione
  - generazione dei test-case
  - analisi di consistenza automatizzata (per requisiti formali)

Nel task di identificazione dei requisiti ad ogni requisito si associa un etichetta, in base alla probabilità che un requisito possa essere modificato sia in fase di sviluppo che durante la fase di uso e manutenzione. Infatti avremo:

- *requisiti stabili*, con probabilità minima di essere modificati nel tempo
- requisiti volatili, con probabilità elevata di essere modificati nel tempo. Possiamo elencare dei sottotipi che ti specificano quale può essere l'origine della modifica:

• mutabili: modifiche legate a cambiamenti nell'ambiente operativo

• emergenti: modifiche causate da una migliore comprensione del sistema software

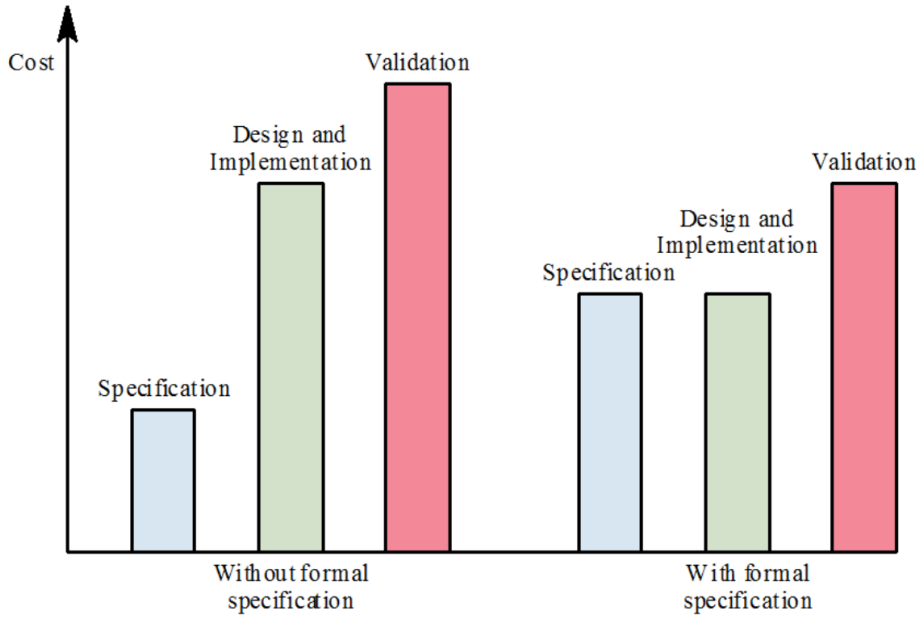
• consequenziali: modifiche legate all'introduzione di sistemi informatici nel flusso di lavoro

• di compatibilità: modifiche legate a cambiamenti nei sistemi e nei processi aziendali

Ogni modica dei requisiti deve essere opportunamente modificata, attraverso:

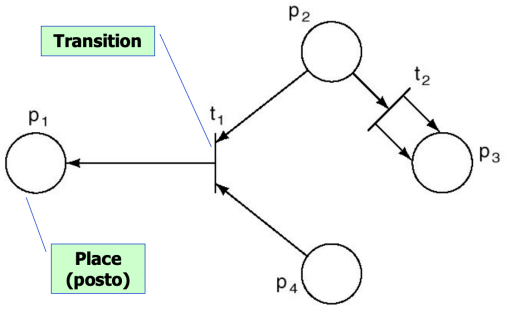
- identificazione univoca dei requisiti
- gestione delle modifiche
    - analisi dei costi
    - analisi dell'impatto
    - analisi della realizzazione
- politiche di tracciabilità (relazioni tra requisiti e tra requisiti e progetto del sistema software)
- uso di tool CASE per il supporto alle modifiche

Qui riportiamo un diagramma che ci descrive il costo di specifiche formali e informali



- ***Lezione 10 – (11/11/2024)***

Un esempio di linguaggio di modellazione formale, dotato di una sintassi visuale semplice e intuitiva, è la *rete di Petri (Petri Net).* Sebbene questo linguaggio non sia stato originariamente concepito per la specifica del software, si è rivelato estremamente efficace in tale ambito.

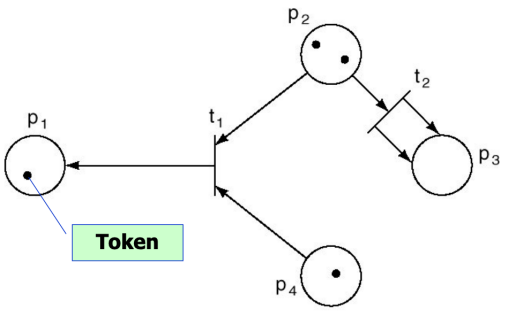


I costrutti essenziali delle reti di Petri sono:

1. *Posti (places):* rappresentano componenti o stati di un sistema.
2. *Transizioni (transitions):* modellano le azioni o esecuzioni che provocano un cambiamento di stato nel sistema.

Le reti di Petri vengono utilizzate per descrivere il *comportamento dinamico di un sistema*, mostrando come questo evolve partendo da uno stato iniziale, attraversando stati intermedi, fino a raggiungere uno stato finale. L'obiettivo è esplorare lo **spazio degli stati** del sistema, analizzandone i possibili comportamenti e valutando alcune proprietà fondamentali.

Dal punto di vista strutturale, i *posti* e le *transizioni* sono collegati tramite *archi orientati*. I posti possono contenere dei *token*, elementi che abilitano l'esecuzione delle transizioni. Questo meccanismo rende la rete di Petri un potente strumento per rappresentare il comportamento dinamico di un sistema.

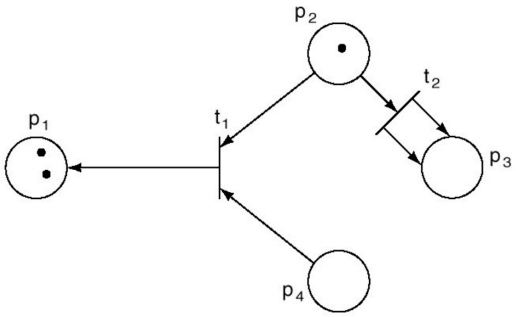


Affinché una transizione possa essere eseguita (o, in termini tecnici, *firing)*, deve essere abilitata. Una transizione è abilitata quando:

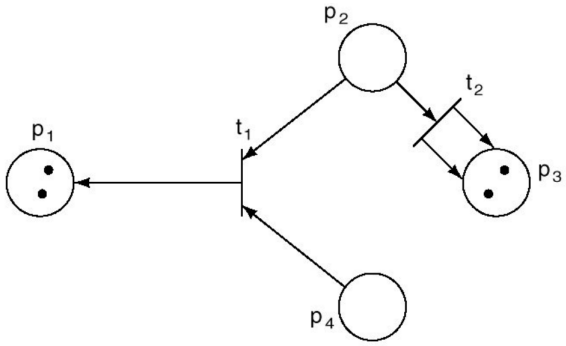
- Ogni posto collegato in ingresso alla transizione contiene almeno un *token.*

Quando questa condizione è soddisfatta, la transizione può "scattare", consumando i token dai posti in ingresso e producendo token nei posti in uscita.

Esempio1: se facciamo scattare la transizione da t_1, facciamo scattare un token da p_2 e p_4 e poi ne generiamo uno solo (arco uscente uno solo), il quale viene depositato nel posto p_1

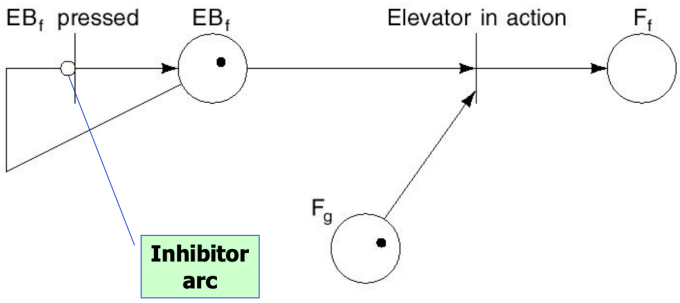


A questo punto scatta pure la transizione da t_2, e non essendoci almeno un token in ogni posto collegato in input, abbiamo uno stato finale.

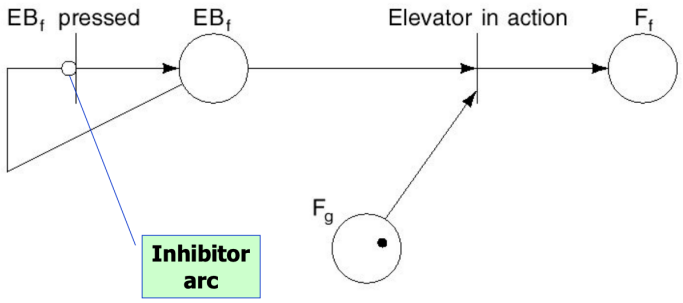


Questa rete di Petri che vediamo rappresentata in figura serve a specificare un requisito molto semplice, che ci dice come un ascensore deve comportarsi.

Il posto f_g rappresenta l'ascensore al piano terra (ground floor)
Il posto f_f rappresenta l'ascensore al primo piano (first floor)
Il posto chiamato EB_f rappresenta il pulsante dell'ascensore (elevator botton)



Dall'immagine possiamo notare che il token che sta all'intero di f_g ci dice che l'ascensore sta al piano terra. Mentre, il token che sta all'interno del posto EB_f ci dice che quel pulsante è stato premuto.
A questo punto voglio rappresentare il movimento dell'ascensore, questo lo faccio attraverso la transizione che si chiama "ascensore in movimento".
La transizione è abilitata quando esiste almeno un token che collega gli archi in ingresso. Quindi essendo abilitata può essere eseguita.



Possiamo notare che se nel posto chiamato EB_f, il quale rappresenta il pulsante dell'ascensore (elevator botton), non è presente il token. Ci serve un'altra transizione, che deve rappresentare la "pressione del pulsante", questa transizione è data da "EB_f pressed". Questa è una transizione con arco inibitore (con il pallino davanti), ovvero ha la funzione di inserire il token nel posto in input.

Nella rete di Petri esaminata abbiamo notato che ogni transizione rappresenta un'azione, l'esecuzione di queste azioni è istantanea, quindi non c'è un tempo di esecuzione della transizione. Per esempio, quando l'ascensore sta a piano terra l'ascensore si ritrova al primo piano, ovviamente questa cosa è assurda.
Questa limitazione è stata superata da dei "dialetti" delle Reti di Petri:

- introduciamo nome di *GSPN*, ovvero *Generalized Stochastic Petri Net*, ad ogni transizione viene associata un tempo con valor medio che ci permette di valutare e convalidare gli aspetti prestazionali del sistema.
- un ulteriore dialetto utilizzato è *CPN, Colored Petri Net*. Consiste in un insieme di colori che possono descrivere comportamenti diversi del sistema per diverse classi di utente.

Quindi il concetto di stato nelle Reti di Petri, si vede in modo indiretto, in base a come sono distribuiti i token nei vari posti.

- ***Macchine a stati Finiti (FSM)***

L'obiettivo di una macchina a stati finiti è uguale a quello delle Reti di Petri, ovvero è quello di rappresentare l'evoluzione di un sistema attraverso un insieme di stati che il sistema attraversa a partire da uno stato iniziale fino ad uno stato finale.

La differenza, rispetto alla Rete di Petri, è che la primitiva di base è quella di rappresentare direttamente un possibile stato del sistema.

In questo contesto, possiamo utilizzare 2 o 3 primitive, dove la primitiva principale è il rettangolo, che rappresenta lo stato. Gli archi orientati indicano la transizione tra uno stato e il successivo.

Possiamo rappresentare gli stati iniziali e finali con due tonalità di grigi diversi (come in foto), e li possiamo riconoscere pure perché gli stati iniziali non avranno nessuno stato in ingresso e gli stati finali non hanno nessun arco in uscita.

Il seguente esempio rappresenta il comportamento di una cassaforte.

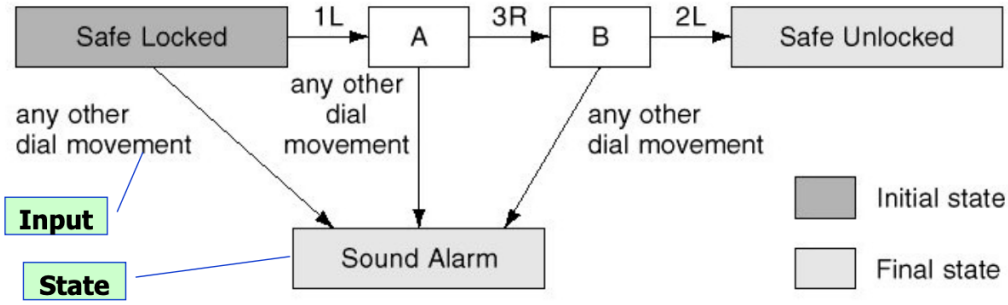


Table of Next States

Dial movement \ Current state	Safe locked	A	B
1L	A	Sound Alarm	Sound Alarm
1R	Sound Alarm	Sound Alarm	Sound Alarm
2L	Sound Alarm	Sound Alarm	Safe Unlocked
2R	Sound Alarm	Sound Alarm	Sound Alarm
3L	Sound Alarm	Sound Alarm	Sound Alarm
3R	Sound Alarm	B	Sound Alarm

N.B: Possiamo vedere nell'immagine la rappresentazione visuale, una tabella di stati successivi

Da questo esempio di macchina a stati finiti possiamo vedere:
Lo stato iniziale è Safe Locked (Stato Chiuso) e lo stato finale può essere Safe Unlocked (Stato Aperta) o Sound Alarm (Suono di Allarme)

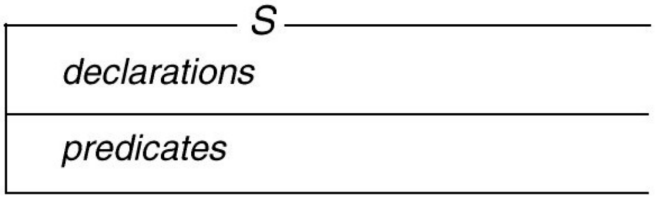
Tra lo Stato di Chiusura della cassaforte e lo Stato di Apertura della cassaforte ha una serie di stati intermedi che rappresentano le combinazioni. Ad esempio:

- lo stato 1L indica un giro verso sinistra
- lo stato 3R indica tre giri verso destra
- lo stato 2L indica un giro verso sinistra

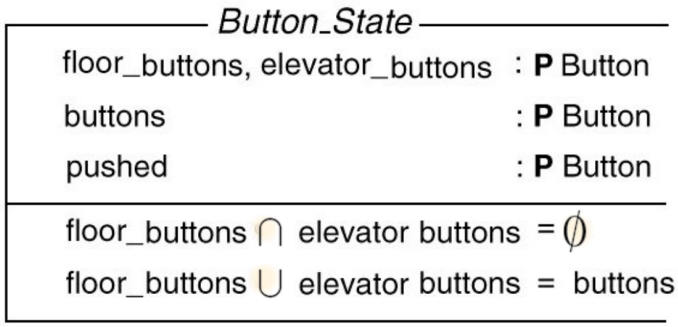
Quando il movimento è spagliato arriveremo allo stato finale Sound Alarm (Suono di Allarme).

Un altro esempio di notazione formale proposta appositamente per la specifica di prodotti software. Questa si chiama linguaggio Z.
Questo linguaggio mette a disposizione un'unica primitiva, è il concetto di schema.
Ogni schema ha:

- un nome
- un primo compartimento che contiene dichiarazioni di variabili
- un secondo dipartimento che contiene l'insieme di predicati che agiscono su quelle variabili



Un *esempio* di specifica di stato è il seguente:



Abstract Initial State

```
Button_init := [Button_State' | pushed' = ∅]
```

Questo è un esempio sempre relativo all'ascensore, nello specifico al pulsante dell'ascensore.

Qui vediamo che questo schema è formato:

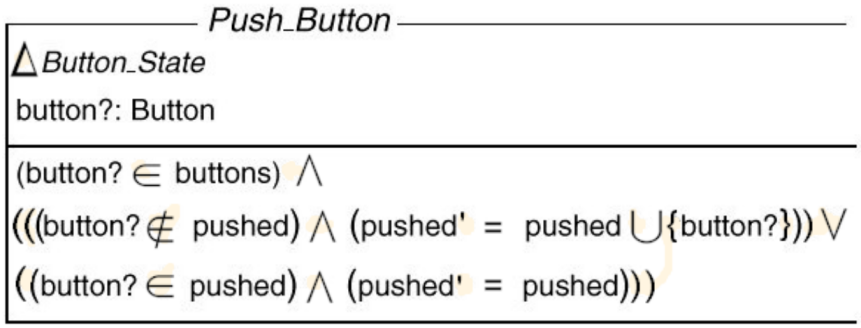
- dal nome → Button_State
- dal primo dipartimento di dichiarazioni di variabili → floor_buttons, elevator_buttons e così via. Per definire queste variabili le definiamo nel seguente modo:

  *nome variabile : **P** Button*

  dove *Button* è l'insieme di tutti i pulsanti e *P* indica l'insieme potenza

- un secondo dipartimento di predicati che agiscono su quelle variabili → vado a vedere come si comportano questi predicati definiti precedentemente. Faccio ciò usando i predicati e attraverso simboli di insiemistica indico le varie "azioni"

Un *esempio* di specifica di un operazione è il seguente:



Lo schema è formato nello stesso "formato" di uno schema per la specifica di uno stato. Possiamo notare:

- il nome che è dato da un'azione
- il Δ che per indicare su quali stati agisce e il *punto interrogativo ?:* specifica un parametro di input

Oltre alle specifiche formali, esistono delle *specifiche semi-formali*.

Il termine "modello del sistema" si riferisce a una rappresentazione formale o astratta del sistema, che aiuta a comprendere le sue proprietà e il funzionamento prima che venga effettivamente realizzato.

L'uso dei modelli nei sistemi software è integrato nei metodi di analisi dei requisiti, ovvero nella fase di specifica del software. Questi metodi fanno spesso uso di tecniche semi-formali per descrivere il sistema.

I metodi di analisi dei requisiti software si dividono principalmente in due categorie:

1. metodi di analisi strutturata, o procedurale;
2. *metodi di analisi orientata agli oggetti.*

Per avere una descrizione completa del sistema, è necessario creare diversi modelli che lo rappresentino da differenti prospettive. In particolare, si devono considerare le informazioni gestite dal sistema, le sue funzionalità e il suo comportamento dinamico.

Per descrivere la specifica *semi-formale* di un sistema software si usano 3 tipi di modelli:

- modello dei dati → rappresenta gli aspetti statici e strutturali relativi ai dati (data requirements)
    - ERD (*not UML*)

- Classdiagram (*UML*)

- modello comportamentale → rappresenta gli aspetti funzionali del sistema (functional requirements)

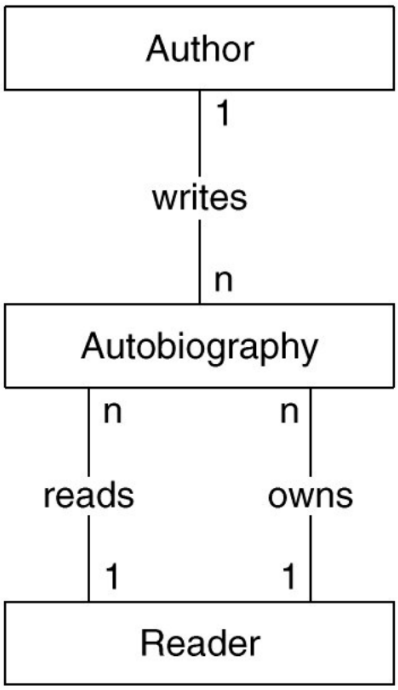
  - data flow diagram (*not UML*)
  - use case diagram (*UML*)
  - activity diagram (*UML*)
  - interaction diagram (*UML*)

- modello dinamico → rappresenta gli aspetti di "controllo“ e di come le funzioni del modello comportamentale modificano i dati introdotti nel modello dei dati

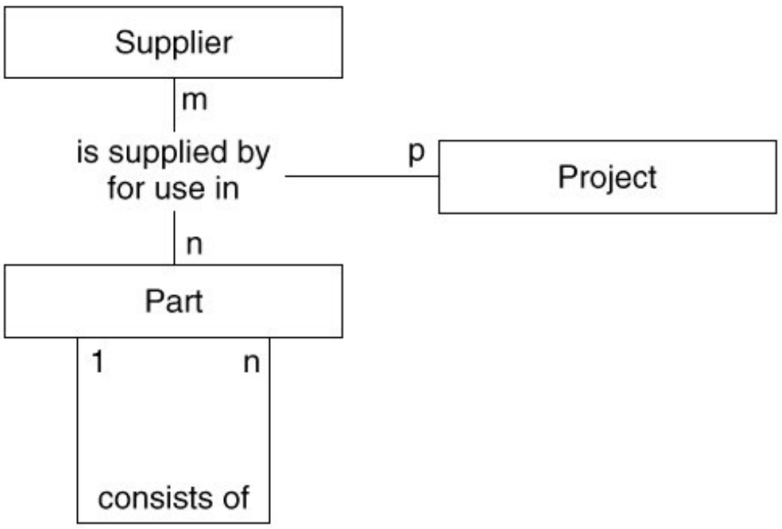
  - state diagram (*UML*)

Per quanto riguarda il modello ERD, ovvero Diagramma Entità Relazioni. Si usa per realizzare lo schema relazionale/logico di un database.

Nella figura affianco possiamo vedere un Diagramma Entità Relazioni, 1:N (uno a molti). Nel seguente esempio vediamo 3 informazioni che il sistema deve gestire che sono Autore, Autobiografia e Lettore. Le relazioni stabiliscono il numero di possibili istanze di un’entità che sono messe in corrispondenza con un’istanza dell’altra entità.

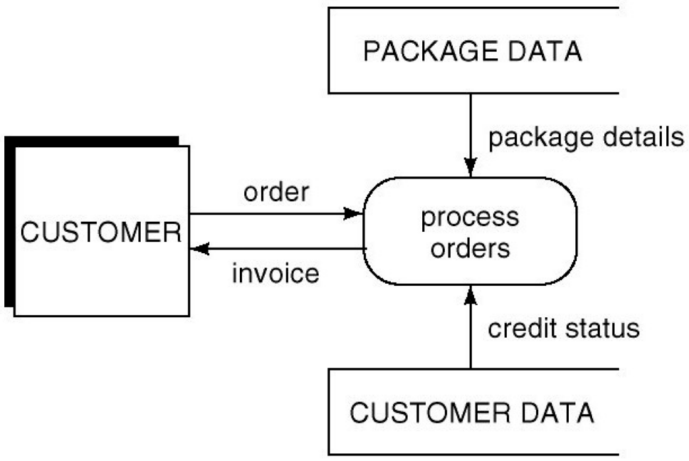


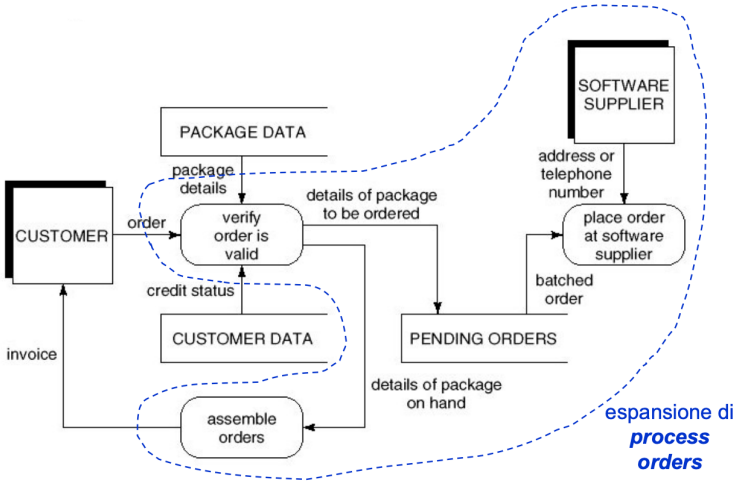
Nella figura affianco possiamo vedere un Diagramma Entità Relazioni, N:N (molti a molti).



Per quanto riguarda il modello DFS, ovvero Data Flow Diagram, serve a costruire modelli comportamentali. I costrutti di cui disponiamo in questo modello sono i seguenti quattro:

Segue un esempio di Software che elabora ordini. Possiamo notare un primo raffinamento (figura a destra) e un secondo raffinamento (figura in basso).





- ***<u>Lezione 11</u>***

Il metodo di analisi strutturata (Structured System Analysis) abbreviato (SSA). È stato introdotto da Gane e Sarson, nel 1979. È Basato sul concetto di step-wisere finement, infatti è costituito da 9 step.

- Primo Step → Disegna il DFD (data flow diagram). Usa il documento dei requisiti (o il prototipo) per:
  - Identificare i flussi di dati
  - Identificare la fonte e le destinazioni dei dati (dove i flussi di dati iniziano e finiscono, rispettivamente)
  - Identificare i processi che trasformano i dati

  Perfezionare il DFD aggiungendo nuovi flussi di dati o aggiungendo dettagli ai flussi di dati esistenti

- Secondo Step → Decidere quali sezioni computerizzare e quali no. Utilizzare l'analisi costi-benefici per decidere quali sezioni del DFD automatizzare. Decidere come informatizzare:
  - Operazioni batch
  - Elaborazione online

  I prossimi 3 passaggi sono il perfezionamento graduale di flussi di dati, processi e archivi di dati.

- Terzo Step → Determina i dettagli dei flussi di dati. Decidi quali elementi di dati devono essere inseriti nei vari flussi di dati.

- Quarto Step → Definisco la logica dei processi.

- Quinto Step → Determinare gli archivi di dati. Definire il contenuto esatto di ogni negozio e la sua rappresentazione (formato specifico in un determinato linguaggio di programmazione). Definire il livello di accesso utilizzando il diagramma di accesso immediato ai dati (DIAD)

- Sesto Step → Definisci le risorse fisiche

- Settimo Step → specifica le forme di input (al livello di componenti e layout), gli schermi di output devono essere determinati allo stesso modo. Anche l'output da stampare deve essere specificato (lunghezza e dettagli stimati)

- Ottavo Step → Determinazione delle dimensioni. Qui calcoleremo:
  - volume di input (giornaliero o orario)
  - frequenza di ogni report stampato e relativa scadenza
  - dimensione e numero di record che devono passare tra la CPU e l'archiviazione di massa
  - dimensione di ogni file

- Nono Step → Specificare i requisiti Harware. Determinare:
    - Requisiti di archiviazione di massa
    - Requisiti di archiviazione di massa per il backup
    - Caratteristiche dei terminali utente
    - Caratteristiche dei dispositivi di output
    - Adeguatezza dell'hardware esistente
    - Costi dell'hardware da acquistare

Il Nono Step è l'ultimo passaggio di SSA. Dopo l'approvazione da parte del cliente, il documento di specifica risultante viene consegnato al team di progettazione e il processo software continua

Possiamo trovare i seguenti svantaggi:
– SSA non può essere utilizzato per determinare i tempi di risposta
– Le dimensioni e la tempistica della CPU non possono essere determinate con alcun grado di accuratezza