

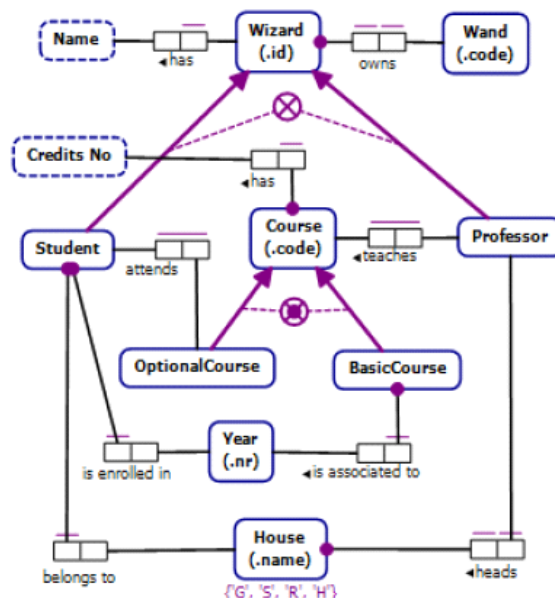
Introduzione

martedì 21 settembre 2021 11:22

Parliamo di modellazione di dati e processi che sono argomenti molto collegati fra di loro ed estremamente importanti. Si vuole fornire un supporto a quello che è il passo principale per costruire sistemi informativi. La modellazione concettuale ha lo scopo di modellare ai fini di una certa applicazione/obiettivo, serve quindi a descrivere una realtà per quella che è, è diciamo la costruzione di un "dizionario". È una descrizione formale perchè non può essere ambigua la formulazione e infatti non si può usare il linguaggio naturale. Conoscendo le regole il linguaggio utilizzato si può usare per la comunicazione.

I processi modellano le attività di attori che hanno un impatto sui dati, facendoli evolvere. Questi due aspetti vanno considerati insieme perchè sono due facce dello stesso mondo. Un altro aspetto sono le risorse, che possono essere fisiche o umane che influenzano come vengono eseguiti i processi e come vengono modificati i dati.

Da dati e processi si hanno i sistemi informativi che permettono di mantenere info e aggiornarle in base alle azioni lecite. La modellazione dei dati verrà fatto con l' **Object Role Modeling (ORM)**. Non si rifà a database o strutture di oggetti ma è indipendente. Si creano usando Norma.



Il punto di forza è la minimalità dell'informazione. ORM non usa attributi, usa solo entità e relazioni e andrà a rappresentare questi dati solo quando è in relazione con altri dati, perchè altrimenti significherebbe che quel dato non è utile. l'obiettivo è cercare e lavorare sui fatti primitivi, quindi quelli che se li scompongo non hanno più senso. Il resto sono vincoli che definiscono come i dati sono relazionati tra loro.

Il business process model: la cosa principale sono le attività e il flusso che collega le attività. Gli eventi sono fatti che corrono in un determinato momento. Questi eventi sono collegati da un flusso. l'esecuzione delle attività potrebbe causare la modifica dei dati.

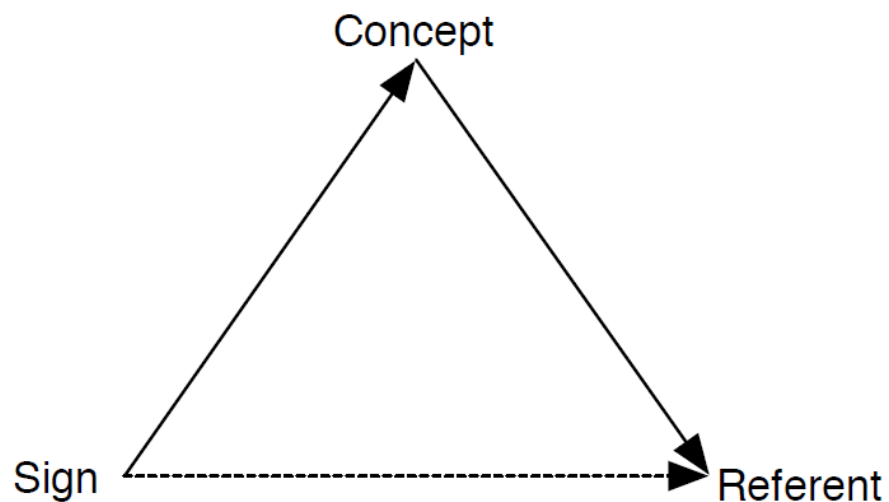
Il modello è la rappresentazione del mondo costruita per un agente.

È una rappresentazione formale, perchè il linguaggio naturale è molto ambiguo, mentre quello formale non lo è e quindi può essere facilmente compreso conoscendo le regole e si può usare per comunicare.

INTRODUZIONE

La mod di dati e processi sono estremamente inerenti all'intelligenza artificiale: un processo è sostanzialmente un piano, la modellazione serve per estrarre quello che vogliamo rappresentare come goal da raggiungere. Quando modelliamo un processo, modelliamo un goal da raggiungere (ossia l'obiettivo di un agente) e si devono deliberare delle azioni per raggiungerlo. Un goal non è solo lo stato finale ma anche come voglio raggiungerlo, il che significa che anche il come è descrizione dell'obiettivo. Si pongono dei vincoli anche, quindi rappresentano dell'incosistenza e delle regole che devono essere seguite.

Traingolo del significato: la relazione tra segno, concetto e referente. Il segno passa attraverso il concetto per individuare una realtà.

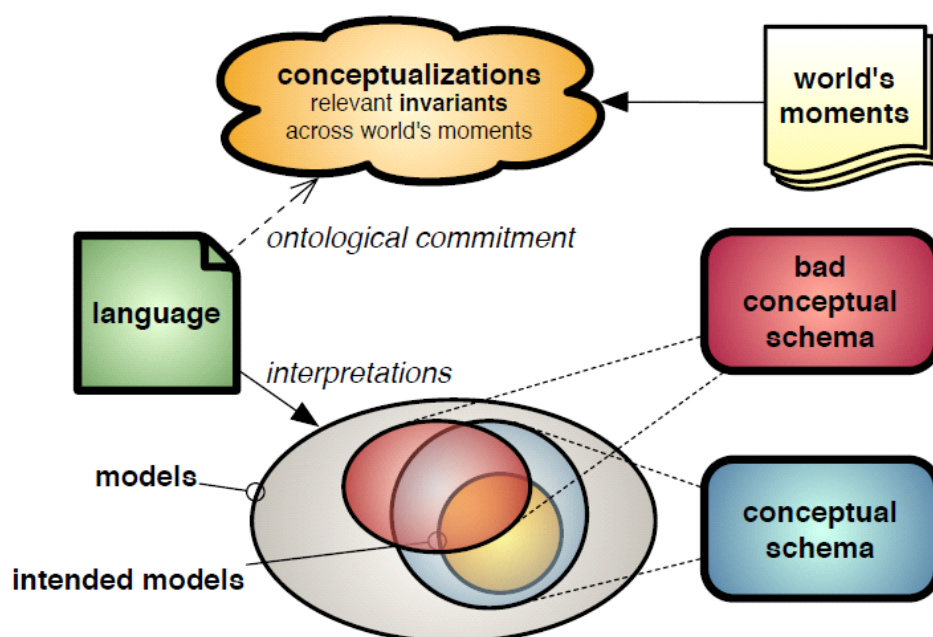


Il concetto Gatto è un concetto che noi abbiamo nella testa e ci formiamo come astrazione della realtà che emerge da un processo di analisi della realtà, e tramite conoscenza ampliata si forma un concetto che va a definire quelle che sono le caratteristiche associabili al Gatto. Comunichiamo attraverso un linguaggio che rappresenta dei concetti che fanno parte della realtà.

Devo catturare nel modo più preciso il concetto in modo tale che il linguaggio sia in grado di definirlo completamente e senza erroneamente comprendere elementi che non identifichino solo quel concetto specifico. (se rappresenti i gatti con la caratteristica 4 zampe non comprende solo il gatto ma molti animali). Bisogna trovare la migliore approssimazione del concetto che si vuole rappresentare.

Il concetto è una rappresentazione intensionale della realtà, cioè esprimo astrando dalle persone singole la caratteristica rilevante per il contesto. Una concettualizzazione emerge come processo di astrazione e generalizzazione da esperienza e percezioni su quello che è il dominio di cui stiamo parlando. Solitamente la concettualizzazione è anche a posteriori, perché si genera come risultato di un confronto, riflessione e astrazione. Si può fare solo dopo aver conosciuto il dominio.

La concettualizzazione è una parte della realtà rilevante come percepita e organizzata da un agente astruendo da una specifica situazione.



Abbiamo delle info che provengono dal mondo e ci permettono di creare una concettualizzazione, ossia una rappresentazione delle varianti rilevanti delle info che ci arrivano. Il linguaggio che usiamo deve andare a rappresentare

quei concetti. Questo linguaggio ha delle interpretazioni: se dico una parola con un segno che viene interpretato associandolo ad un concetto. l'interpretazione e il significato devono coincidere. Il modello considerato deve essere incluso completamente nel modello costruito. Se invece la descrizione prende eccessivamente altre informazioni o non comprende tutte quelle elementari di cui necessito è invece considerato uno schema sbagliato. Il problema del linguaggio è questo quindi, avere una corretta concettualizzazione delle informazioni.

Un sistema informativo, obiettivo della modellazione, colleziona, immagazzina, processa e distribuisce informazioni sullo stato del dominio per facilitare pianificazione, controllo, coordinazione e processo decisionale in una organizzazione. Il dominio è considerato universo del discorso.

Le funzioni di un Informative System (IS) sono:

- Memoria, mantenere una rappresentazione dello stato del dominio. A livello intensionale si deve memorizzare la semantica e i dati oltre ai dati stessi, come ad esempio i vincoli. A livello estensionale si hanno le istanze dei dati descritti dal livello intensionale. Per memorizzare le info può essere inserito un dato su richiesta oppure semplicemente ad ogni aggiornamento vengono memorizzati i dati.
- Informazioni, fornire info sullo stato che memorizza. Ci permette di estrarre info sul sistema su richiesta, quindi interrogazione, oppure in modo autonomo mostrando le info senza richieste.
- Attivo, eseguire azioni che cambiano lo stato del mondo. Permette al IS di agire e anche in questo caso può avvenire in maniera automatica in base a dei trigger o sotto richiesta dell'utente.

Per porre le domande, query, deve essere definito un linguaggio anche per questo che deve essere compreso da utente e IS. Le query possono essere di due tipi: intensionali ed estensionali. Si può chiedere che vengano dati dei dati con caratteristiche specifiche, ed il sistema restituisce le istanze di dati. Altresì potrebbero essere fatte delle query intensionali dove il sistema deve rispondere con delle proprietà. Il primo caso è dei database, il secondo delle basi di conoscenza.

Un sistema informativo richiede conoscenza del dominio e delle funzioni che deve poter eseguire. La rappresentazione di un dominio ha bisogno dei dati che lo costituiscono e di uno schema concettuale. Bisogna ricordare che il dominio evolve nel tempo quindi lo schema concettuale deve contenere aspetti sia statici che dinamici. Non tutte le info sono importanti perchè alcune possono essere inferite tramite regole.

Un linguaggio generale di modellazione tipicamente permette di rappresentare le entità, che rappresentano istanze di persone oggetti ecc e le relazioni tra di esse.

Oltre alle relazioni servono i vincoli, ossia delle proprietà che devono essere rispettate nel dominio. Ci sono due tipi di proprietà: FATTI PRIMITIVI, non scomponibili ulteriormente, oppure DERIVATI.

I vincoli invece possono essere STATICI relativi ai dati e alla struttura, TEMPORALI, legati all'evoluzione dei dati e DINAMICI su come le attività possono essere eseguite nel tempo.

Parlando di dati un problema è comprendere se si tratta di mondo chiuso, e che quindi tutto quello che è vero nel mondo è descritto dallo schema e tutto il resto è falso, oppure mondo aperto e che quindi le informazioni che non abbiamo sono sconosciute e non implicitamente false.

Conceptual Model = Conceptual Schema + Information Base

- **Conceptual schema:** blueprint of the domain inside the IS.
 - Orders, employees, deliveries, cancelation, customer, gold customer, gift, payment, payment transaction . . .
- **Information base (or conceptual database):** blueprint of a specific state of the domain inside the IS.
 - Order o-123-bzFGH, employee Mario Rossi, delivery of o-123-bzFGH via airmail, . . .

La struttura mette in relazione le entità e descrive le proprietà delle relazioni. Anche uno schema di comportamento specifica i cambiamenti sui dati ed esprime dei vincoli che poi definiscono quelle che sono le operazioni lecite durante l'evoluzione.

Leggi slide saltate di 1.Intro

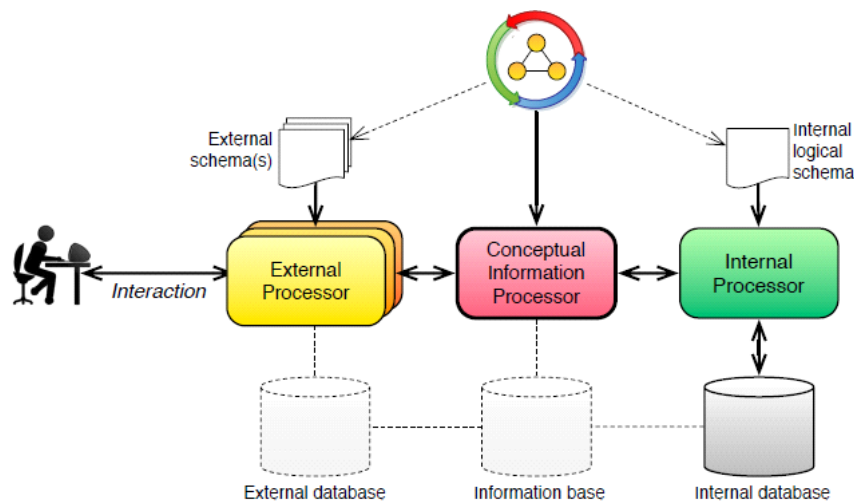
Conceptual schema = fact types + constraints + derivation rules

Conceptual schema = fact types + constraints + derivation rules

I vincoli possono essere di integrità, quindi per la consistenza dei dati. Si ha che TOTAL INTEGRITY = VALIDITY + COMPLETENESS. Il che vuol dire che le informazioni devono essere corrette sempre ma anche completa, quindi ci devono essere tutti i dati rilevanti.

Un passo importante è la validazione del sistema rispetto al modello che intendevamo, quindi verificare che il sistema sia conforme a quello che si voleva rappresentare. Se ci sono interpretazioni diverse, si hanno problemi di comunicazione. Se i modelli non sono precisi ci possono essere casi di false agreement dove sembra esserci intesa dei concetti, ma in realtà non coincidono con il modello inteso perché quello concettuale è poco preciso.

L'architettura astratta di un IS prevede un layer interno e uno esterno, che interagisce con l'utilizzatore, quindi query ed eventi o agire per causare azioni. Quello interno invece si occupa di memorizzazione e gestione delle informazioni. Vi è poi un layer intermedio tra essi che funge da interfaccia tra i due ed è detto Conceptual Information Processor, cioè la parte interna è vista come un vero storage mentre quello esterno è l'organizzazione logica dell'informazione e queste due parti sono allineate da questo livello. Ha proprio la funzione di far comunicare le due parti.



Abbiamo quindi uno schema esterno, un database esterno e un modo di elaborare le informazioni esterno che poi si vanno a collegare a quelle che invece sono le strutture interne.

Quello con cui l'utente interagisce è una visione ad alto livello dell'information system vero e proprio, che risulta in realtà essere molto complesso e difficile da gestire esternamente. Abbiamo questa visione esterna vicina all'utilizzo dell'utente.

Il Conceptual Information Processor, tramite metalinguaggi e schemi metaconcettuali permette di collimare le due parti e quindi rendere la modellazione esterna un effettivo schema nell'information system e gestire update e query.

Per la parte di data modeling,
libro di halping e morgan, lui
segue abbastanza questo libro
(sono in biblioteca disponibili
credo)
Per la parte di process
modeling, libro di Silver, libro
di weske è un po complesso

Norma : strumento per il data modelling. Si usa sotto windows (yay)

Ignavio è online e serve per modellare ed analizzare processi

Information System Development

mercoledì 3 novembre 2021 16:21

Un sistema informativo deve essere ingegnerizzato.

Development of an IS: **problem-solving**.

1. **Analysis** (includes conceptual modeling): what the IS is about, what are the requirements.
2. **Design** (includes logical modeling): how to accomplish the requirements.
3. **Implementation**: coding of the design under specific architectural/technological choices.
4. **Testing**: check if the implementation works and meets the requirements.
5. **Maintenance**: assist users after release, keep the IS working.

Ci sono vari metodi di sviluppo come il metodo waterfall, il metodo iterativo o ad esempio l'Extreme Programming.

Un aspetto importante è l'analisi dell'universo del discorso: può essere molto ampio e in esso si possono individuare delle sottocategorie anche indipendenti tra di loro. La cosa più facile è suddividere il dominio dell'IS e trattarle singolarmente con un processo di sviluppo autonomo.

I passi per lo sviluppo sono:

1. Verifica della fattibilità
2. Analisi dei requisiti design concettuale dello schema
3. Progettazione logica
4. Progettazione fisica
5. Prototipo
6. Completare l'implementazione
7. Testare e valutare
8. Rilascio

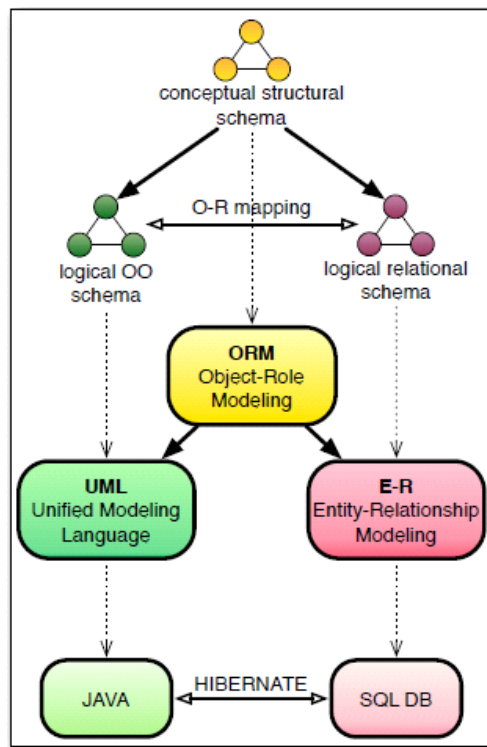
È importante interagire con l'utente o chi ci commissiona l'IS perchè potrebbero esserci diversi punti di vista, quindi è meglio esplicitare.

È importante la relazione tra gli oggetti e i dati effettivamente memorizzati nel sistema informativo. Gli oggetti mantengono delle informazioni che sono transienti, cioè memorizzati negli oggetti che però esistono durante l'esecuzione di un particolare caso d'uso. Molti però di quei dati devono diventare persistenti, ecco perchè entra in gioco la memorizzazione dei dati. La parte transiente è realizzata con l'object oriented, invece quella della persistenza è realizzata con schemi relazionali. Serve poi far dialogare queste due parti.

Quando si modella uno schema concettuale quello che ci troviamo a dover affrontare è gestire uno schema logico OO e lo schema relazionale. Il nostro modello dei dati in mente dovrà essere usato in modo transiente dalle applicazioni, ma poi dovrà essere anche memorizzato quindi sarà necessario un mapping verso una rappresentazione relazionale. Lo schema dati dovrà avere una rappresentazione Object Oriented e una logica relazionale per poter essere memorizzata.

Il punto è scegliere un linguaggio per i modelli concettuali che siano il più semplice possibile per cogliere al meglio i concetti della realtà per avere modelli precisi, corretti e completi; ma deve anche essere semplice per andare a realizzare i due tipi di schemi (OO e relazionale). Un linguaggio di modellazione concettuale deve essere preciso, cioè chiaro e non ambiguo, ma deve essere anche semplice utilizzarlo per comunicare. Inoltre deve essere conciso e non avere dettagli non necessari. Il principio di base nella modellazione concettuale è quello di modellare solo gli aspetti rilevanti, tramite un linguaggio che evidenzia e rappresenta solo le parti fondamentali da rappresentare.

Un altro aspetto è il numero di concetti utilizzati: molti concetti si hanno rappresentazioni più compatte ma più complesse.



ORM permette il mapping con gli altri due. Permette di avere delle viste UML o E-R dello stesso schema. Serve a mantenere una vista unica e facilmente comprensibile che sia mappabile negli altri due.

E-R

Lo schema e-r aveva come obiettivo quello di facilitare il passaggio da modello concettuale a database relazionale. Quindi si è già abbastanza vicini allo schema relazionale. Ha come caratteristica le entità, le relazioni e gli attributi delle entità. È indipendente dalla piattaforma sw utilizzata ma manca di dinamicità nella modellazione. La dinamicità nella modellazione significa che le caratteristiche non sono definitive ma nelle interazioni successive saranno per forza di cose modificate durante lo sviluppo. L'E-R non è così dinamico.

UML

Sviluppato come linguaggio unificato di proposte diverse di vari autori mirati all'ingegnerizzazione del sw, modellazione e progettazione. È pensato per l' Object Oriented. Di base si hanno diversi diagrammi (classi, interazioni, stati) per rappresentare il comportamento degli oggetti, l'interazione di essi con altri oggetti e la struttura complessiva delle classi. Utilizza gli attributi, o meglio le proprietà, che sono metodi, funzioni e i valori che possiamo avere in una classe. Le relazioni che ci sono tra le varie classi vengono messe in evidenza dalle molteplicità e i ruoli annotati dalle varie associazioni.

ORM

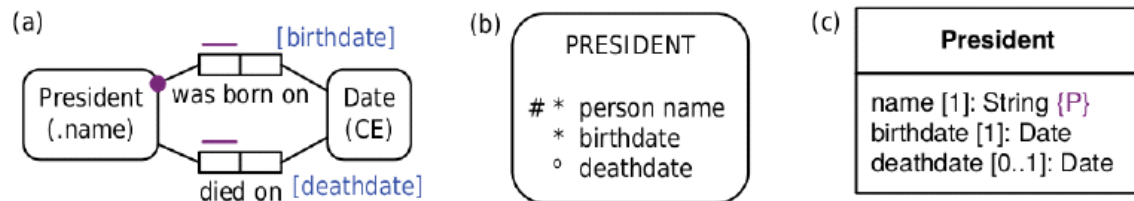
Vuole essere più semplice rispetto agli altri due, essendo meno ricco di linguaggio ma i costrutti sono più complessi. Si hanno le entità, le classi che indicano il tipo di oggetto e delle relazioni tra questi elementi. E negli altri linguaggi le entità sono le effettive istanze degli elementi, in orm le istanze si manifestano solo quando giocano un ruolo in relazione con altre entità. I ruoli sono rappresentati da quadratini e sono contenitori per un elemento per la relativa class ed appartenenza messo lì quando è in relazione con altri elementi. Ciascun elemento può essere coinvolto in più relazioni.[vedi 1) da quad]

Gli elementi rilevanti sono gli oggetti, cioè gli elementi primitivi, e i ruoli che questi giocano con altri elementi. Non ci sono attributi. Ha però molto vincoli legati alle relazioni in senso matematico e le relazioni sono predicati, cioè fatti veri affermati. In questo modo è facilitata la dinamicità. I diagrammi sono più complessi e ricchi di elementi perché gli elementi sono atomici, ma si guadagna di dinamicità. Inoltre ha sia una forma a diagramma, ma anche una forma testuale con un Linguaggio Naturale Controllato, ossia un linguaggio più semplice il cui schema è più schematizzato per renderlo più facile da interpretare e comprendere.

Il punto fondamentale è appunto che secondo i suoi ideatori , gli attributi in una progettazione (quindi non modellazione finale) concettuale sono prematuri perchè è un processo di sviluppo che è molto variabile. Dicono infatti che non sono adatti in questa fase perchè sono degli impegni appunto prematuri, prima che possa averla analizzata completamente e potrebbe cambiare successivamente.

Le relazioni sono dei FATTI veri, affermazioni. Si ha quindi una stabilità semantica, una verbalizzazione naturale, ho dei fatti d'esempio per capire se quello che modello è consistente, evita i null quindi tutte le relazioni devono essere elementari e non collegare elementi con proprietà diverse ma ci sono tutte relazioni separate le une dalle altre.

Partendo da un modello E-R per arrivare ad uno relazionale si incontrano poi alcuni problemi che devono essere gestiti, come i valori multipli negli attributi. Si presentano problemi anche a livello di vincoli esprimibili ad esempio in UML. [vedi 2) quad]



In UML e E-R la data di morte e la data di nascita sono due attributi diversi, che in queste rappresentazioni non esplicitano a primo sguardo che sono dello stesso tipo, ossia data. In UML il tipo esiste, ma in in E-R no e nei casi in cui il tipo ci è sconosciuto l'interpretazione è più complessa. In ORM invece abbiamo una sola entità che si chiama Date che è in relazione con il President in due modi diversi. Inoltre in ORM è stato eslicitato che la data di nascita è obbligatoria, in E-R anche, ma in UML non si può , dimostrando ancora come siano incomplete e meno precise le due rappresentazioni rispetto ad ORM.

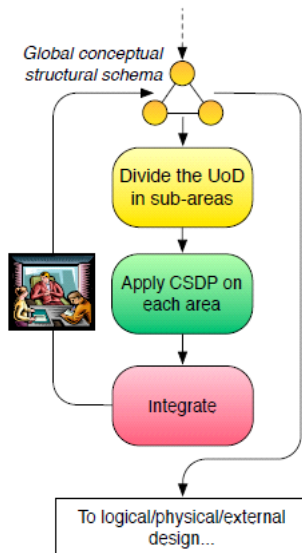
ORM

giovedì 4 novembre 2021 19:46

CSDP: Conceptual Schema Design Procedure

È una metodologia per costruire modelli ORM.

È composta da 7 passi:



1. Transform familiar examples into elementary facts.
2. Draw the fact types, and apply a population check.
3. Check for entity types to be combined, and note any arithmetic derivations.
4. Add uniqueness constraints, and check the arity of fact types.
5. Add mandatory role constraints, and check for logical derivations.
6. Add value, set-comparison, and subtyping constraints.
7. Add further constraints, do final checks.

Sostanzialmente il linguaggio è composto da individuazione delle entità, i ruoli che esse giocano con altre entità e i vincoli delle relazioni.

Passo 1:

Obiettivo di trasformare esempi familiari in fatti elementari. I fatti sono affermazioni. Il punto più difficile è comprendere l'universo del discorso perché se non abbiamo molta conoscenza del dominio diventa complesso. Dobbiamo isolare le informazioni rilevanti da rappresentare ai fini dell'obiettivo di chi vuole il sistema informativo. È rilevante solo quello che serve a raggiungere lo scopo. In questo caso i casi d'uso sono importanti perché mi permettono di capire quali sono le informazioni che mi servono, si estraggono i requisiti. Potrebbe essere necessario dover modificare il modello di dominio che è facilitato con ORM.

Il fatto elementare è una asserzione che coinvolge uno o più oggetti che sono in relazione tra loro e ciascuno di essi gioca un ruolo nella relazione.

Esempio: *If Ann employs Bob, then Bob gets a salary*. Non è elementare perché sono coinvolti due fatti elementari (Ann assume Bob e Bob prende salario). Se è scomponibile in più fatti non è elementare.

Gli oggetti di base possono essere dei valori statici e immutabili. Questi sono associati a delle entità/oggetti che ne definiscono una referenziazione ad essi. Quindi la stringa 'Lee' nella frase non si riferisce alla stringa vera, ma all'oggetto rappresentato da quella stringa. Quindi i valori sono dei 'puntatori' agli oggetti.

Le entità sono referenziate da un valore, da un esplicito entity type e dal reference mode, ossia cosa è il valore e in che modo individua l'oggetto in quel contesto.

Definite description

1. **value** ('Lee')
2. + **explicit entity type** (the Person 'Lee')...
3. + **reference mode**: the manner in which the value refers to the entity type (the Person with surname 'Lee').

Il modo di individuare gli oggetti dipende dal contesto.

(vedi quad 3))

I ruoli sono predicati con variabili. Sono detti Object holes perchè ad esempio nella frase "____ employs ____" i buchi si riempiono con Ann e Bob. Sono quindi dei placeholder per gli oggetti. Quando un oggetto partecipa in una relazione, allora quel participare è il ruolo. L'ordine ha importanza per le azioni. Gli oggetti possono avere object type diversi così come uguali. Bisogna raccogliere tutte le info a disposizione ed analizzarli per identificare sinonimi, casi di studio ecc e costruire una forma verbalizzata del sistema come è stato descritto dagli esempi. A questo punto si processano le verbalizzazioni modellando andando ad individuare i fatti elementari e scomponendo quelli composti. Si riscrive quindi il sistema come dato dagli esempi in termini di fatti elementari. Questo processo serve a capire quale parte del dominio è quella rilevante. Poi c'è la fase di rielaborazione perchè magari gli esempi non rappresentano tutto quello che serve, quindi si va ad arricchire con altri fatti elementari per colmare le mancanze. (vedi esempio slide 10 in orm.1)

Passo 2:

Disegnare i fact type e applicare la verifica degli esempi che dispongo.

1. Draw an **instance diagram** from the factual information obtained so far.
2. Generalize the instance diagram to a **conceptual schema diagram (structural schema)**.
3. Validate the correctness of the conceptual schema diagram with **sample population**
→ **conceptual model** or **conceptual knowledge base**.

La validazione coinvolge delle query per appunto verificare il rispetto dei vincoli.

Per la modellazione a partire dagli esempi prima si esegue il passo 1 precedentemente descritto andando ad eseguire la verbalizzazione dei dati, quindi definiamo le entità coinvolte.

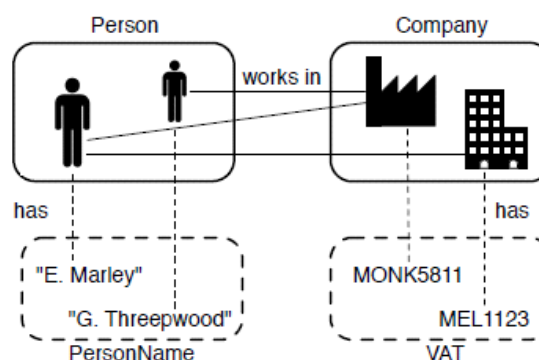
ESEMPIO:

| Person | Company |
|---------------|---------|
| G. Threepwood | MON5811 |
| E. Marley | MEL1123 |
| E. Marley | MON5811 |

- The Person named 'G. Threepwood' *works in/employs* Company with VAT 'MON5811'.
- Person (.Name) 'E. Marley' *works in/employs* Company (VAT) 'MON5811'.

In questo esempio vediamo che gli entity type sono Person e Company che quello che sta in company è il VAT o codice fiscale mentre in Person c'è il nome. Nome e VAT sono i reference mode. Avremo come verbalizzazione " Person (.Name) 'E. Marley' works in/employs Company (VAT) 'MON5811'".

Sottoforma di diagramma delle istanze otteniamo questo:

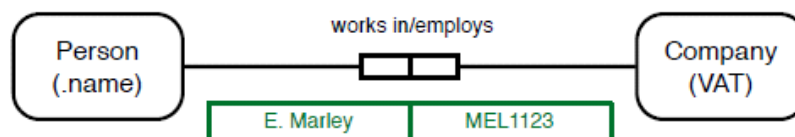


All'interno del box tratteggiato sono i value statici.
(vedi quad 4))

- Conventions:
 - Binary roles: optional inverse reading (separated from the mandatory one by '/').
 - N-ary roles ($n > 2$): ellipsis '...' to represent object holes.
- How many (alias) readings for n-ary roles?
 - In general? $n!$ (permutations)
 - Displayed? **1** solo una, non tutte
 - To easily query the schema? n
- Guideline: define inverse reading for binary role, alias readings for n-ary roles only when needed.

Tipi di fatti elementari perchè type è una generalizzazione e alcuni tra questi fatti elementari ci sono reference che servono ad utilizzare un valore per individuare un oggetto specifico(le box tratteggiate). Queste sono relazioni tra entità e valori e sono chiamati fatti esistenziali. Quando facciamo individuazione è uno schema 1:1 quindi vi è univocità.

In maniera compatta la reference si rappresenta così:



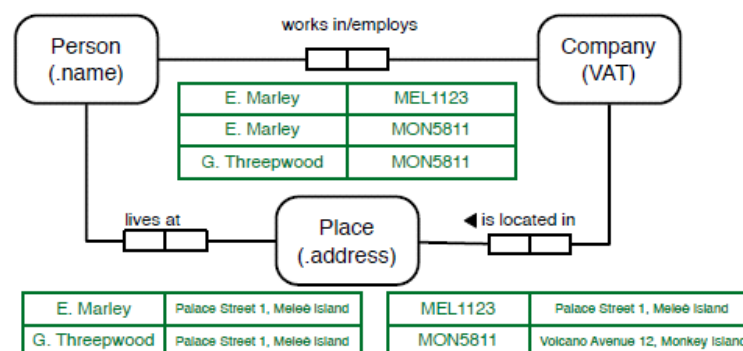
VAT non ha il punto perchè è un concetto che esiste come istanza esterna.

I riquadri verdi sono la popolazione della relazione.

Ci sono 3 tipi di reference mode:

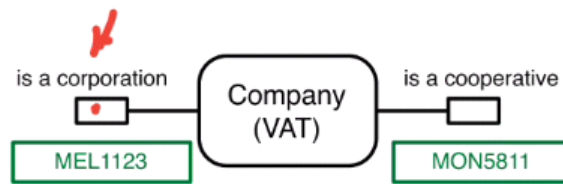
- POPOLARE: associato ad una popolazione, quindi ad un entity type e serve proprio ad identificare gli elementi di una popolazione.
- MISURE: serve appunto per indicare l'unità di misura di un oggetto. È indicato con i due punti dopo la misura(EUR:)
- GENERALE: come il codice fiscale, gli isbn, ossia tutti valori di identificazione di tipo generale che hanno regole specifiche.

Quando lo schema è associato a dei fatti abbiamo il diagramma della base della conoscenza, ossia **lo schema + la popolazione corrispondente sottoforma di tabella**. I fatti corrispondono alle istanze originali dei fact types. Supporta la validazione per verificare se i vincoli vengono soddisfatti. Inoltre si dovrebbe andare a provare concretamente almeno qualche esempio per ognuna delle reazioni per verificarne il corretto funzionamento.



Nell'esempio le fact tables sono quelle in verde. Il triangolino nella relazione 'is located' indica che si legge dal verso opposto.

È possibile aver anche relazioni con un solo ruolo, sono detti fatti unari, affermazioni di verità. Quando ho una relazione unaria, vado a specificare una caratteristica dell'entity type.



Se ci sono più fatti unari per un certo entity type potrebbe essere rappresentato con uno stato e quindi diventando una relazione binaria. Questo solo se gli elementi sono simili, altrimenti no. I fact types possono essere eterogenei, quando coinvolgono oggetti dello stesso tipo, oppure omogenei se invece i ruoli sono giocati dallo stesso object type.

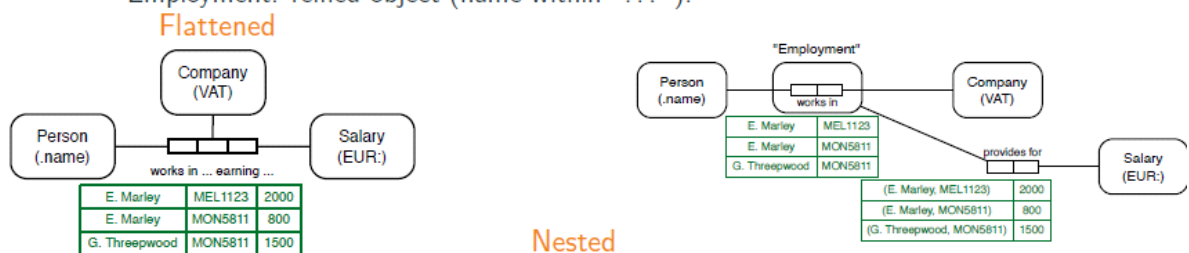
Una relazione può essere oggettificata, ossia le si può assegnare un nome quindi trattandola come un oggetto vero e proprio che può essere coinvolto in relazioni. Questa operazione è detta REIFICAZIONE.

- Person (.name) 'E. Marley' works in Company (VAT) 'MEL1123' earning a Salary (EUR:) of 2000.

vs

- Person (.name) 'E. Marley' works in Company (VAT) 'MEL1123'. This Employment provides for a Salary (EUR:) of 2000.

- Employment: reified object (name within "...").



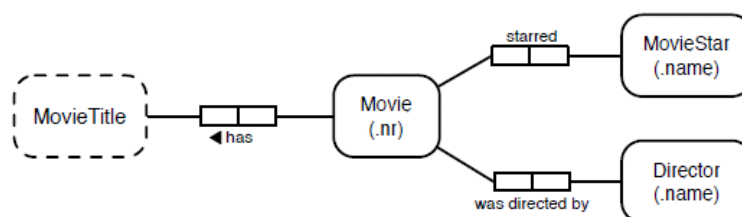
Sotto certe condizioni la versione annidata e quella no sono equivalenti, ma quale delle due usare in quel caso dipende dalla modellazione.

Passo 3:

Verifica degli entity type che possono essere combinati e verificare anche le derivazioni aritmetiche. Molto semplice ma abbastanza fondamentale perchè gli entity type sono tutti come tipi primitivi, ossia gli oggetti devono essere separati gli uni dagli altri quindi le istanze non devono poter essere categorizzate in più entity type. Il passo di combinarli quindi è importante perchè ci si rende conto di correlazioni.

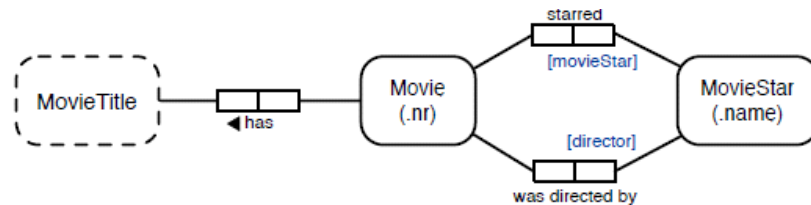
L'universo del dominio è quindi partizionato in due: valori ed entità. Le entità a loro volta sono partizionate in tipi primitivi, che sono il top level, quindi mai si devono sovrapporre i tipi primitivi. I sottotipi esistono. I valori dei top level però possono coincidere.

Esempio:

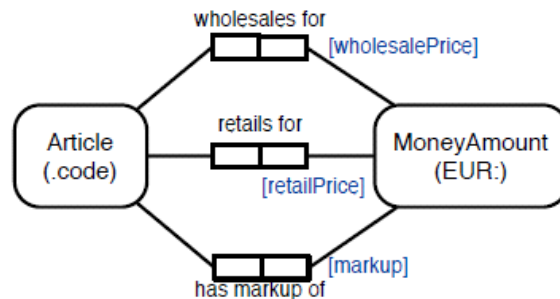


In questo caso ci sono due entità diverse che rappresentano attore e regista del film, però ci sono casi in cui il regista fa anche da attore e viceversa, e in questo modo non è rappresentabile perchè non potrebbe avere due entity type contemporaneamente. Si risolve andando quindi ad usare una sola entità e specificando poi director o movieStar come nome del ruolo. Combino quindi due entità

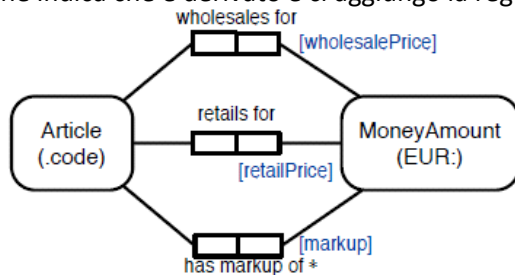
in una sola.



Esempio:



In questo caso possiamo vedere come il guadagno è il prezzo di vendita - il prezzo all'ingrosso, il che vuol dire che markup è derivabile aritmeticamente dagli altri due, e questa cosa devo segnlarla con un *, che indica che è derivato e ci aggiungo la regola di derivazione.



$$\begin{aligned} \text{markup} &= \text{retailPrice} - \text{wholesalePrice} \\ \text{retailPrice} &= \text{markup} + \text{wholesalePrice} \\ \text{wholesalePrice} &= \text{retailPrice} - \text{markup} \end{aligned}$$

Se invece metto il + vicino alla relazione indico una semidrivazione, ossia più valori dipendono l'uno dall'altro ma implico che uno sia sempre derivato dagli altri, ma non da la priorità su quale dipende dagli altri.

Nelle relazioni numeriche devo identificare come scrivere le regole di derivazione:

Example (attribute style)

for each Article, $\text{markup} = \text{retailPrice} - \text{wholesalePrice}$

Oppure

Example (relational style)

Article has markup of MoneyAmount iff
 Article retails for MoneyAmount₁ and
 Article wholesales for MoneyAmount₂ and
 $\text{MoneyAmount} = \text{MoneyAmount}_1 - \text{MoneyAmount}_2$

Il ** significa che è derivato all'update, quindi lo metto nel momento in cui faccio l'update dei dati quando ci sono gli altri due, mentre un solo * significa che li ottengo quando faccio la query calcolandolo in quel momento.

Passo 4:

Prevede l'aggiunta di vincoli. I vincoli che inseriamo per primo sono quelli di unicità per le relazioni.

Le relazioni sono degli insiemi di coppie, terne ecc dove i vari elementi in relazione giocano dei ruoli. Ma quale è la molteplicità di un ruolo che una entità può giocare?

Ad esempio la persona è relazionata a cognome e nome, in questo caso posso avere un solo nome-cognome, quindi la molteplicità è 1. Lo stesso nome-cognome può essere di più persone, quindi non c'è un vincolo di unicità.

Vincoli di Unicità

Questi vincoli permettono anche di verificare la correttezza dei fact type: abbiamo dei fatti che abbiamo affermato e questi sono elementari e il vincolo di unicità ci aiuta a capire se quel fatto è elementare o no.

Quindi mi aiuta a capire se terne o altro sono scomponibili ulteriormente in fact types.

Il vincolo di unicità ci dice che al massimo un fact type di un certo tipo è concesso e permette di identificare le chiavi dei fact types.

I fact type fotografano un certo istante della realtà e questi devono essere fatti costantemente veri.

Se prendiamo la persona e il peso possiamo dire che una persona può pesare x1, x2 o x3, ma non tutti e tre contemporaneamente. Quindi introduciamo un esplicito riferimento al tempo per poter permettere di assegnare alla stessa Persona più pesi, perchè appunto sono variazioni che avvengono nel tempo, garantendo l'unicità dell'affermazione.

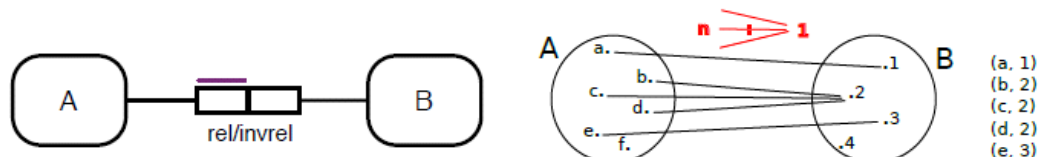
Ci sono tre tipologie di relazioni su cui possiamo applicare i vincoli di unicità :

- **Relazione unaria:** la relazione unaria è caratterizzata dal fatto che un certo elemento gioca un certo ruolo se è vero quel ruolo per quell'elemento. Quindi affermo un fatto. Quando un entity type ha una relazione unaria, gli elementi che compaiono in essa possono comparire solo una volta, quindi ogni relazione unaria ha implicito un vincolo di unicità nel ruolo perchè altrimenti avere più elementi uguali non ha senso perchè $[a \text{ and } a = a]$. Questi elementi sono in sostanza degli insiemi, e come insiemi appunto sono contati una sola volta. In ORM si indica che un ruolo è giocato in modo unico da un entity type si rappresenta così:

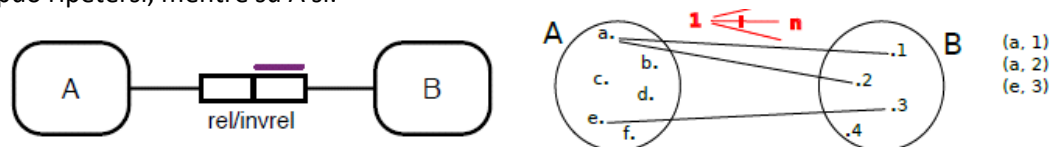


Vuol dire che gli elementi che compaiono qui possono esserci solo una volta. Nonostante sia implicito, va segnato.

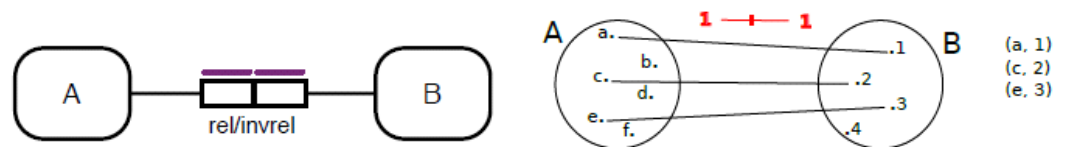
- **Relazione binaria:** ci sono 4 casi che corrispondono a:
 - **Molti-a-Uno(n:1)**, cioè un insieme di elementi può essere mappato su un singolo insieme. Più elementi di A vanno sullo stesso elemento di B. Quindi per B il valore può ripetersi e non è unico questo ruolo, invece per A il valore non si ripete quindi è unico.



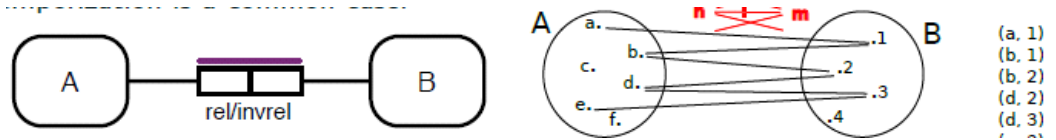
- **Uno-a-Molti(1:n)**, uno può andare su molti. Un elemento di A può essere associato a più elementi di B, il vincolo di unicità va su B. In questo caso il valore su B non può ripetersi, mentre su A si.



- **Uno-a-Uno(1:1)**, un elemento solo con un solo elemento. Ad un elemento di A è associato al più solo un elemento di B e da B al più un solo elemento di A. Nelle coppie che sono in relazione gli elementi compaiono una volta sola.

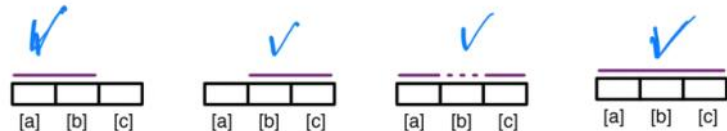


- **Molti-a-Molti($n:m$)**, molti elementi possono essere accoppiati con molti elementi e viceversa. Un elemento di A può essere associato a più elementi di B e viceversa. Le coppie possibili presentano duplicati sia per A che per B. La coppia però può comparire una volta sola perché è un fatto, perché appunto essendo fatto elementare non ha senso dirlo più volte, quindi è la coppia stessa ad avere vincolo di unicità.

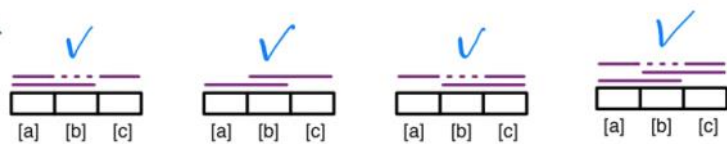


- **Relazione ternaria**: sicuramente possiamo dire che la terna può essere unica, ma è possibile avere coppie nella terna che hanno vincoli di unicità? Relazione ternaria con un vincolo di unicità binario è possibile. Questo è l'equivalente ad avere una relazione molti a molti con i ruoli coinvolti nel vincolo di unicità binario, effettuare l'oggettificazione di questa relazione e andare poi a relazionarlo con una relazione con il terzo ruolo come relazione Molti a Uno.

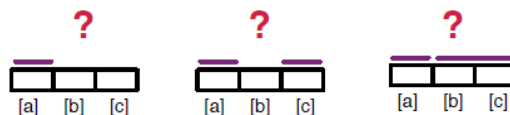
- **Single UCs.**



- **Combined UCs.**



Gli altri casi possibili possono essere:



In via generale hanno senso di esistere, ma andrebbero ad indicare il fatto che quello rappresentato non è un fatto elementare la terna.

Il vincolo di unicità di una relazione ci dà un criterio per individuare anche i fatti non elementari. Difatti se trovo un vincolo di unicità su un solo ruolo di una relazione ternaria, questo mi è sufficiente per dire che quella non rappresenta un fatto elementare. Un secondo metodo per individuare questo è la tecnica di proiezione-join: se riesco a dividerle in due e avere le stesse informazioni iniziali allora vuol dire che è decomponibile in fatti elementari.

Controllo sulla chiave

Mi permette di individuare i predicati troppo lunghi. Ogni fatto n -ario ha almeno una chiave di lunghezza $n-1$. quindi ho n ruoli, allora deve esistere una chiave che ha almeno $n-1$ ruoli coinvolti in essa, se non è così non è elementare. Quindi se non riesco a trovare un vincolo su $n-1$ ruoli allora è decomponibile perché non è un fatto elementare.

Quando abbiamo una combinazione di colonne X e Y in una fact table, allora Y dipende funzionalmente da X , il che vuol dire che la colonna X determina i valori della colonna Y , se per ogni valore della colonna X c'è al più un valore della colonna Y .

Ma se il valore di X colonne determina funzionalmente una colonna Y , allora al più c'è un valore associato a X .

Abbiamo quindi un vincolo di unicità per le X colonne.

Le dipendenze funzionali, ossia data una persona so la sua età o data una persona so la sua residenza, se non hanno una chiave che coinvolge $n-1$ colonne della relazione, vado a violare

quello che era il vincolo del controllo sulla chiave.

Le dipendenze funzionali nascondono la presenza dei vincoli di unicità, quindi devono essere analizzate. (RIVEDI DAL LIBRO)

Questo è un modo strutturale per individuare se non sono fatti elementari.

Proiezione e join concettuale

Nella proiezione concettuale si va a proiettare le colonne di interesse. Quando si vanno a considerare alcune delle colonne, si ottengono sempre delle fact table quindi gli elementi devono essere sempre senza duplicati.

Il join permette di derivare una nuova relazione da due relazioni precedenti scegliendo uno o due elementi come pivot che devono essere comuni alle due fact table che vogliamo unire. La relazione ottenuta dal join deve essere segnata con un * per ricordare che è derivata. Se si riesce ad individuare un pivot in una relazione ternaria che mi permette di ottenere due relazioni che quando faccio il join mi danno la stessa fact table vuol dire che non è un fatto elementare e quindi posso splittare la relazione ternaria, questo perchè non perdo informazione.

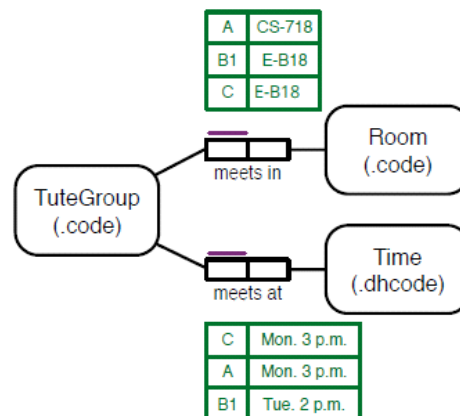
In questa modalità bisogna chiaramente analizzare gli esempi veri e propri.

(vedi esempio slide 21-23)

Nell'esempio facendo gli split con le proiezioni e poi facendo il join degli split fatti se si ottiene la fact table iniziale allora non si ha un fatto elementare e si deve separare.

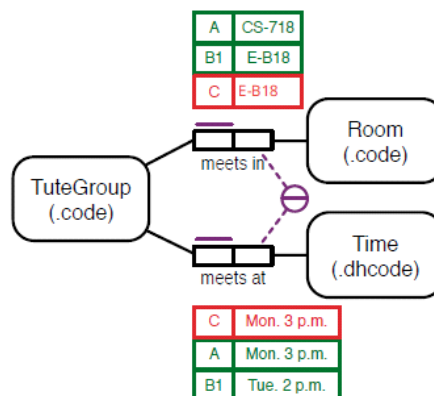
External Uniqueness Constraints...

Nell'esempio sottostante si può notare come TuteGroup ha una stanza e un'ora e queste sono due info indipendenti tra loro, cioè il gruppo di tutoraggio si riunisce sempre nella stessa stanza e sempre nella stessa ora. Però se dico una stanza e un'ora, so dire quale è il gruppo di tutoraggio che si riunisce perchè la sua stanza e la sua ora sono sempre le stesse, quindi è identificabile dalle altre due informazioni.



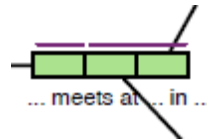
Per rappresentare quindi questa condizione lo rappresento in questo modo:

...



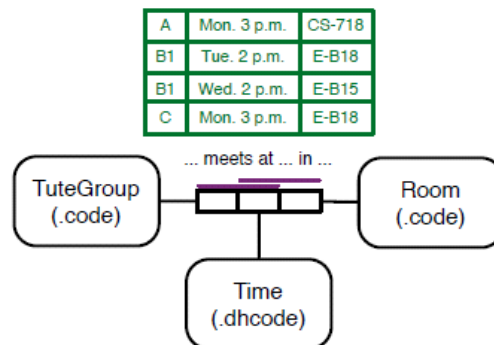
Each combination of MeetingRoom and MeetingTime is paired with at most one TuteGroup

È quindi derivabile una tripla con questo tipo di vincoli:

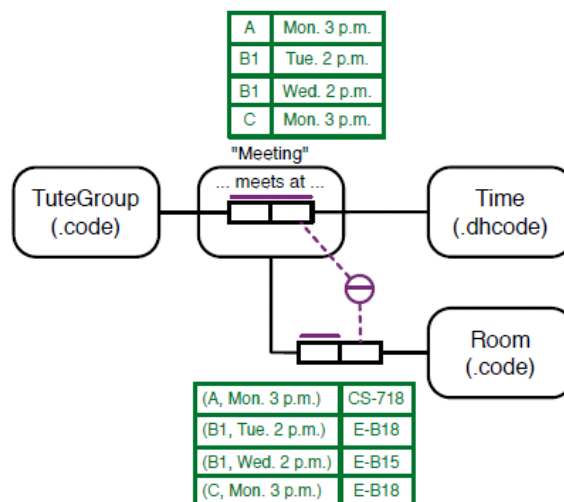


... and Objectification

Se si ha questo tipo di situazione:



Vediamo presente un vincolo doppio su una ternaria sulle colonne a due a due. Significano che dato un gruppo di tutoraggio e un'ora, posso dire la stanza e se dico un'ora e la stanza posso identificare un gruppo di tutoraggio. Questi due vincoli possono essere trasformati in questo modo, andando ad eseguire l'oggettificazione della coppia TuteGrup-Time, la relazione rimanente diventa relazione con l'oggettificazione. Facendo così dico che dando una coppia Gruppo-Ora posso dire la stanza, il che è giusto, quindi rappresenta la dipendenza funzionale della coppia con la stanza. Per dire che questo è equivalente a un vincolo esterno in questo modo:



Il vincolo di unicità esterno serve a rappresentare questa situazione in maniera equivalente.

Passo 5:

un altro vincolo fondamentale è il vincolo di obbligatorietà per un ruolo. Una certa entity type se ha un vincolo di obbligatorietà, significa che gli elementi dell'entity type devono partecipare per forza a quella relazione. È diverso dal dire che tutte sono elencate nella relazione, ma se siamo a conoscenza nell'information system dell'elemento, questo deve giocare un ruolo.

Ad esempio potremmo dire che è necessario che una persona abbia un codice fiscale. Questo non significa che devo mettere nelle fact table tutte quelle esistenti, ma se ho inserito una persona, devo necessariamente avere quell'informazione.

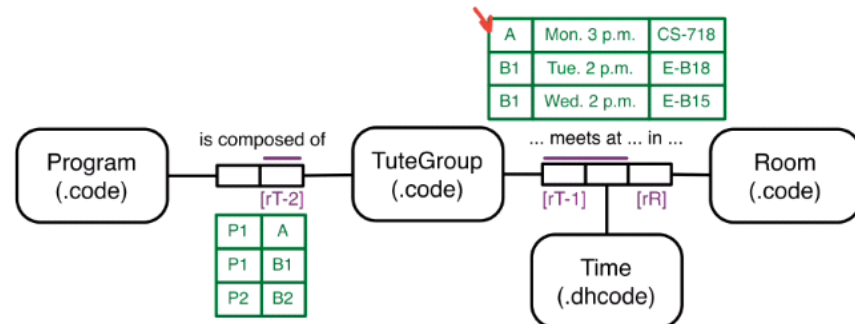
Inoltre questo step controlla eventuali derivazioni logiche.

Il vincolo di unicità era "al più un elemento che compare", questo p al contrario : se un elemento

gioca un ruolo, deve giocarlo minimo una volta.

Es:

Un programma che è identificato da un codice è composta da gruppi di tutoraggio, anch'essi identificati da un codice. Un gruppo di tutoraggio ha associato un programma da seguire, mentre lo stesso programma può essere adottato da più gruppi. Il gruppo di tutoraggio si incontra a una certa ora in una certa stanza.



In base allo stato attuale dei fact table conosco del gruppo di tutoraggio A, B1, B2. come possiamo notare il B2 ha un programma ma non si hanno info su quando si incontrano.

Dato un ruolo, possiamo definire l'insieme di valori giocati da un entity type per quel ruolo. Per questo ruolo, ad esempio il primo di TuteGroup è formato da (A,B1) mentre il secondo ruolo è giocato da (A,B1,B2). Il valore mi dà i reference mode, ossia l'insieme dei valori che identificano l'elemento dell'entity type.

$$\bullet \text{val}(rT-1) = \{ 'A', 'B1' \}$$

quindi dato un insieme di valori che sono quelli che compaiono nel ruolo, identifico la popolazione come l'insieme degli entity type referenziati da quel valore che giocano quel ruolo.

$$\bullet \text{pop}(rT-1) = \{ \text{TuteGroup}(.code) 'A', \text{TuteGroup}(.code) 'B1' \}$$

In questo momento posso dire che la popolazione del tutor group è data dall'unione dei valori dei vari ruoli giocati dall'entity type. Quindi :

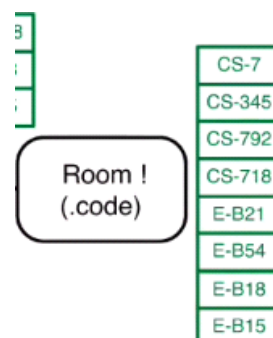
$$\bullet \text{pop}(\text{TuteGroup}) = \text{pop}(rT-1) \cup \text{pop}(rT-2) = \{ A, B1 \} \cup \{ A, B1, B2 \} = \{ A, B1, B2 \}$$

Questo non significa che questi sono tutti gli elementi, ma sono tutti quelli che io conosco al momento.

Supponiamo che io voglia dire che io posso aggiungere tutti gli elementi che voglio, purché questi siano all'interno di un certo insieme. Supponiamo che in orario si voglia aggiungere un nuovo appuntamento: si possono aggiungere tutte le terne che voglio purché le aule siano quelle che conosciamo. Voglio quindi specificare quale è la popolazione.

Questo si chiama Independent Entity Type.

Per farlo si indica un punto esclamativo (!) nell'entity type e si mette la fact table di tutte quelle che si possono usare.



Serve ad indicare che questo è un dettaglio importante e che posso scegliere solo tra quello che è disponibile, e non mettere valori casuali, perciò definiamo un vero e proprio vincolo. Queste sono

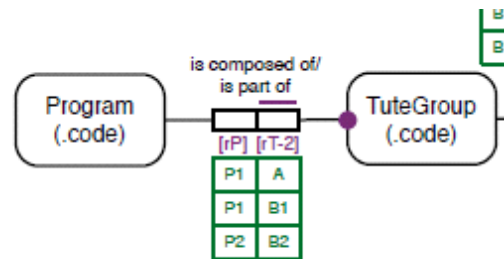
immagazzinate diversamente dalle relazioni, cioè non esiste necessariamente un coinvolgimento degli elementi nelle relazioni nelle fact table.

La verbalizzazione è: "The Room with name 'CS-7' exist".

Possiamo anche mettere un vincolo di essitenza su un'oggettificazione. Ad esempio elenco la coppia TuteGroup-Hour e le coppie che elenco sono tutte quelle che esistono.

Supponiamo che vogliamo dire che se un gruppo di tutoraggio esiste, necessariamente debba avere un programma.

In sostanza la popolazione di un certo entitu type deve giocare un ruolo. Quindi se c'è un elemento di quell'entity type nella popolazione, c'è anche in quel ruolo. Cioè la popolazione di un entity type corrisponde alla popolazione di un certo ruolo. Se esiste nella popolazione deve necessariamente avere quel ruolo "per esistere". Questo si rappresenta così:



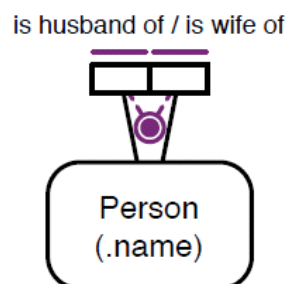
Nell'esempio significa che non posso mettere un incontro di un gruppo ad una certa ora se quel gruppo non ha un programma, quindi il ruolo di TuteGroup nella relazione con Program deve esserci obbligatoriamente. Questo è detto vincolo MANDATORY. c'è quindi una sorta di dipendenza tra la fact table con il mandatory e le altre.

I ruoli con il vincolo mandatory sono obbligatori, gli altri sono opzionali.

Se si ha sia il vincolo di unicità che quello di obbligatorietà, questi uniti creano l'EXACTLY ONE. Cioè se ci deve essere al più un elemento e deve essercene almeno uno, vuol dire che la verbalizzazione è "per ogni.... Ci deve essere esattamente un.....".

- In general, 0..1 constraint + 1..* constraint → 1-1 constraint.
- 1-1 constraint is a bijection.

Il mandatory può anche essere associato a due ruoli associati allo stesso entity type.



In questo caso possiamo dire che questa rappresenta l'insieme delle persone sposate.

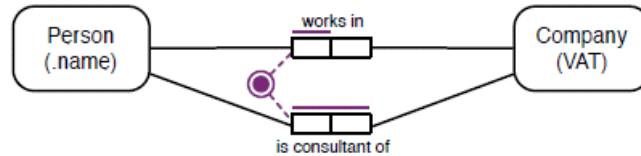
Per ogni entità non-indipendente(senza !), la popolazione è quella di tutti i ruoli in cui è coinvolta. Invece la popolazione di quelli indipendenti è data dalla fact table associata. Il mandatory di solito non si associa ad essa perchè allora la popolazione di quel ruolo deve essere la fact table e quindi dovrebbe metterle già tutti. Se metto un mandatory associato ad un entity type indipendente, in pratica perde l'indipendenza.

Implicitamente c'è un mandatory che è sparso su tutto. Questo perchè se la popolazione è data dall'unione dei valori dei ruoli in cui è coinvolto l'entity type, vuol dire che sicuramente tutti i membri della popolazione dell'entity type in questione giocano almeno un ruolo tra quelli in cui l'entity type partecipa. Questo è rappresentato da un vincolo esterno con un cerchio e all'interno un cerchio pieno.

Il vincolo non deve essere esplicitato perchè se dovessi modificare lo schema potrebbe avere un impatto sulla nuova relazione, a meno che non sia fondamentale l'obbligatorietà che viene imposta in quel caso.

Es:

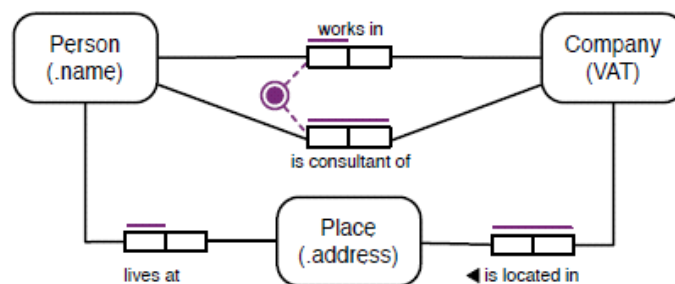
Una persona lavora in una certa azienda oppure è consulente dell'azienda, oppure entrambi. Se metto il vincolo di disgiuntive mandatory vuol dire che gli vado a dire che le persone che lavorano per l'azienda sono necessariamente consulenti per quella stessa azienda.



In questo caso se dovessi aggiungere una nuova relazione con ad esempio University e Person, tutte le persone che sono segnate nella fact table di questa relazione dovrebbero necessariamente essere anche persone che lavorano, quindi avrebbe senso se si stesse segnando solo gli studenti lavoratori, però per gli studenti in generale non andrebbe bene perchè non è detto che tutti gli studenti sono anche lavoratori.

La relazione mandatory disgiuntiva ha senso solo se dopo aggiungiamo informazioni.

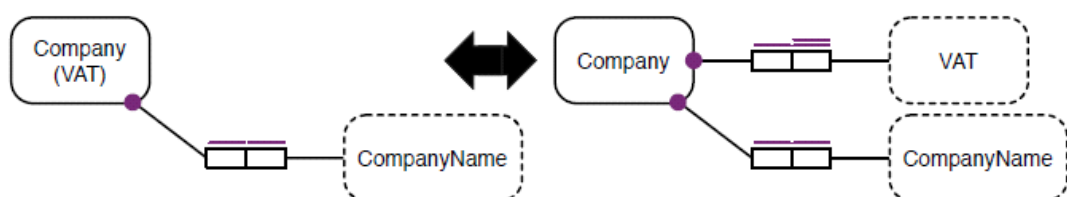
Es:



Sto indicando chiaramente che quelli di cui mi chiedo l'indirizzo sono solo le persone che lavorano come consulenti o semplicemente in azienda, e non di qualsiasi persona. Quindi non ci metto nella fact table di "lives at" una persona che non lavora per Company.

REFERENCE SCHEMA

Supponiamo che company sia in relazione con CompanyName e un nome può essere associato ad una sola azienda. Una azienda può avere un solo nome e una azienda deve necessariamente avere un nome. Da questo posso dedurre che, dato un nome da quelli elencati, posso identificare l'azienda: questo lo rende in effetti un reference mode. L'azienda ha però un suo reference mode, allora se volessimo esplicitarlo con una relazione diverrebbe che company è in relazione mandatory 1:1 con VAT.



Lo schema a sinistra ci dice che VAT viene preferito tra i due come reference mode, quindi questo a sinistra è il preferred reference schema. Se volessimo esplicitare questa preferenza dovremmo indicarlo come a destra, andando a mettere due linee sul ruolo di VAT nella relazione con Company.

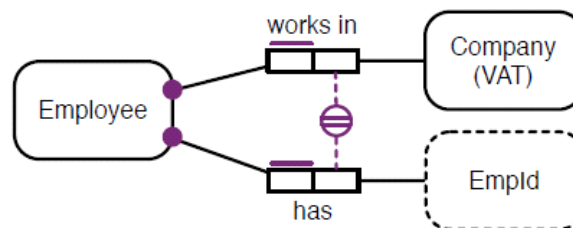
Possiamo anche avere un vincolo di unicità su una coppia: dato un ID per un impiegato e una azienda

identificata da un VAT noi identifichiamo un impiegato.
Partendo da questo :



In questo modello sto dicendo che la matricola dell'impiegato è diversa per ogni azienda perchè non dipende da essa.

Nel secondo modello(sotto) invece sto dicendo che c'è una matricola, può essere diversa da azienda ad azienda e gli impiegati sono identificati da un numero matricola e l'azienda a cui la vado ad applicare. Questo è un reference mode di coppia che vado ad indicare in questo modo:

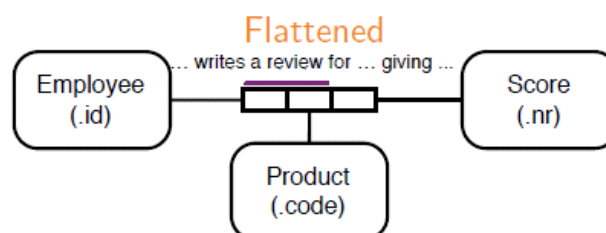


Allora il vincolo di unicità quando è esterno ed è anche reference mode, lo si scrive con la doppia linea. Quindi gli Employee sono identificati dalla coppia VAT- Empld.

Per scegliere il reference mode bisogna basarsi sulla realtà rappresentata, ma bisogna anche evitare di complicare lo schema quindi evitare quando possibile i vincoli esterni e preferire gfi identificatori rigidi.

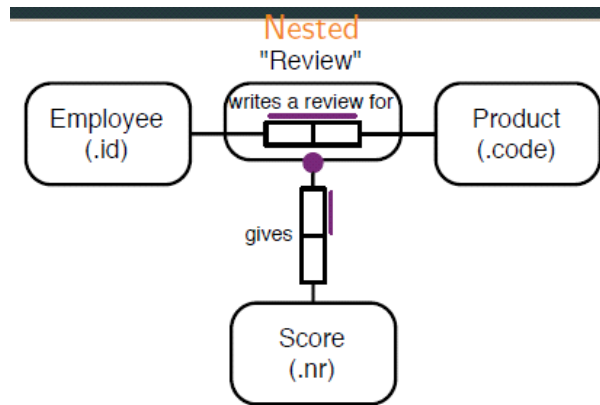
Es: supponiamo di avere questo schema dove rappresentiamo un certo impiegato, identificato da una matricola, un Prodotto e uno Score.

La relazione rappresenta un impiegato che scrivere la recensione di un certo prodotto dando un determinato Punteggio. La rel è elementare; lo stesso voto può essere dato a più prodotti e splittando no riesco ad avere la stessa informazione. Questa è detta rappresentazione appiattita dell'informazione.

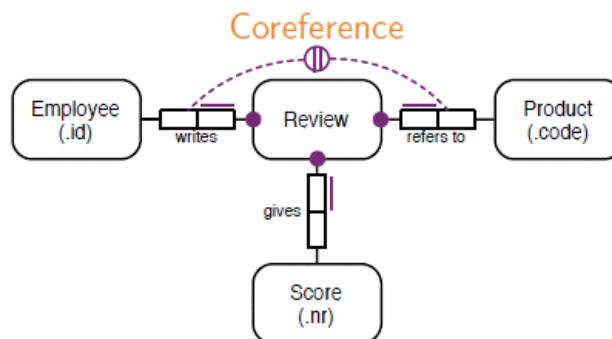


Può essere rappresentata anche in forma annidata dove la relazione chiave tra Impiegato-Prodotto viene oggettificata e poi viene aggiunta una con Score. Per rendere le due versioni assolutamente e equivalenti bisogna mettere un vincolo mandatory all'oggettificazione verso Score. Questo perchè tutti gli elementi hanno anche un voto. Se non metto un mandatory potrebbero esserci delle recensioni senza voto, e allora la relazione sarebbe diversa, perchè tutte le triple sono formate da impiegato-prodotto-voto.

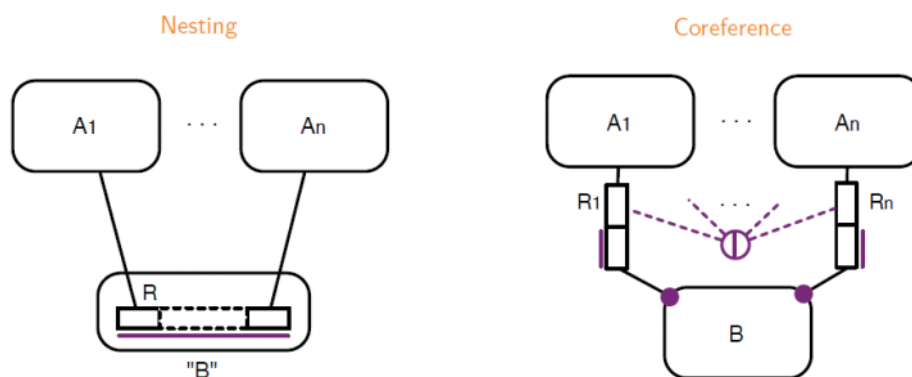
In questa versione esplicitiamo il concetto recensione e la voglio proprio chiamare "Recensione"



Un'altra versione è detta di Coreference. In questa versione diciamo che la recensione è un oggetto reale (tipo un vero foglio di carta che esiste di per sé) e quindi ha una sua entità. In questo caso abbiamo 3 relazioni diverse con le tre entità Employee, Product e Score. Per ciascuna c'è il vincolo mandatory perchè devono essere tutte presenti per poter rappresentare le stesse informazioni della versione appiattita. Diciamo anche che dato un impiegato e un voto, identifico il punteggio, e questo è anche lo schema di riferimento.

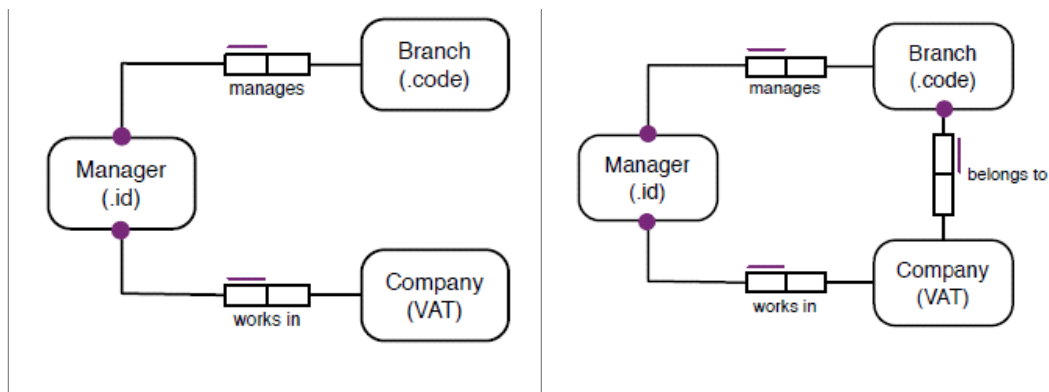


In generale:



DERIVAZIONI LOGICHE

Visto che mettiamo tanti elementi, potremmo chiederci se non ci siano derivazioni. Ad esempio i manager sono identificati da matricola e obbligatoriamente gestisce un solo ramo di una azienda, e un manager lavora per una azienda ed è una sola. Ma allora il ramo è dell'azienda visto in questo modo. In questo caso significa che c'è una relazione derivata.



Questo però non è semplice perchè ci abbiamo messo della semantica di mezzo andando a ragionare sul significato di quello che è rappresentato. Schematicamente non c'è una regola precisa per capire quando ci sono derivazioni logiche.

PASSO 6:

Permette di aggiungere dei vincoli di valore, di sottoinsieme, di equaglianza, di esclusione e di sottotipo.

I vincoli di valore possono assumere gli entity type dei valori, però possono essere inseriti nei vincoli sugli entity type ma anche quali si possono usare per giocare dei ruoli.

I vincoli insiemistici tra i vari ruoli di stesso tipo oppure su sequenze di ruoli uguali e sono subset, equaglianza e sottotipo. Come la popolazione di un ruolo o la sequenza di ruoli si relaziona con la popolazione di altri ruoli.

- **Subset** (\subseteq): $\text{pop}(r_1) \subseteq \text{pop}(r_2)$.
- **Equality** ($=$): $\text{pop}(r_1) = \text{pop}(r_2)$.
- **Exclusion** (\times): $\text{pop}(r_1) \cap \text{pop}(r_2) = \emptyset$.

I vincoli di sottotipo sono specializzazione o generalizzazione di un certo object type che ha aspetti comuni a quello che estende/generalizza.

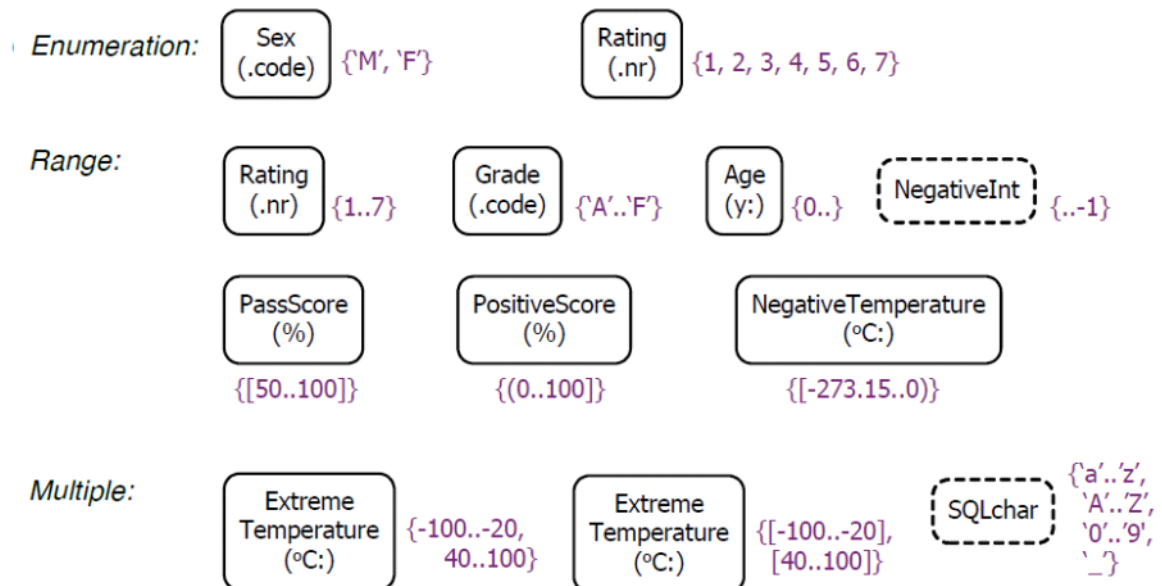
VINCOLI DI VALORE

Vincolano i possibili valori che possono essere assunti da un ruolo o da un tipo. Questo si consiglia di usarlo quando è stabile, ossia quando l'insieme di valori lo conosciamo e sappiamo per certo essere quello. Viene indicato con delle parentesi graffe {...}. Possono essere indicate delle enumerazioni tramite elenco dei valori oppure tramite indicazione del range di valore.

Se è un vincolo dei valori di un object type, questo vincolo lo metto attaccato all'object type, se invece è un vincolo dei valori di un ruolo lo metto vicino al ruolo.

Se si ha un object type indipendente, ossia con !, lo usiamo se si vogliono immagazzinare informazioni riguardo un object type la cui lista di valori potrebbe essere molto lunga e impossibile da inserire nel modello, oppure se è instabile o non noto a priori l'insieme di valori.

Es:

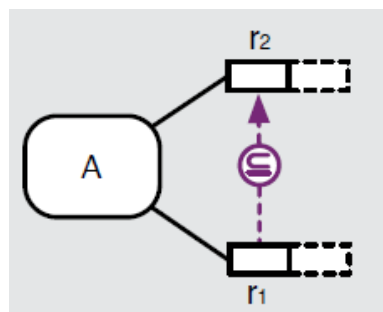


Una persona guadagna un salario gestendo una azienda. Il valore del Salario non può essere negativo, infatti si mette il vincolo accanto all'entity. Se sono manager magari ho un salario minimo, quindi si mette un vincolo sul ruolo perchè solo se gestisco una azienda vale quel vincolo.

VINCOLI DI SOTTOINSIEME

Tra i ruoli.

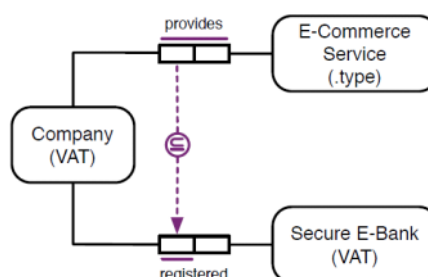
Se volessi esprimere il vincolo che la popolazione di un certo ruolo R1 è inclusa nella popolazione del ruolo R2, ovviamente ruoli giocati dallo stesso entity type, dovrei usare il vincolo di sottoinsieme così indicato:



La freccia va da quello incluso verso quello che include. Questo vincolo è puramente un confronto insiemistico e quindi per i duplicati non vale come cosa.

Es:

Una azienda si può registrare al più su un servizio di sicurezza E-Bank e una azienda può offrire un servizio di E-Commerce. Se una azienda offre l'E-Commerce, allora deve essere registrata su un servizio di sicurezza E-Bank.



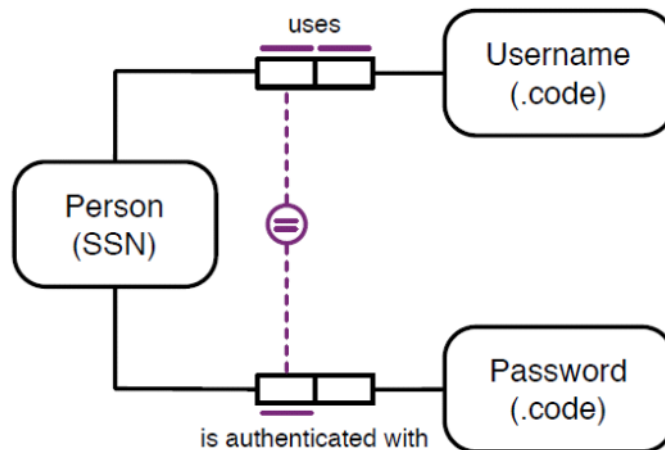
VINCOLO DI UGUAGLIANZA

Come l'inclusione, ma invece di dire che una popolazione di R1 è inclusa in quella di R2, diciamo che

sono esattamente uguali. Quindi una gioca un ruolo se e solo se lo gioca l'altro. Se metto un mandatory su uno dei ruoli giocati, allora implicitamente è come se lo avessero entrambi, quindi non possono esserci un vincolo mandatory per un ruolo e uno opzionale per l'altro. L'uguaglianza ha senso metterla solo se si tratta di ruoli opzionali.

Es:

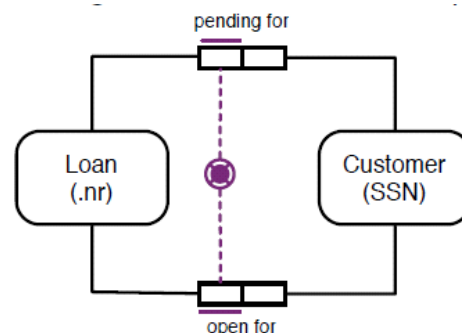
- A Person could use an Username, and be authenticated with a Password.
- Either a Person does not have a Username nor a Password, or the Person has both.



VINCOLO DI ESCLUSIONE

Cioè la popolazione di R1, intersecata alla popolazione del ruolo R2 è un insieme vuoto. Cioè per ogni A al più uno dei due ruoli vale, è l'exclusive OR praticamente. Anche qui non ha senso se c'è un mandatory solamente, vale solo se sono tutti opzionali i ruoli.

Ci sono casi in cui almeno uno dei due ruoli deve essere giocato, in questo caso si mette il vincolo mandatory distribuito. Questo da solo però ci dice che potrebbero essere entrambi, perciò si associa l'exclusive OR che ci dice che uno per forza deve essere giocato, ma mai insieme. Esiste per indicare questo un simbolo apposito:



Nell'esempio è rappresentato il fatto che ogni prestito è o aperto o in attesa di apertura, mai insieme e necessariamente deve essere in uno dei due stati. (vedi loan example pag 13)

I vincoli possono essere associati anche a coppie di ruoli.

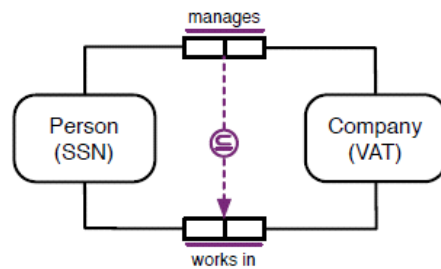
Pair-subset constraint

For each information base state: $pop(r_1, r_2) \subseteq pop(r_3, r_4)$.

Each pair in $pop(r_1, r_2)$ is also in $pop(r_3, r_4)$.

Es.

- Consider Persons who work in Companies and Persons who manage Companies.
- Clearly, Each Person who manages a Company works in that Company.



Quando si introducono questi vincoli si devono rivedere le inconsistenze o ridondanze presenti.
Casi tipici:

- Se abbiamo un mandatory su un ruolo R2 e uno opzionale R1 l'inclusione è implicita in R2 per definizione perchè comprende tutti i ruoli che devono essere obbligatoriamente assunti dall'entità
- Se ho due mandatory e si ha il vincolo di uguaglianza vi è una ridondanza perchè sono sicuramente uguali per definizione data la presenza del mandatory
- Se ho un inclusive OR e poi un vincolo di sottoinsieme da r1 a r2, vuol dire che almeno uno dei due ruoli deve essere giocato, e se c'è in r1 allora deve esserci anche in r2. questo può essere tradotto con un mandatory verso r2 perchè si presuppone con questi vincoli che la popolazione di r2 sia tutta.
- Se abbiamo l'inclusione su una coppia, non serve l'inclusione sui singoli ruoli, ma non vale il viceversa perchè l'inclusione potrebbe essere su coppie spaiate.
- Se ho l'esclusione su un singolo ruolo della coppia, allora questo implica che tutta la coppia ha l'esclusione e non vale il viceversa.

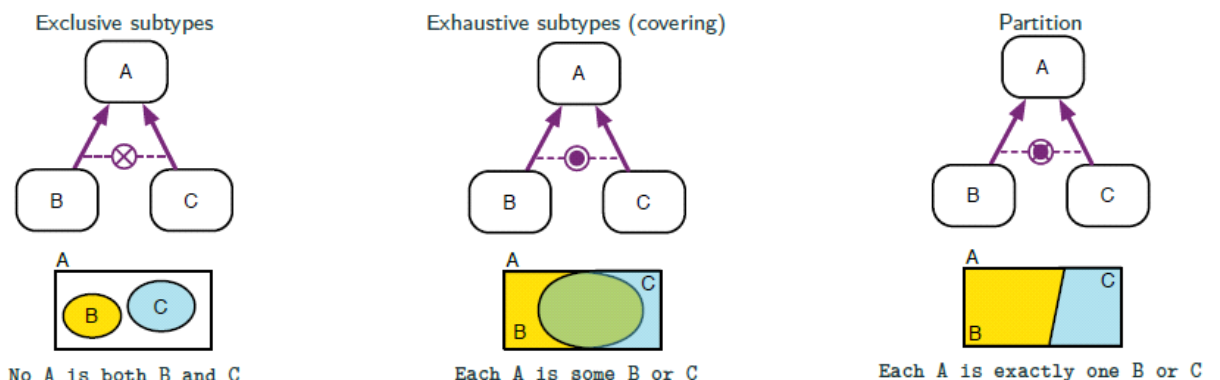
SOTTOTIPO

l'entità di sottotipo presuppone che le due entità siano diverse (due insiemi distinti) e che la popolazione del sottotipo sia inclusa in quella del tipo generico. La popolazione deve essere diversa da quella di A e inclusa, il che vuol dire che è un sottoinsieme proprio.

Quando uso B do delle restrizioni perchè ci sono dei ruoli in A che non vengono giocati in B e ne consegue la limitazione.

Dovremmo usarlo per dichiarare ruoli giocati dal sottotipo specifici e non dal sopratipo. Può servire per generalizzare o individuare delle gerarchie.

I sottotipi possono essere dichiarati in 3 modi specifici quando coinvolgono più sottotipi di una stessa entità:



Nessun tipo può essere sottotipo di se stesso e non possono esserci cicli nel determinare il sottotipo, deve essere un grafo aciclico, una struttura gerarchica vera e propria.

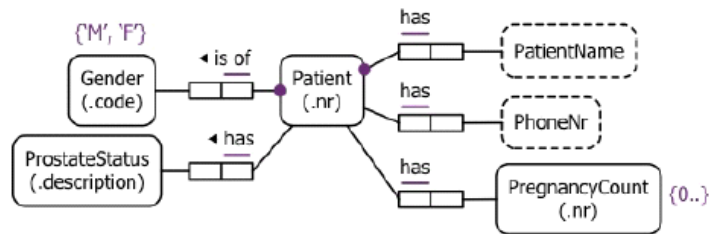
I due modi sono per generalizzazione, quando trovo elementi comuni, mentre la specializzazione

serve ad aggiungere dettagli.

A differenza dei tipi primitivi non vengono fatte assunzioni sul vincolo mandatory sui ruoli: quando si hanno diversi ruoli giocati da A, implicitamente come se ci fosse un mandatory disgiuntivo, mentre per i sottotipi si deve esplicitare questo tipo di vincolo.

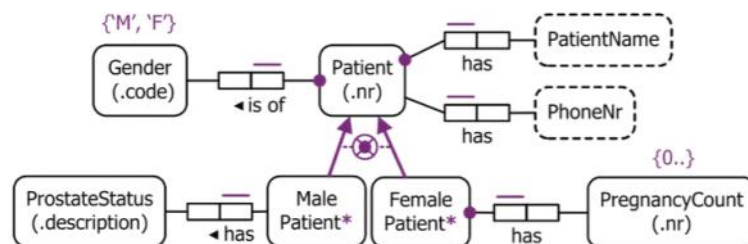
Ogni sottotipo deve essere definito nei termini di qualche ruolo giocato dal sopratipo, cioè posso derivare la popolazione del sottotipo da qualche ruolo giocato dal sopratipo. Ogni sottotipo è derivabile da una relazione giocata dagli elementi del sopratipo.

Es:



| PatientNr | Name | Gender | Phone | Prostate status | Pregnancies |
|-----------|-----------|--------|---------|--------------------|-------------|
| 101 | Adams A | M | 2052061 | OK | — |
| 102 | Blossom F | F | 3652999 | — | 5 |
| 103 | Jones E | F | ? | — | 0 |
| 104 | King P | M | ? | benign enlargement | — |
| 105 | Smith J | M | 2057654 | ? | — |

Analizzando la tabella possiamo dire che lo stato della prostata non si può avere per i pazienti di genere femminile, e per quelli di genere maschile non è un'informazione presente per tutti i pazienti. Per le donne il numero di gravidanze è sempre noto. Tutte le informazioni su nome, genere e numero di telefono è presente per tutti i pazienti, mentre prostata e gravidanze sono valori che dipendono dal sesso del paziente e nello schema sopra questa info non è modellata perchè sembrano semplicemente opzionali, ma in realtà sono non applicabili in base al sesso. Questa realtà deve essere modellata nel modo corretto, e questo schema(sopra) non lo è. Servono i sottotipi perchè ci sono info legate ai sottotipi e si vanno a legare i ruoli che corrispondono a queste parti di dati. Adesso possiamo anche dire che, per il sottotipo adatto, è obbligatorio il numero di gravidanze. I due sottotipi sono derivabili dalla relazione con Gender.



*Each MalePatient is a Patient who is of Gender 'M'.

*Each FemalePatient is a Patient who is of Gender 'F'.

Il simbolo grafico nella relazione di sottotipo è implicato perchè sono partizioni disgiunte di patient. Con l'asterisco poi andiamo a specificare da quale ruolo è derivato il sottotipo.

Asserire un sottotipo vuol dire che il sottotipo non è dato da un ruolo, ma è il ruolo giocato che dipende dal sottotipo, quindi il sottotipo non ha una definizione esplicita. Il fatto che io dichiari lo stato della prostata implica il sesso maschile, quindi il Gender sarebbe derivato. È simile all'OO perchè se un oggetto ha una certa proprietà vuol dire che appartiene a una certa sottoclasse.

Potremmo avere elementi semiderivati. Ad esempio se una persona gestisce una azienda, la sua denominazione è manager, quindi questo è un sottotipo completamente derivato. Potremmo specificare che però appartiene alla classe Manager se gestisce una azienda oppure se so a priori che è un manager anche se al momento non stanno gestendo una azienda. Per definizione classica non dovrebbero essere considerati manager, ma con la semiderivazione si può fare.

SPECIALIZZAZIONE

Given an object type. . .

1. Specify all mandatory role constraints.
2. For each optional role: **if**
 - it is recorded only for a known subtype **and** cioè se è propria
 - there is a subtype definition stronger than a set-comparison constraint **or** another role is recorded only for that subtype**then**
 - 2.1 Introduce the subtype.
 - 2.2 Attach its specific roles.
 - 2.3 Declare the subtype derived (*), asserted, or semiderived (+).
 - 2.4 If * or +, add a derivation rule.
 - 2.5 Goto step (1) considering the subtype.

GENERALIZZAZIONE

Da sottotipi andiamo a trovar eil sovratipo/i che ci permettono di categorizzarli. Servono perchè accomunano delle proprietà o perchè le sovraclassi introdotte possono servire in maniera più specifica.

Given two completely separated object types A and B. . .

If

- A and B overlap, or can be compared **and** we want to model this
- **or** A and B are mutually exclusive **and** common information is listed for both **and** we want to list A and B together for this information

then

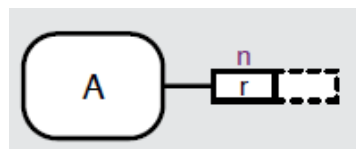
1. Introduce their supertype $A \cup B$ with its own identification scheme.
2. Add classification predicates on the supertype, to identify A and B.
3. Attach common roles to the supertype.
4. **If** A (or B) plays specific roles **then** define A (B) as a subtype and attach such roles.

PASSO 7:

Aggiunge vincoli di frequenza, ossia il numero di volte che una entità deve o può partecipare in un ruolo, e fare le verifiche finali. È una sorta di generalizzazione dei vincoli di unicità.

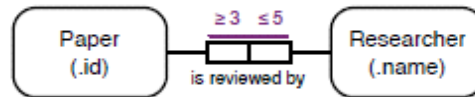
INTERNAL FREQUENCY CONSTRAINTS

Se una istanza partecipa ad un certo ruolo, è possibile definire che debba farlo n volte. Non implica però che l'istanza debba esserci obbligatoriamente, ma solo che se partecipa al ruolo debba farlo per n volte.

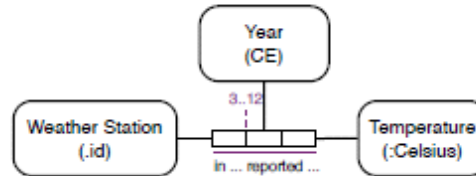


Potrebbero esserci anche Frequency Ranges. Ossia si impone un limite maggiore o minore di partecipazione ai ruoli.

Ad esempio in questo caso diciamo che un Paper deve essere recensito almeno 3 volte da 3 diversi Ricercatori, mentre un Recensore può recensire un massimo di 5 paper diversi.



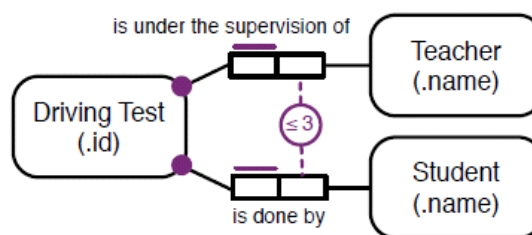
Ci sono anche i vincoli di frequenza composti che definiscono i limiti di quante volte una coppia può comparire.



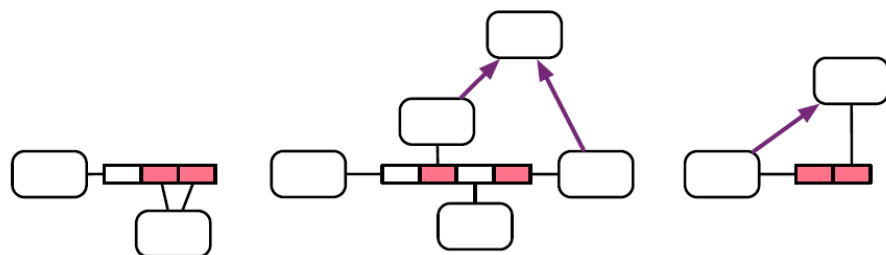
EXTERNAL FREQUENCY CONSTRAINTS

Si definisce ad esempio il numero di volte in cui una stessa coppia di ruoli può partecipare a una relazione con l'istanza.

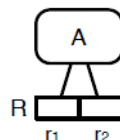
Nell'esempio vediamo come venga imposto il limite di un massimo di 3 coppie Insegnante-studente per il test di guida, quindi il test può essere ripetuto dallo stesso insegnante con lo stesso studente massimo 3 volte.



Si parla di Ring Constraint quando ci sono due ruoli che compaiono nelle stesse relazioni giocate dalla stessa classe o da loro sottoclassi.



- Prototypical setting:



- R is a relation, that could enjoy different properties:

- **Reflexivity:** $\forall x. xRx.$
- **Symmetry:** $\forall x, y. xRy \rightarrow yRx.$
- **Transitivity:** $\forall x, y, z. xRy \wedge yRz \rightarrow xRz.$

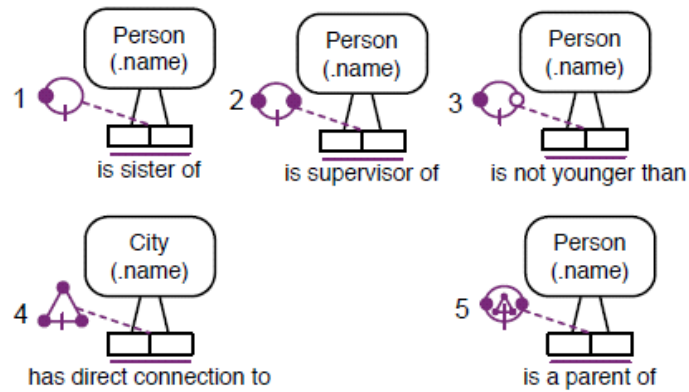
where all variables x, y, z range over $pop(r_1) \cup pop(r_2).$

- These properties are “positive”: they state how to derive *additional* information.

Queste hanno delle proprietà generative, che però se neghiamo diventano dei vincoli e vengono annotati.

Abbiamo:

- 1• **Irreflexivity:** $\forall x. \neg xRx$. non è mai vero che x è in relazione con se stesso
- 2• **Asymmetry:** $\forall x, y. xRy \rightarrow \neg yRx$. se x è in rel con y, non è vero che y è in rel con x
 - Obviously stronger than irreflexivity.
- 3• **Antisymmetry:** $\forall x, y. x \neq y \wedge xRy \rightarrow \neg yRx$. come asimmetria, ma x e y devono essere anche diversi
 - Is asymmetry without irreflexivity.
- 4• **Intransitivity:** $\forall x, y, z. xRy \wedge yRz \rightarrow \neg xRz$.
- 5• Can be also combined.



Verifiche finali:

1. Fatti Elementari
2. Consistenza
3. Vincoli esterni corretti
4. No ridondanze
5. Completezza
6. Tutti i dati derivabili

Relational Mapping

martedì 26 ottobre 2021 11:18

I passi sono progettazione dello schema concettuale, concretazione dello schema con le scelte fatte in esso, il mapping dello schema concettuale in uno logico, la manipolazione dello schema logico ottenuto per adeguarlo al scelte o vincoli e infine generazione in uno schema fisico di un vero dbms. ORM avendo la peculiarità di voler rappresentare il minimo indispensabile, permette il mapping verso altri strumenti, non essenzialmente tagliate su quello che è l'essenziale da esprimere. UML è nato per la modellazione concettuale e non per visualizzare classi. Quasi nessuno lo usa se non per documentazione, ma non nello sviluppo. ORM per la modellazione di sistemi informativi.

Il modello relazionale segue l'uml, nasce per fare modellazione concettuale per la progettazione OO, mentre il relazionale nasce per fare progettazione per i modelli dbms relazionali. Rappresenta il nostro database in termini di relazioni, dove ogni relazione è una tavola con un nome. Le colonne delle relazioni sono intese concettualmente come degli attributi di solito associati a quello che è il concetto rappresentato dalla relazione in questione. Il dominio dei dati spesso non vi è una specifica corrispondenza tra il tipo di quello che vorremmo esprimere e quello che abbiamo a disposizione. Il modello rel permette di mettere dei vincoli, che sono le chiavi, le Foreign key, quando un elemento è chiave di un'altra tavola, o se è opzionale.

Database schema constituted by a set of relation schemas containing:

- name of the relation (table name);
- (named) set of attributes (table columns), each ranging over some data domain (unnamed equivalent version also exists).
- Extensional information represented as a set of unnamed tuples (records) over such relations, where each attribute is filled in with a value belonging to the corresponding data domain.

Lo standard sql supporta i tipi definiti dagli utenti.

Durante l'analisi si astrae dal dominio dei dati.

Il modello relazionale viene rappresentato con la notazione orizzontale che usa il nome della tavola e tra parentesi il nome delle varie colonne:

Employee(empNr, empName, deptCode, gender, salary, tax)

E quella verticale dove poi in verticale il nome delle colonne e si ha un box con il nome in cima alla tavola:

| Employee |
|----------|
| empNr |
| empName |
| deptCode |
| gender |
| salary |
| tax |

I vincoli di unicità sono un buon candidato per quelle che sono definite chiavi. E' un insieme minimale di attributi constraint. Le chiavi vengono sottolineate nella orizzontale, nella verticale vengono scritte di lato(PK primary key, Un le altre). Se ci sono più di una chiave nella orizzontale si evidenzia la primaria in qualche modo.

È possibile indicare attributi obbligatori o opzionali. Quelli opzionali sono quelli che possono essere null, mentre gli obbligatori sempre con un valore, mai null. Nella verticale sono in grassetto quelli

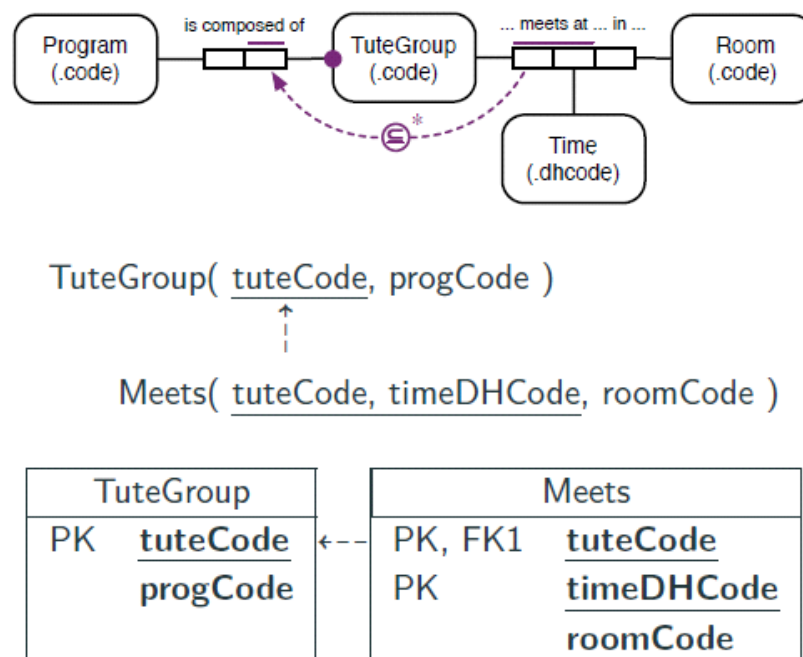
obbligatori, nella orizzontale si mettono tra le quadre le opzionali. Una chiave primaria è sempre obbligatoria perchè non può essere null e tutte le sue colonne lo devono essere.

In ORM se abbiamo un ruolo obbligatorio vuol dire che qualunque elemento che giochi un ruolo in un'altra relazione, quello stesso elemento deve giocare un ruolo nelle relazioni in cui quel ruolo è obbligatorio. In altri termini: se abbiamo che r è mandatory per un certo object type, se lo stesso entity type gioca un altro ruolo allora la popolazione di questo secondo ruolo è presente in r.

La Foreign key permette di dire che se abbiamo un ruolo di elementi che in questo ruolo sono opzionali. È rappresentata da una freccia tratteggiata che va verso la primary key della tavola a cui si vuole collegare.

Es:

Abbiamo che TuteGroup ha un codice e ha un ruolo mandatory con Program, mentre è opzionale con Room e Time. Cioè il gruppo ha un solo programma e deve averlo per forza, ma non è detto che ci siano effettivamente degli incontri. Se si incontrano però deve avere per forza un programma allegato. Questo schema mappa le relazioni in ORM in tavole. Si potrebbe avere una tavola TuteGroup e una che si chiama Meets e mappano le due relazioni, intuitivamente. Nella prima tavola definiamo il vincolo di unicità e obbligatorietà definendo tuteCode come chiave. Il codice del programma in esso è obbligatorio. Per Meets anche abbiamo che sono tutte obbligatorie e tuteCode e time sono in coppia la chiave perchè c'è il vincolo in ORM che lo specifica. In modo implicito c'è un vincolo di inclusione dalla relazione meets at alla relazione is composed of. Il che vuol dire che se metto un'indicazione di un gruppo di tutoraggio in Meets, quello stesso gruppo deve essere necessariamente presente in TuteGroup, e questo lo si indica con la foreign key.



In ORM ci sono molto vincoli, bisogna capire come rappresentarli nel relazionale. Non ci sono modelli che indicano le frequenze o il dominio quindi semplicemente si annotano sotto e si cerca di rispettarli. I vincoli possono essere rappresentati come in sql. Vengono eventualmente definiti anche tramite controlli così :

```

check(not exists
(select resCode from Reviewed
group by resCode having count(*) > 5))

```

Le regole di derivazione possono essere definite nel relazionale tramite Viste, colonne generate,

colonne trigger(?), procedure ripetute periodicamente per fare gli update delle tavole.

Rmap

Questo algoritmo tenta di bilanciare due aspetti: l'efficienza e evitare ridondanza. Queste perché il relazionale non ha il principio guida che ogni relazione è elementare, infatti le tavole non sono elementari anzi spesso si mettono elementi anche futili. Questo perché meno tavole ho, più efficiente è il dbms perché non deve fare troppi join. Se metto tutto in una tavola però ho delle ridondanze, quindi avere meno tavole vuol dire questo, e introdurre tanti null. ORM aveva l'obiettivo di avere relazione elementari perché ogni relazione è un fatto e i fatti non hanno null perché se esistevano erano completamente istanziati. ORM nelle sue relazioni definiva un predicati specifici diversi da tutti gli altri.

Una stessa riga può rappresentare più predicati mentre in ORM no. Una riga quando la definiamo non possiamo dire che è un fatto.

Per passare da orm a relazionare serve tener conto che quest'ultimo non serve a definire i fatti.

Il modo con cui Rmap cerca di evitare la ridondanza è fare in modo che ogni fatto sia mappato in al più una tavola.

Se lo stesso tavolo è su più tavole c'è il problema di mantenere la coerenza fra più tavole.

Segue die semplici punti:

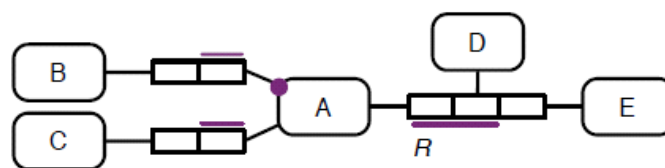
1. Se un fact type ha un vincolo di unicità interno nella relazione composto, quindi non singolo e cioè o un binario o un ternario molti a molti con almeno due ruoli coinvolti, questa viene mappata in una tavola destinata a solo quella relazione e il vincolo di unicità diventa la PK
2. Se invece il fact type ha un ruolo funzionale (vincolo su uno solo dei due ruoli) allora vado a raggruppare tutti quelli che hanno la stessa object type, quindi raggruppo tutte le relazioni funzionali che hanno lo stesso object type.

Tutte le relazioni che non sono con vincolo di unicità singolo, vanno tutte in una tavola singola dedicata mentre le relazioni funzionali si riducono in una sola tavola e le si associa all'entity type in una tavola.

Nel caso 1) si dà alla tabella il nome della relazione; nel caso 2) è il nome stesso dell'entity type a cui sono associate. I nomi dei ruoli si usano come nomi delle colonne. Nel caso 2) la PK della tavola è il reference mode dell'entità. Se ci sono altre relazioni mandatory tra quelle funzionali vengono definite come chiavi alternative.

Es.

Hp: a, b, c, d, e represent the set of attributes representing the preferred identification scheme for the corresponding object type.



Case 1

Case 2

- Applies to binary fact types with simple UCs all over roles played by the same object type.
- Mandatory dots determine mandatory columns in the grouped table.
- Applies to binary fact types with spanning UCs and to n-ary fact types with $n > 2$ (why?).
- FKs must be added for those roles attached to object types that play mandatory roles elsewhere (\subseteq).



Caso 1) una chiave composta, quando il vincolo di unicità è più di uno ed è presente nella terna A-D-E. Il caso 2) dice che devo raggruppare quelli funzionali. Dove è funzionale diamo alla tavola il nome

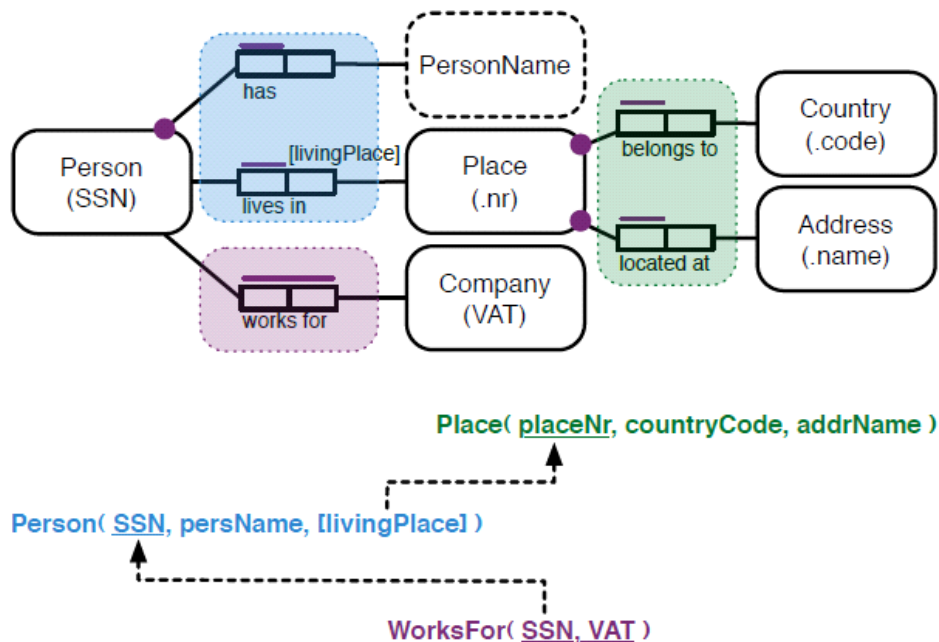
della classe che ha le relazioni funzionali come si vede sopra. Nell'altro caso, poichè sono composte, usiamo il nome della relazione, che è R.

Nel primo caso i reference mode sono gli identificatori delle colonne. La chiave primaria di A è il reference mode dell'entità A, che è a. Raggruppando B e C ci sono delle coppie in cui in alcuni casi ci sono valori nella rel B-A che non ci sono in C-A perchè la relazione C-A è opzionale. B è mandatory mentre C è optional.

In R si ha come chiave il reference mode di A e D e poi abbiamo l'elemento e come colonna. Non è optional perchè tutte le relazioni nella tripla devono essere messe obbligatoriamente.

l'opzionalità in ORM dipende da quella relazione stessa perchè sono indipendenti, mentre nello schema relazionale se si aggiunge una nuova colonna, questa va a modificare le proprietà delle altre colonne perchè potrebbero dipendere da essa. Questo perchè potrebbe portarmi a delle righe in cui le altre righe sono vuote se ha una variazione, quindi magari un attributo che era ritenuto obbligatorio fino a quel momento, si definisce opzionale con l'aggiunta di una nuova colonna che va a modificarne il significato. In ORM invece si ha una struttura incrementale, perchè lo posso modificare senza andare ad impattare su quello che esiste. Perciò è importante avere una rimappatura perchè con l'ORM è più semplice fare modifiche successive.

ES.



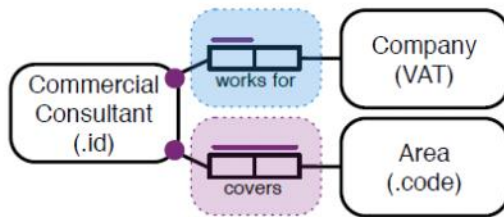
Dato che livingPlace è il nome del ruolo, si usa questo. Inoltre è opzionale perchè la persona può avere un nome e non un posto dove vive. Inoltre abbiamo dei vincoli impliciti: quello che compare in WorksFor, deve necessariamente essere presente in Person. Posso avere degli SSN in Person che non sono in WorksFor, ma non viceversa.

Stessa cosa per livingPlace, cioè se il living place è segnato, deve essere presente in Place però ci possono essere dei Place che non sono segnati in livingPlace.

Es:

Abbiamo una molti a molti composta e poi una funzionale. Sulle due tavole ci sono dei vincoli, nello specifico due mandatory. Inoltre abbiamo delle foreign key che dipendono dagli elementi comuni. Non c'è modo di rappresentare il fatto che commercialConsultant è una foreign key ma non può essere definita come tale perchè è parte della primary key.

The two mandatory roles form an equality constraint.



CommercialConsultant(commConsId, VAT)

↕

Covers(commConsId, AreaCode)

The equality constraint becomes a pair of subset constraints in the relational schema, forming a **referential cycle**.

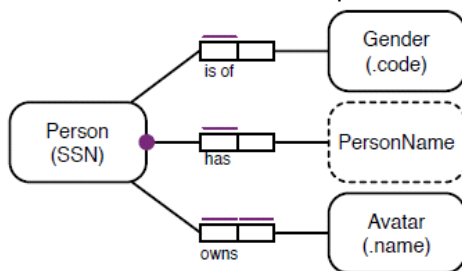
- The subset constraint from Covers to CommercialConsultant is a FK constraint.
- The subset constraint from CommercialConsultant to Covers is **not** a FK constraint, because it targets only part of the PK in Covers.

It can be enforced using assertions, stored procedures, triggers, ...

MAPPING 1:1 ASSOCIATION

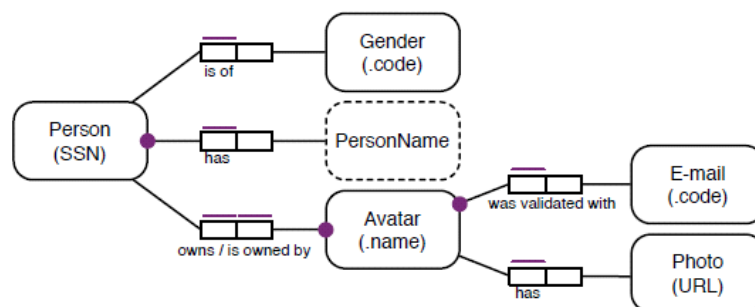
Dipende da come è contestualizzata:

1. È una relazione 1:1 e uno dei due lati già raggruppa, mentre l'altro no. L'algoritmo richiede efficienza, quindi non si deve avere ripetizioni. Si va a creare la tavola sull'entity type che già raggruppa e quindi le informazioni su di esso non saranno ripetute nello schema relazionale. Si inserisce l'informazione della relazione 1:1 impostandola come chiave in modo che venga evidenziato il fatto che non può essere ripetuta.



Person(SSN, [gender], persName, [avName])

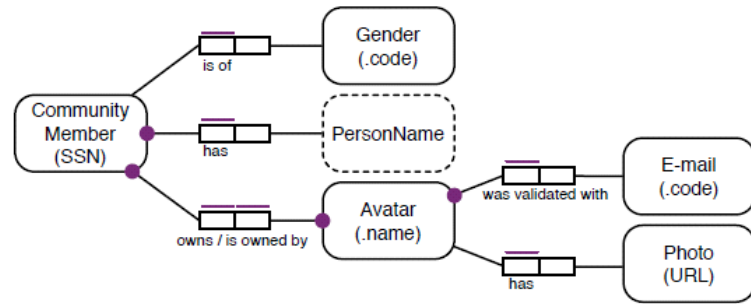
2. Abbiamo la 1:1 e ciascuna delle estremità della relazione sono complesse e già raggruppano e rappresentano una tavola. Se uno dei due ha il mandatory, si raggruppa verso quel lato. Inoltre si impone un vincolo di foreign key verso la tavola non mandatory per indicare che gli elementi devono essere sempre presenti nella sua popolazione.



Person(SSN, [gender], persName)

Avatar(avName, SSN, eMailCode, [photoURL])

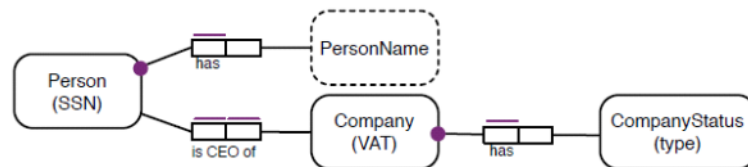
3. Se entrambi i ruoli sono mandatory nella relazione si sceglie uno dei due e si impone un vincolo di foreign key doppio perchè ci sarebbero in entrambe le tavole la stessa informazione, che deve essere uguale.



(2) CommunityMember(SSN, avName, [gender], persName)

Avatar(avName, eMailCode, [photoURL])

4. Se non c'è un mandatory ma sono già collegate con altre, vuol dire che ci sono dei null. Si raggruppa dal lato che è più probabile che sia coinvolta nella relazione. Nell'esempio è più probabile che una azienda abbia un CEO piuttosto che una persona sia un CEO. Se considero che le probabilità sono simili potrei definire direttamente una nuova tavola per ridurre i null e poi mettere le foreign key verso le altre.



Hp: it is more likely for a Company to have a CEO that for a Person to be CEO of a Company.

Hp: it is likely that both solutions (grouping on Person or on Company) yield many null values.

Person(SSN, persName)
 ↑
 Company(VAT, [SSN], ComStatus)

Person(SSN, persName)
 ↑
 IsCEOof(SSN, VAT)
 ↓
 Company(VAT, ComStatus)

Quando abbiamo il coinvolgimento di un vincolo esterno, tipo che abbiamo due vincoli funzionali ma sono tra loro legati da un vincolo di unicità.

Abbiamo diversi casi:

- cioè usato come reference type
- External UC is the preferred identification scheme for an object type attached to other functional roles.
 - External UC is the preferred identification scheme for an object type attached to other nonfunctional roles.
 - External UC is not the preferred identification scheme for an object type attached to other functional roles.
 - External UC is not the preferred identification scheme for an object type, and it is paired with an n:m fact type.

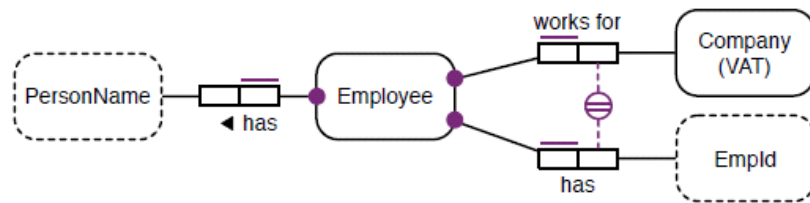
Caso 1:

Abbiamo employee che gioca un ruolo funzionale con PersonalName, e poi ha worksFor e has Empld. Abbiamo un vincolo esterno di unicità che è anche reference mode quindi employee si identifica con la coppia Company-Empld.

Raggruppa gli object type con dei surrogati, senza considerare i predicati coinvolti nel vincolo di unicità esterno. Poi espande la PK usando l'object type coinvolti nel vincolo esterno.

Quindi si definisce una tavola e si va a lavorare definendo il resto delle relazioni funzionali coinvolte

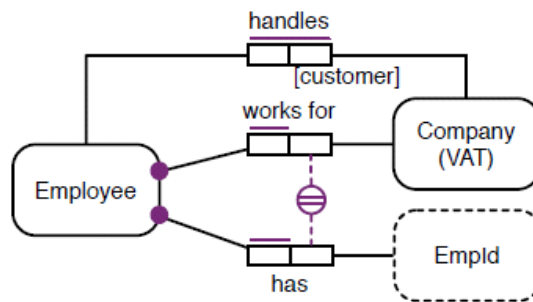
con l'object type e solo in seguito si espande con gli oggetti coinvolti nel vincolo esterno.



| | | |
|--------------------------|--------|---|
| Employee(<u>e</u> , n) | -----> | Employee(<u>empld</u> , VAT, empName) |
|--------------------------|--------|---|

Caso 2:

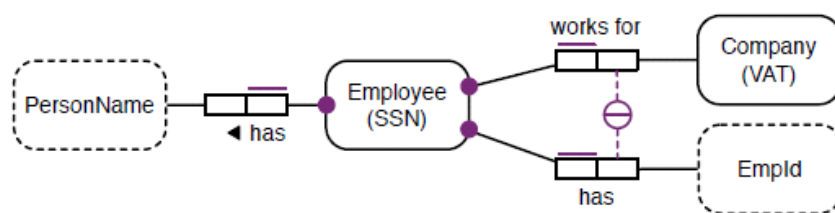
È una PK il vincolo di unicità esterno, ma employee è coinvolta in una relazione non funzionale. Mappa l'associazione n:m in una tavola separata dove si usa un surrogato per la PK dell'object type, ignorando il vincolo esterno. Si espande utilizzando poi la chiave avuta dal vincolo esterno.



| | | |
|-------------------------|------|--|
| Handles(<u>e</u> , c) | ---> | Handles(<u>empld</u> , VAT, customerVAT) |
|-------------------------|------|--|

Caso 3:

Caso in cui il vincolo esterno di unicità non è il preference mode, e quindi non è esso ad indentificare l'object type. Si segue il mapping standard, perchè l'obj type ha il suo reference mode. La coppia con vincolo di unicità viene inserita nella tavola come normali "colonne" con indicazione di chiave candidata, ma non chiave primaria.



Employee(SSN, VAT, empld, persName)

Caso 4:

Se lo stesso elemento è coinvolto anche in una relazione molti a molti si segue il mapping standard ignorando il vincolo esterno e poi lo si annota in maniera ufficiale in documentazione.

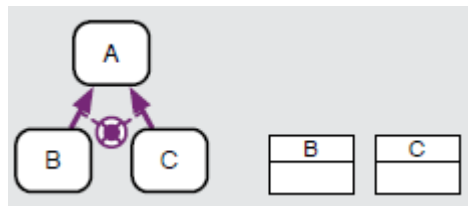


Se abbiamo le oggettificazioni andiamo a definire una tavola per la relazione e si usa un surrogato per definirne la primary key e poi si espande con i reference mode delle object type coinvolte .infine si incorporano i vincoli più specifici.

Gli oggetti indipendenti sono quelli con il !, ossia non giocano mai dei ruoli mandatory ma ho un vincolo legato all'esistenza degli elementi. Essi vengono trattati con tavole a se stanti e poi dove è coinvolta viene messo un vincolo di FK verso la tavola definita.

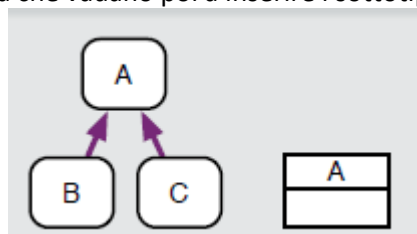
Traduzione vincolo di sottotipo

Se è una partizione, ossia che le popolazioni delle due sottoentità non hanno elementi in comune, si va a rimuovere il supertype e si hanno solo due tavole separate che traducono i sottotipi andando a replicare le informazioni del supertipo in essi.

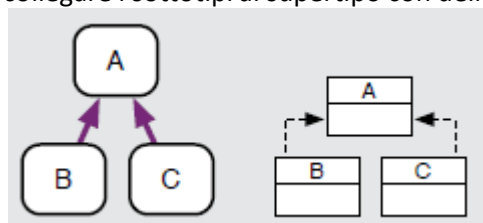


Non è detto che questa sia la soluzione migliore perchè molto probabilmente i supertipi sono coinvolti in relazioni e potrebbe creare problemi durante la fase di interrogazione del db. Bisogna quindi valutare volta per volta in base alle esigenze: se le sottoclassi sono quelle più interrogate allora conviene usare solo queste come tavole; se però si interrogano le taole per avere info di elementi comuni ai due sottotipi, allora converrebbe una sola comune, quindi la sovraclasse e nella quale dovrò predisporre l'info che mi dice di che sottotipo sono.

Questo secondo metodo è definito assorbimento, ossia i sottotipi sono assorbiti dalla superclasse e si aggiungono delle indicazioni nella tavola che vadano poi a inserire i sottotipi.



La separazione invece consiste nel separare gli elementi nei sottotipi e nel supertipo andando a creare tre tavole distinte e andando poi a collegare i sottotipi al supertipo con delle foreign key.



VEDI SLIDE DA 35 A 41

Se dovessimo avere diversi tipi di chiave nella gerarchia si va ad utilizzare come chiave della tavola il ref mode indicato nell'entità, ma quello del suo sovratipo deve essere indicato e come chiave candidata, per evidenziarne l'unicità.

Business Process Management

lunedì 8 novembre 2021 18:49

L'obiettivo del Business Process sta nel desiderio di modellare dei processi che coinvolgono aziende, produzioni, attività, quindi un concetto di coordinazione di attività all'interno di uno specifico contesto per raggiungere degli obiettivi di interesse per chi lo descrive.

Vincoli statici: che definiscono il contesto di un sistema informativo, quindi descrizione dei dati e dei vincoli da rispettare da parte dei dati

Vincoli dinamici

Un sistema informativo che ci permette di prendere decisioni non è statico ma subisce eventi causati da azioni/operazioni effettuate. Queste devono rispettare alcuni vincoli per raggiungere un certo obiettivo. Anche ad esempio scegliere la sequenza di modifiche da applicare al mondo per raggiungere lo stato che si vuole è un vincolo imposto. Definire come devono essere organizzate delle attività è stabilire un vincolo.

I casi d'uso sono un esempio di vincolo dinamico perché spieghiamo come si deve utilizzare un certo sw.

Se in vincoli statici definiscono uno stato fermo nel tempo e impongono dei limiti su di esso, quelli dinamici sono in termini evolutivi di passaggio da uno stato all'altro.

Il Business Process Management include i concetti, metodi e tecniche per supportare la progettazione, amministrazione, configurazione, analisi e attuazione dei processi aziendali. L'idea di processo non è recente. Quella che è l'organizzazione del lavoro a catena di montaggio è la base di questo studio. Alcuni esempi sono la fabbrica di spilli o il processo per ottenere un prestito, che seguivano una serie di passi ben precisi in sequenza per raggiungere l'obiettivo. La gestione delle aziende negli ultimi 30 anni è passata da non considerare i processi, al ritenerli l'elemento fondamentale. La gestione di una azienda è gestione di processi vera e propria. Ha determinato il passaggio da una visione funzionale dell'azienda ad una visione per processi: cioè da un organigramma con l'insieme di funzioni eseguibili dall'azienda per descriverla, si è passati poi, dagli anni 90, all'analisi dei processi sottostanti che utilizzano le funzioni degli organigrammi come mattoni della attività che sono poi organizzati da scegliere. Le funzioni sono riorganizzate a seconda dell'organizzazione dei processi che usano le funzioni. Confrontando con i processi di sviluppo sw se c'è un sw e voglio risistemarlo non chiedo cosa si vuole modificare ma si chiede come si utilizza il sistema e le difficoltà che si incontrano. In base a questa descrizione si va a vedere se le funzionalità offerte dal sw sono quelle che servono o meno e le dipendenze tra esse.

Un altro punto importante è la gestione del cambiamento come un'esigenza permanente per l'azienda. Prima solo in casi di grandi cambiamenti si andava ad agire per affrontarli il cambiamento negli ultimi anni invece si è visto come il cambiamento sia una esigenza continua per l'azienda come pratica costante di adattamento all'ambiente. Una azienda vive all'interno di un ambiente, composto da mercato, stakeholder, dipendenti, e solo avendo costantemente un modello posso affrontare i piccoli cambiamenti continui all'ambiente che comportano un adattamento ad essi.

La ristrutturazione dei processi ha il fine di migliorare le prestazioni dell'azienda. L'azienda quindi valuta costantemente la posizione nel mercato e cerca costantemente nuove tecniche per innovarli e acquisire un vantaggio competitivo con le aziende concorrenti, quindi una costante valutazione della sua posizione.

I cambiamenti possono essere:

- Cambiamenti del contesto **esterno** (clienti, fornitori, concorrenti, dinamiche di mercato, leggi, ...)
- Cambiamenti del contesto **interno** (efficienza, efficacia, controllo, coordinamento delle attività, ...)

Un sistema "aperto"

L'azienda è un sistema **complesso** (gran numero di componenti, strutture, attività, prodotti, attori, risorse, ...), **distribuito** (composto da sistemi eterogenei e geograficamente dispersi), **aleatorio** condizionato da frequenti e imprevedibili eventi perturbatori interni ed esterni).

Questa visione dell'azienda necessita una visione a processi per poterla affrontare al meglio. Il goal del BPM è la comprensione di come una organizzazione lavora. Capire le attività chiave in una organizzazione e come questi sono correlati con le altre attività. Capire come alcune attività sono innestate all'interno dell'organizzazione e del contesto tecnico. Potenzialmente questi obiettivi sono tutti per aumentare automazione e migliorare il busine processing.

Un modo per valutare le aziende nasce dalla Teoria delle Code matematica. L'approccio sistemico si appoggia a quello di vedere l'azienda come una scatola chiusa e considerare questa scatola chiusa in termini di quante richieste arrivano all'azienda, clienti, e quanti ne escono soddisfatti dopo aver ottenuto quello che desideravano, valutando magari dal punto di vista temporale (quindi quante richieste soddisfatte in un arco temporale determinato). Questo è stato fatto con le leggi di Little, che lavora su quelli che sono i numeri delle code che si vengono a creare.

Es:

Leggi di Little per la valutazione

Un'agenzia di assicurazioni tratta 120000 pratiche in un anno e in media ci sono 600 pratiche in agenzia in un qualsiasi momento. Qual è il tempo di ciclo se in un anno consideriamo 50 settimane lavorative?

- $\lambda = 120000/50 = 240$ pratiche per settimana, tasso di arrivo
- Da $N = \lambda * \Lambda$ abbiamo $\Lambda = 600/240 = 2.5$ settimane, tempo di ciclo
- Il gestore del processo può focalizzarsi su due parametri e trovare il terzo con le leggi di Little.
- Un sistema è stabile quando $\lambda \sim \lambda'$, per ridurre le code si deve ridurre il tempo di ciclo

il tasso di entrata e il tasso di uscita
sono molto simili

Questo è l'approccio sistemico e non dice nulla sull'organizzazione interna dell'azienda, ma dice solo di ridurre il tempo di ciclo, quindi non da una guida.

L'organizzazione funzionale è la visione tradizionale dell'organizzazione di una azienda. È importante per individuare attività e mansioni da attribuire, però le funzioni non ci dicono come sono organizzate tra di loro e come vengono utilizzate nel "processo produttivo" o meglio nella catena del valore.

La catena del valore di Porter

Il fine di ogni attività svolta dall'azienda è quello di creare dei prodotti che vengano incontro ai desideri dei clienti,

creino del valore per il cliente. Per avere dei clienti che escano soddisfatti, le varie attività all'interno dell'azienda aumentano il valore di esso andando a fornire dei servizi o andando a lavorare la materia grezza. ogni componente dell'azienda dà un contributo verso l'ottenimento di un valore aggiuntivo per il cliente che si è presentato. Ogni componente crea del valore per il cliente.

L'azienda è vista come un sistema di attività generatrici di valore.

Ci sono attività di supporto che supportano altre attività, attività che aggiungono valore o che

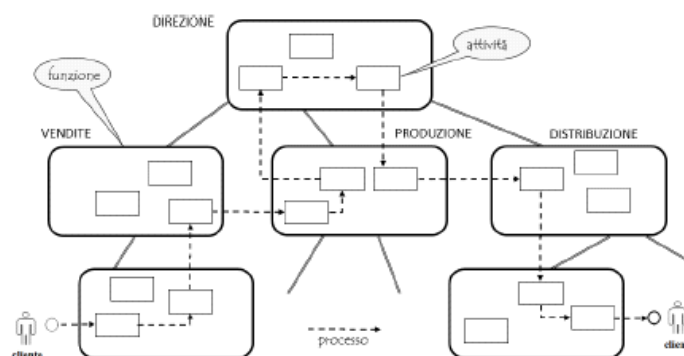
aggiungono valore all'azienda.

Le varie funzioni dell'organizzazione contribuiscono tutte alla catena del valore, quindi esistono perché danno un contributo all'accrescimento del valore dal cliente che si presenta, al cliente che esce soddisfatto. Questa catena ha però dei problemi: è difficile tramite questa semplice analisi individuare le responsabilità verso il cliente. Se si presenta uno specifico problema di un prodotto chi sono le persone coinvolte alla causa di esso?

Ha una visione ristretta, focalizzata al proprio interno, per cui le comunicazioni con le altre funzioni spesso non sono pienamente soddisfacenti e il coordinamento necessario fra le attività aziendali risulta carente.

È necessaria una visione più ampia dell'azienda, ecco che nascono quindi i processi

I processi possiamo vederli come giacere sull'organizzazione funzionale. L'organizzazione funzionale definisce le mansioni (funzioni) che possono essere svolte, il processo è un percorso all'interno delle varie funzioni che permette di creare quel valore aggiunto. È lo studio di come queste sono tra di loro collegate che mi permette di avere una visione più generale dell'azienda. Il processo "riduce" le attività a partire da un evento esterno generato da un cliente e può "attraversare" diverse funzioni aziendali per ritornare al cliente.



Un processo aziendale come un insieme di attività (controlli e azioni) tra loro interrelate per realizzare un risultato definito e misurabile, il prodotto o servizio che trasferisce valore al fruitore (al cliente) del prodotto o servizio stesso.

Possiamo identificare queste caratteristiche:

- Risponde ad un evento esterno eseguendo delle attività
- Deve raggiungere certi obiettivi
- Deve fornire un risultato
- Consuma delle risorse aziendali per essere eseguite
- Deve soddisfare i requisiti dei clienti
- È vincolato da regole interne ed esterne
- Ha generalmente un responsabile (process owner)

Il BPM, ossia Business Process Management, è un insieme di metodi, tecniche e strumenti per gestire nel modo migliore l'azienda.

È il risultato di un'evoluzione che ha raccolto risultati da vari filoni di ricerca, classificabili in tre categorie:

- Gestione aziendale
- Controllo della qualità
- Tecnologie ICT

Gli obiettivi generali del BPM sono:

- Ottimizzare i processi "interni" dell'azienda
- Facilitare l'allineamento dei processi gestionali e produttivi agli obiettivi strategici dell'azienda
- Migliorare la flessibilità dell'organizzazione aziendale in modo da rispondere "rapidamente" ai cambiamenti di scenario
- Facilitare il flusso del processo fra i vari partecipanti cercando di eliminare le attese

- Automatizzare il flusso di controllo e dei documenti al proprio interno
- Analizzare, modellare e misurare il processo (indicatori di efficienza)
- Utilizzare il modello per fare delle previsioni (analisi "what if")
- Migliorare l'ambiente di lavoro eliminando i passi ripetitivi mediante uso "intelligente" delle tecnologie
- Ripensare i processi avendo come obiettivo il miglioramento delle qualità e dell'efficienza

Storicamente:

Taylor (1911)

Scientific Management applied to work.

catena di montaggio. obiettivo: individuare come suddividere il lavoro per poter trovare il personale più adatto per ogni mansione in base alle competenze personali

Hammer & Champy, Davenport (1990s)

Business Process re-engineering and innovation.

introdotta una visione nuova dei processi come elementi fondamentali per affrontare le modifiche. allora erano più di innovazione

Smith & Fingar (2000s)

BPM - the third wave.

negli ultimi 20 anni la visione è affrontare costantemente i cambiamenti con l'analisi per confrontare e adattarsi ai continui cambiamenti del mercato o dell'ambiente

Now

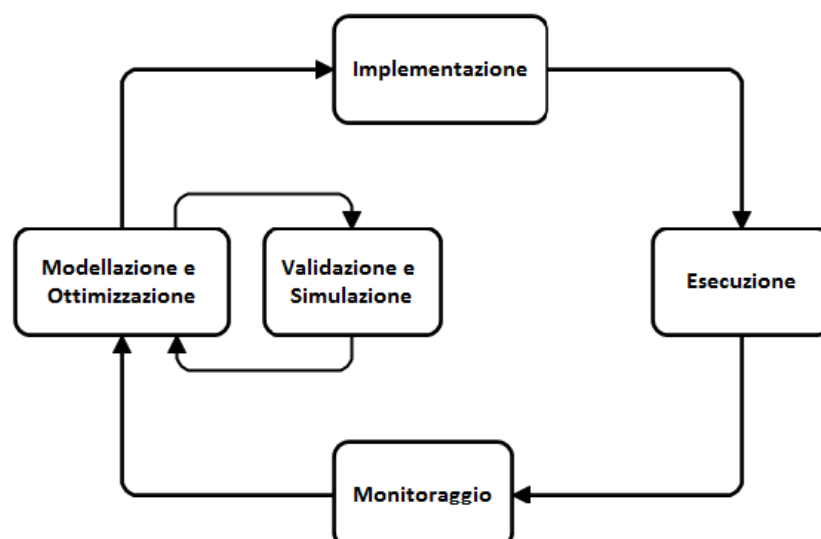
Business Process lifecycle.

Dapprima si ha quindi un'organizzazione del lavoro in base alle competenze della forza lavoro. La visione della reingegnerizzazione, ossia aver dei processi per poterli ristrutturare e focalizzarsi sui cambiamenti. In questo caso compare la prima modellazione di processi come studio di come le attività sono organizzate. Le attività quindi sono definite con un ordine di esecuzione di esse e come sono correlate tra di loro le attività.

Si arriva infine a Weske : *a Business Process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities, all of them, jointly realize a business goal. Each BP is enacted by a single organization, but it may interact with BPs performed by other organizations.*

L'ultima ristrutturazione è stata quella di utilizzare continuamente la modellazione, andando ad introdurre il ciclo di vita dei Business process. Ha come obiettivo la valutazione e monitoraggio continuo di un processo in modo da fornire miglioramenti incrementali, e non a cambiamenti radicali. Sempre secondo Weske *il BPM includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of BPs.*

Il ciclo di vita dei processi:



Ci sono vari tipi di BP.

Organizational vs Operational Organizational BPs are high-level BPs describing inputs, outputs, expected results, and dependencies on other organizational BPs in a coarse-grained way. One organizational BP corresponds to many operational BPs, which make activities and their relationships explicit.

Intra vs Inter-organizational Intra-organizational BPs are internal to the company, and describe relevant activities that are internally **orchestrated** to produce a good or deliver a service. Inter-organizational BPs describe interactions among multiple processes running in different cooperating organizations, in terms of **choreographies**.

Degree of automation and presence of human activities.

Degree of repetition which is important to check whether the explicit modeling of BPs really deserves the effort.

Degree of structuring ranging from predictable, repetitive BPs (**production workflow**) to unpredictable, adaptive BPs with **ad-hoc** activities and creativity of the experts (**knowledge-intensive**).

Un processo può essere visto in maniera astratta, dove le attività sono descritte in maniera astratta, ma poi hanno delle istanze, le istanze gestiscono i casi concreti, intesi come esecuzione vera e propria. È possibile passare da istanze di processi a una generalizzazione, process mining, per poter anziché descrivere il processo da interviste o altro, facendo il mining dalle istanze che occorrono e dai log sulle descrizioni per poterle raffinare, una sorta di automazione del processo. Lavoreremo con la visione astratta.

Altri modelli oltre al modello a processi esistono e spesso lo affiancano. Ad esempio il modello strategico dell' Object Management Group definisce anche il business motivation model. Esso permette di definire formalmente quel è il modello strategico per raggiungere l'obiettivo e verificare se il processo è conforme.

Il modello organizzativo è un altro. Il modello funzionale, che definisce ogni funzione e come questa è definita rispetto ai dati di cui ha bisogno o che ci permette di ottenere. Mi serve a capire se colloco una certa funzione in un certo reparto se è quello giusto per le mansioni da svolgere.

I modelli fino ad ora considerati, Strategico, Organizzativo, e Funzionale, sono essenzialmente dei modelli statici che rappresentano la struttura dell'azienda, delle sue componenti e l'architettura delle sue funzioni, ma non è possibile risalire da questi modelli a come si comporta effettivamente l'azienda.

Serve specificare come avviene l'esecuzione dei processi e delle attività in azienda prendendo in considerazione un Modello di Attività; un tale modello descrive l'insieme di esecuzioni possibili, o istanze, dell'attività in esame. In un certo istante potrebbero esistere (sono attive) più istanze di un'attività.

Un'**istanza di attività** si evolve nel tempo e può passare attraverso vari stati, ad esempio può essere attiva o essere in attesa di risorse, e così via; questa dinamica può essere descritta da un diagramma stati/transizioni in cui nodi sono gli stati e gli archi le transizioni che permettono di passare da uno stato all'altro.

Le **transizioni** sono determinate da eventi (che identificano le transizioni stesse) e per ogni stato l'insieme delle transizioni uscenti determina i possibili stati che possono essere raggiunti a partire da quello in esame come conseguenza del verificarsi degli eventi corrispondenti.

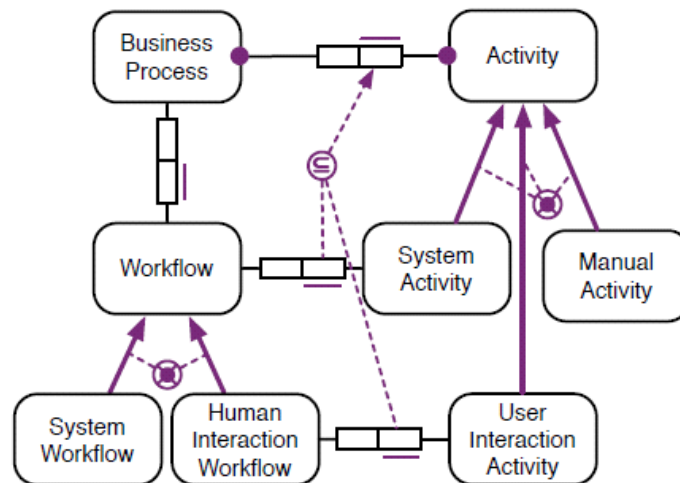
Un **evento** è il verificarsi di un fatto in un certo istante di tempo che provoca una transizione cioè il passaggio da uno stato a un altro del sistema.

Un modello dei processi descrive l'insieme di esecuzioni possibili, o istanze, del processo in esame.

Il processo ha dei casi, che sono quelli che vengono effettivamente trattati, ha i modelli e il meta

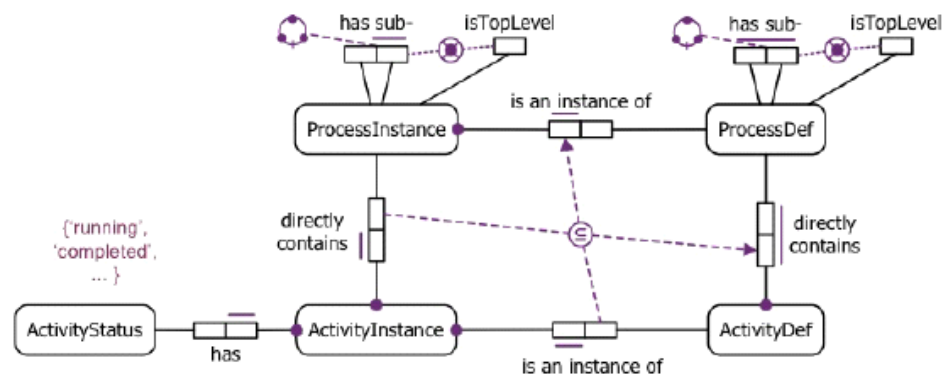
modello, ossia il linguaggio che uso per descrivere il modello stesso.

Descriveremo i processi in termini di Workflow: saranno di sistema o che prevedono interazione con le persone. Un business process ha una serie di attività di sistema, manuali o di interazione con gli utenti, che compongono i workflow.



I processi in una organizzazione complessa non sono singoli, e possono essere di innovazione, di gestione, di marketing, di sviluppo del prodotto, delle vendite ed è importante capire come si relazionano tra loro. Lo stesso processo poi non solo è correlato con altri processi, ma dipende anche dal modello delle funzioni, da che info ho, dal modello delle informazioni, da che informazioni ho a disposizione, dall'organizzazione e dall'infrastruttura tecnologica. La modellazione si appoggia su tutti questi pilastri.

Ogni istanza di processo deve essere legata a una definizione di processo, che appunto contiene più istanze di attività. Un'istanza di attività ha degli stati associati (in esecuzione, completato, ecc.). Le transizioni tra le varie attività sono generate dagli eventi che determinano il cambio di stato. Bisogna collezionare le sequenze di eventi che permettono di definire il processo da dover astrarre. Questo è definito con il mining o metodologie di costruzione.

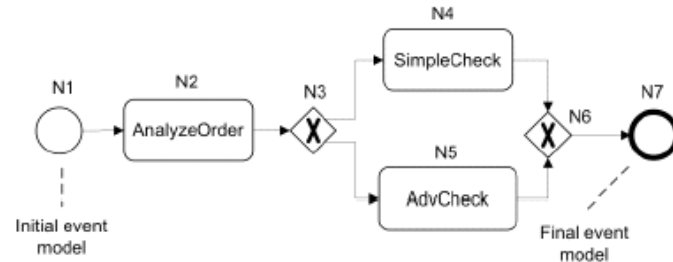


L'istanza di un processo sono insieme di eventi che determinano l'esecuzione delle attività, e ci sarà un evento che determina l'inizio, uno di fine, uno di attivazione. Quello che noi vediamo sono l'occorrenza di questi eventi che formano appunto l'istanza di un processo. Possiamo vedere gli eventi come tracce dei processi dalle quali possiamo ricavare dall'interazione con le persone o con processi di mining.

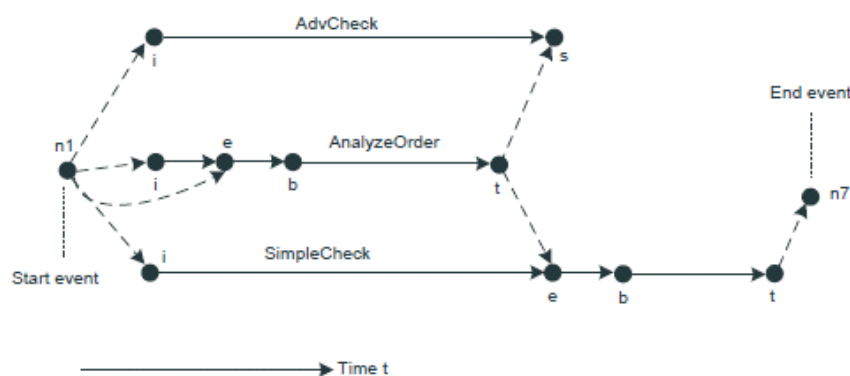
Quando abbiamo il cambiamento degli stati di una attività sono la sequenza di eventi che fanno progredire e sono questi gli eventi che vanno a determinare la sequenza di attività.

Il modello di un processo è formato da un insieme di nodi, almeno 2 nodi, e da degli archi. Un arco ha come sorgente un nodo e come target dell'arco un altro nodo. I nodi potranno essere di 3 tipi: attività, evento o gateway. L'attività descrive una attività che può essere completata, un evento è l'occorrenza di un evento o un gateway, che descrivono flussi di esecuzione,

diramazioni o unificazioni di flussi. Alcuni potranno essere etichettati con delle domande che sono condizioni che servono a capire in uno specifico caso quale è il flusso in cui proseguire. l'insieme del modello descrive quella che è una orchestrazione del processo.



M. Weske: Business Process Management,
© Springer-Verlag Berlin Heidelberg 2007



M. Weske: Business Process Management,
© Springer-Verlag Berlin Heidelberg 2007

Sopra è rappresentato un process model e l'instance process model.

All'interno di un processo tutto il flusso documentale avviene sempre con il flusso di esecuzione, ossia agli input che servono alle attività passano attraverso i flussi di esecuzione. Non ci sarà mai all'interno di un processo un dato che passa da una attività ad un'altra. Tra processi diversi che si coordinano e interagiscono per raggiungere ognuno il proprio obiettivo, la loro interazione è rappresentata tramite messaggi. Annoteremo le attività che producono documenti e quelli che li usano. Tra processi diversi non esistono flussi ma messaggi. Avremo message flow tra processi e control flow all'interno dello stesso processo.

Tra due processi potremo avere dei legami nelle istanze, ossia un evento di un processo è determinato da un'attività di un altro processo.

Abbiamo tre importanti aspetti:

Controllo flow e data flow. Questi si influenzano vicendevolmente. Per generare delle informazioni dipendo da entrambi. Prima di leggere dei dati devo avere delle attività che li producono. Anche le risorse influenzano le informazioni, ossia chi o cosa esegue l'attività influenza lo svolgimento di essa. l'assegnamento delle risorse è fondamentale

Specification of how a certain object/fact type is manipulated: CRUD (Create, Read, Update, Delete). Devo conoscere il ciclo dell'attività per poter applicare correttamente e i vincoli che ci sono tra le attività.

Dati e processi si influenzano in modo reciproco. Anche le risorse che fanno parte dell'organizzazione influenzano la definizione del processo stesso. Una organizzazione è formata da un certo numero di persone che occupano posizioni, e nelle posizioni sono definite chi può delegare il lavoro ad un altro e delle posizioni che sono associate ai ruoli. Le posizioni sono organizzate in team. I vincoli da tenere in considerazione riguardano cosa le posizioni possono fare e con quale ordine devo essere eseguite le attività. Ci sono vari fattori che influenzano il tipo di risorsa utilizzato. (vedi slide 58)

È uno standard stabilito dalla OMG.

Ha anche associato un linguaggio di esecuzione WS-BPEL, WebService. Permette una orchestrazione di servizi. Ci sono alcuni Engine che eseguono anche processi BPMN quando vengono mappati nella semantica di esecuzione di WS-BPEL.

Assomiglia a dei flow chart ma ci sono delle differenze:

- Bpmn ha una specifica formale e un mapping verso le reti di Petri per descrivere processi concorrenti. ha una semantica specifica quindi.
- Bpmn rispetto ai flowchart ha il concetto di evento, quindi posso descrivere l'occorrenza di un fatto. Quindi un certo flusso può generare degli eventi
- Bpmn vuole catturare come si interagisce tra processi diversi con scambio di messaggi ecc.

Un bpmn definisce quali eventi sono occorsi, quali attività sono state completate e quali dati sono stati prodotti.

Un bpmn ha anche dei problemi: una delle principali è che se rappresento una generalizzazione di istanze di esecuzione, le caratteristiche di queste istanze vengono perse o non riesco a caratterizzare processi legati all'istanza. Ad esempio una certa decisione di un processo potrebbe essere legata ai dati disponibili in quel momento per quell'istanza e questo spesso è difficile da generalizzare. Difatti dati e risorse sono considerati secondari e difficili da considerare insieme al control flow. Un altro problema è il many to many. Ci sono delle attività che devono essere svolte e gestite in interazione con molte altre attività e queste sono difficili da trattare e le soluzioni offerte non sono soddisfacenti. Nel processo bpmn ci sono situazioni in cui vorrei dire che la gestione di certe attività è ad hoc per certe situazioni ma è difficile da rappresentare come situazione. Conosco un ordine ma non so quando dove o perché vengono svolte certe attività.

È importante saper descrivere un processo e le problematiche che lo coinvolgono.

Viene utilizzato per descrivere :

- Processi
- Collaborazioni che possono essere coreografie o conversazioni

Possiamo far vedere la comunicazione con un altro processo anonimo, o far vedere nello specifico come avviene la comunicazione tra le varie attività del processo.

Bpmn2.0 non definisce nessuna metodologia o stile. È un linguaggio vero e proprio ma non ci dice come deve essere utilizzato. Lo stile è importante perché definisce le best practice. Serve solo per rappresentare dei modelli e nessuna spiegazione di come ricavare i modelli.

Bpmn è un linguaggio grafico ed è organizzato in 5 famiglie di elementi. Ogni famiglia è suddivisa su due strati, basic e advanced. (vedi famiglie da slide 15)

Nella modellazione seguiremo un approccio determinato da come organizziamo la presentazione stessa. Nel modellare ci saranno due modelli:

- Descrittivo, dove si individua la struttura di base in termini di gateway, attività ed eventi iniziali e finali;
- In seguito il modello sarà Analitico, dove ci saranno eventi di errore, di gestione di errore, eventi intermedi da intercettare.

Bisogna capire anche su cosa si va a modellare, e questo si fa a partire dai casi che devono andare a gestire e rappresentare, come questi partono e come questi finiscono. Per capire come decido di passare da una attività all'altra con i gateway ci devono essere dei task che portano a poter decidere, e quindi da questo inizio ragionare e vedo gli eventi che influenzano queste decisioni.

Il processo lo avremo a rappresentare come un evento di inizio, uno o più di fine e dentro i vari gateway, attività ecc che determinano la diramazione delle cose. Tutti i vari elementi saranno uniti dal process flow, il flusso di esecuzione.



Come definire le connessioni:

| | |
|--|---|
| | Sequence flow. |
| | Message flow. |
| | Directional association: link from/to a data item (data flow). |
| | Connection with other artifacts (e.g., notes, textual annotations). |

Le attività possiamo vederle di due tipi: atomiche o complesse. Le attività atomiche rappresentano delle attività che non sono ulteriormente scomponibili, mentre quelle complesse sono scomponibili e analizzabili al loro interni, e sono descritte da un processo BPMN figlio. L'attività definisce una unità di lavoro eseguita da un esecutore.

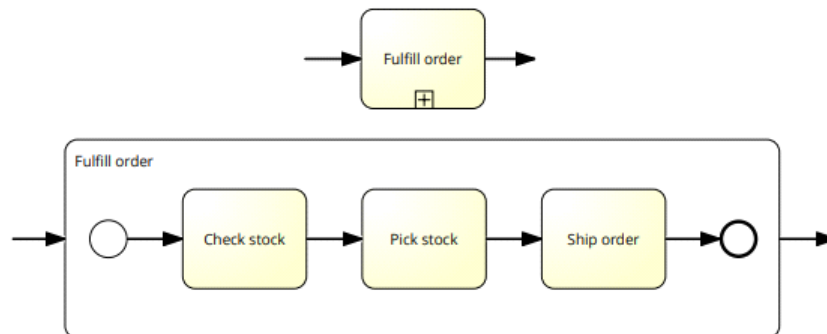
Le attività composte servono per raggiungere vari obiettivi. Principalmente quello di rendere più accessibile il nostro modello, infatti come è usuale il modello non dovrebbe essere troppo complesso alla vista da analizzare. Spesso si utilizzano sottoprocessi per raccogliere delle porzioni del processo e isolarle per analizzarle a parte. Queste porzioni si etichettano in maniera coerente in modo da avere idea di quello che avviene e serve una analisi più approfondita si scende di livello per analizzarlo ma a una vista complessiva deve essere chiaro cosa fa quel sottoprocesso. Le attività composte (o sottoprocessi) hanno delle funzioni in più, ossia delimitare delle porzioni di processo al fine di stabilire lo spazio in cui certi eventi possono essere utilizzati, o essere sensibili a certi eventi. Potremmo anche definire che un insieme di attività devono essere svolte solo sotto certe condizioni. Le attività si etichettano con la coppia **"verbo-sostantivo"**. Si possono inoltre mettere delle annotazioni sui task per indicarne la tipologia come ad esempio:

| | | |
|--|---------------|---|
| | Abstract Task | A generic task, whose type is not specified. |
| | User Task | A task under the responsibility of a human (interacting with the information system). |
| | Service Task | An automated task, managed autonomously |

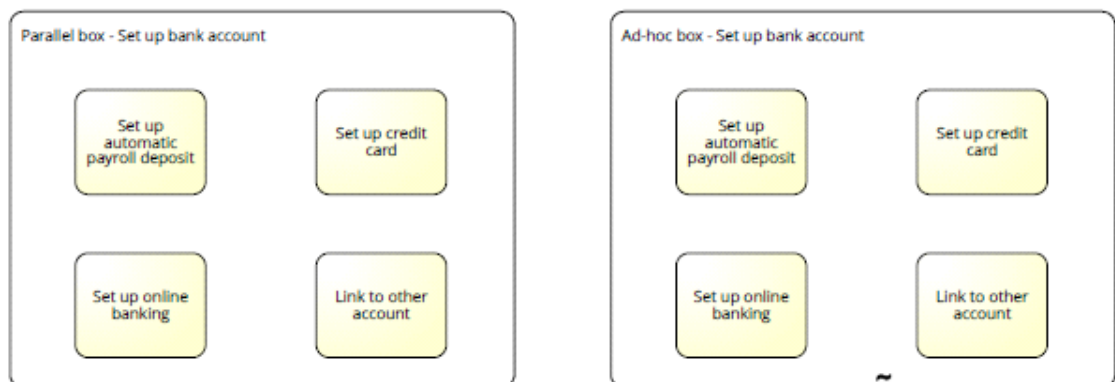
Il sottoprocesso possiamo rappresentarlo espanso nello stesso foglio del processo principale. Il sottoprocesso può anche essere rappresentato collassato ed è espandibile quando necessario. Quando un sottoprocesso in modo identico compare in più posti dello stesso processo possiamo usare la call activity. È descritta una sola volta e dove la riutilizziamo fa capo alla stessa definizione. A differenza dei programmi, è molto più facile utilizzare i sottoprocessi rispetto alle call activity.



Abbiamo la visione collassata e la visione espansa del sottoprocesso. Un sottoprocesso inizia sempre in un singolo punto ed è sempre uno start generico(ossia il pallino bianco). Può concludersi in maniera diversa, di solito per poi utilizzare i diversi modi di terminare il sottoprocesso per fare test a livello superiore.



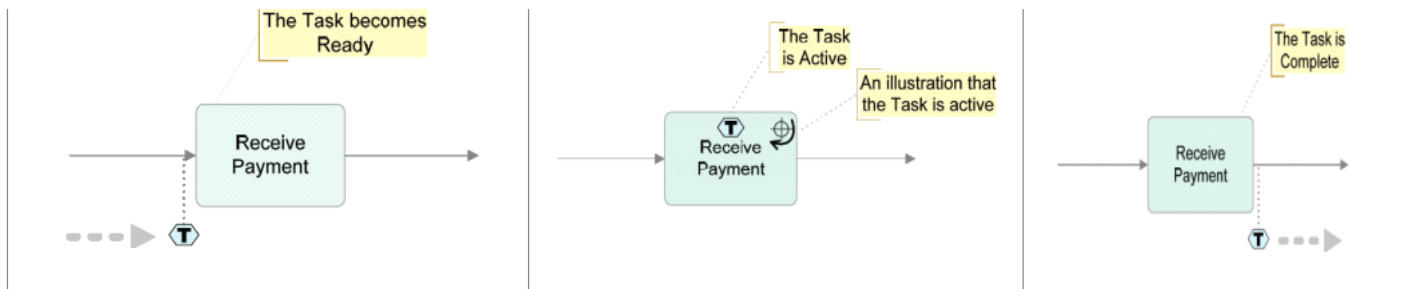
Ci sono dei sottoprocessi speciali: il parallel box identifica un sottoprocesso composto da più attività diverse che verranno svolte anche in parallelo tra di loro. Quando tutte saranno abilitate il sottoprocesso partirà e quando saranno tutte completate il sottoprocesso terminerà; l'ad hoc box(tilde in fondo) ha al suo interno le attività definite in un certo ordine ma non è specificato quello, questo perchè l'ordine potrebbe dipendere dalla situazione che stiamo trattando.



I sottoprocessi servono per avere una modularità e una visualizzazione da un capo all'altro del processo più facil e compatta. I sottoprocessi dovrebbero essere messi su fogli a parte rispetto a processi di livello superiore. Per noi poi è più facile modellare in maniera top down, pensando quindi prima a macro attività e poi scendere nel dettaglio con i sottoprocessi. Un sottoprocesso definisce dei limiti, i suoi confini, e questo permette di gestire eventi all'interno di un sottoprocesso e si potrebbe capire quando un problema non è risolvibile a livello di sottoprocesso ve deve essere mandato a quello superiore. È utile in modo che si possa rendere visibile quello funzionale a quel sottolivello, nascondendo quello non necessario.

Token Game

Semantica di esecuzione del BPMN, è ispirata dalle reti di Petri. Vogliamo rappresentare l'istanza, ossia il fatto che ci possono essere più istanze di esecuzione dello stesso processo e ogni istanza sia un caso specifico a se e che deve essere trattato diversamente e attraversare il processo per essere gestito. Il token è un oggetto teorico che descrive una sorta di simulazione su quale è il comportamento associato ai vari elementi di BPMN durante l'esecuzione di una specifica istanza di processo. Il token viene creato all' start event e attraversa tutto il flusso descritto dal processo per poi poter essere distrutto nell'end event. Quando il token è di fronte all'attività, questa diventa ready. Se entra nell'attività questa è attiva ed è eseguita con le informazioni associate al token. Il token è la collezione in maniera astratta che ha raccolto nel suo percorso specifico. Quando l'attività è completata il token esce dall'attività arricchito dalle informazioni a seguito dell'aver eseguito l'attività.



Di un processo possiamo avere più istanze di uno stesso processo in base alle situazioni diverse. Quando un token deve eseguire un sottoprocesso lo fa a seguito della completa esecuzione dell'attività precedente e di conseguenza poi il token viene messo in stato attivo nel sottoprocesso e si mette in esecuzione l'attività. Eseguire il sottoprocesso significa creare un nuovo token legato al sottoprocesso che inizia la sua esecuzione fino a raggiungere la terminazione. A questo punto il token nel sottoprocesso, arricchito delle info del sottoprocesso, può proseguire poi il flow del processo a livello più alto. L'idea è quella di una gerarchia di livelli.

I gateway

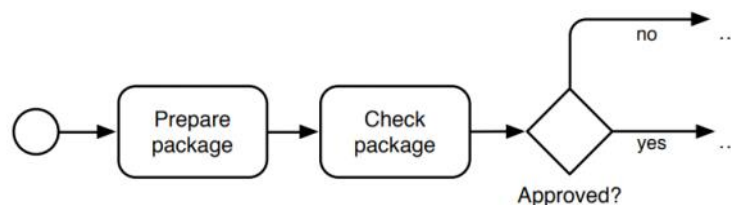
I gateway gestiscono quando un control flow diverge o converge. I gateway sono sempre di due tipi: *split gateway*, che ha una freccia entrante e molte uscenti, *il gateway join* che ha più frecce entranti e una sola uscente. A seconda di cosa è disegnato nel rombo del gateway avremmo diversi comportamenti in base al tipo di split o convergenza.

Vediamo lo XOR gateway. Si rappresenta in due modi:



Uno *XOR split* viene utilizzata per rappresentare una scelta: uno solo dei rami in uscita rappresenta il flusso che continuerà la sua esecuzione, quindi il token che si presenta a un gateway xor split prosegue una sola strada tra quelle in uscita. La scelta tipicamente è basata sulle informazioni di stato presenti in quel momento. La domanda che viene messa rappresenta un'informazione di stato che deve essere nota quando ci si appropria al gateway e quindi funge da switch tra i passi successivi in base allo stato corrente. Nel descriverlo metteremo la domanda e metteremo nelle varie frecce d'uscita le condizioni per la scelta dei rami. Devono coprire i vari casi e devono essere esclusivi tra di loro. Ci deve anche essere un'uscita.

I gateway non sono attività, non presuppongono che il gateway compia attività, ma è solo una "porta automatica" aperta o chiusa in base allo stato in cui ci troviamo. Bisogna quindi definire bene la descrizione del processo: ci deve essere una attività precedente al gateway che concretamente determini le varie strade in cui proseguire.



Lo *XOR join* rappresenta la fusione in un unico arco di proseguimento per più flussi. Serve per unire un flusso di esecuzioni che precedentemente si era diviso per uno split, quindi funge da merge. Abbiamo più frecce che vanno verso un rombo e una sola freccia in uscita. Anche questa è una porta, quindi la porta in uscita viene aperta nel momento stesso in cui una istanza di processo arriva in una delle porte, la porta si apre. Se arrivassero più istanze verrebbero continuamente aperte e portate verso l'uscita.

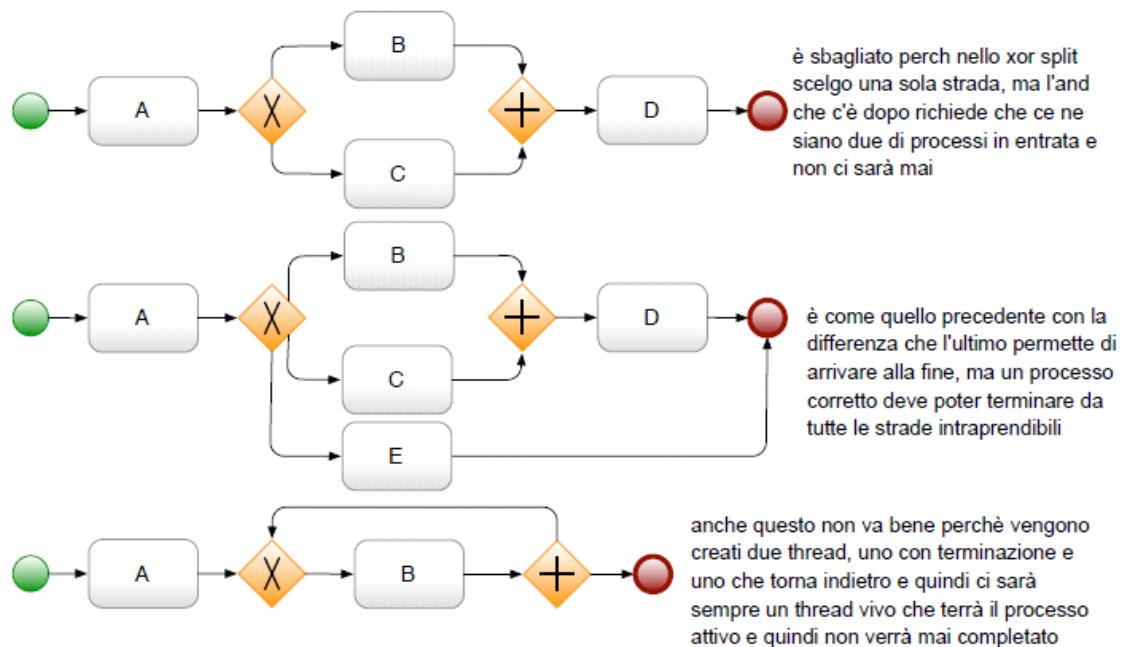
Un altro gateway molto utilizzato è il gateway AND. È un rombo con il +. È detto anche parallel gateway e modella quello che possiamo intendere come fork nei processi, ossia il fatto che da un

processo si abbia una divisione su più processi. *AND split* tratta proprio il punto in cui questi processi vengono suddivisi in più sottoprocesso. Tutti gli output del nostro gateway rappresentano un thread che è attivato nel momento in cui l'istanza si è presentata al gateway split.

Non vi è condizione per il fork, è sufficiente che le istanze in esecuzione si presentino al gateway, senza condizioni. Questi thread possono poi eventualmente riunificarsi.

L'*AND join* rappresenta un punto di sincronizzazione dei thread. Quando per ogni porta in entrata si presenta un thread, a quel punto la porta in uscita verrà aperta e ne uscirà un unico thread, ma solo quando ci sono tutti i thread entranti.

Esempio:



Ricorda: se ci sono due linee uscenti dalla stessa attività è implicitamente un AND. Quando invece due frecce si riuniscono in una attività è implicitamente uno XOR join.

Eventi

L'evento di start è rappresentato da un cerchio con il bordo sottile. Indica dove parte il processo e come un processo parte. Ogni sottoprocesso ha un unico punto di start e questo deve essere sempre di tipo generico, quindi il pallino semplice. Un processo a top level può avere dei punti di partenza multipli. A top level si possono associare dei trigger all'evento, quindi un modo che determina la partenza del processo.

Il trigger indica la semantica dell'evento, il perché l'evento parte, quindi identifica il significato del processo e che cosa tratta. Lo start è importante se c'è un trigger o una annotazione è associata per conoscere quale è l'obiettivo di esso. Ogni caso determina uno start, e l'istanza che parte da esso rappresenta la gestione di quel caso.

None start event, solo un cerchio.

Non ha trigger al suo interno ed è l'unico start per un sottoprocesso. In un processo a top level può significare che o ci sono trigger o il trigger per cui parte non è specificato o ancora che il processo viene fatto partire in modo autonomo, senza una condizione specifica descritta ma per intervento manuale ad esempio. In un sottoprocesso significa che questo parte quando il suo processo padre lo attiva, quindi che il token arriva al sottoprocesso e lo attiva.

Il processo è attivato quindi quando il trigger scatta (con i raggi intorno al cerchio), viene creato il token e parte l'esecuzione del processo.

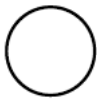




Message start event, è rappresentato da un cerchio fine con all'interno una busta da lettere bianca. Il processo è attivato nel momento in cui viene ricevuto un messaggio. Solitamente è associato anche ad un arco del dataflow in entrata. Ad esempio ci può essere una collaborazione con scambio di messaggi e il processo parte quando l'altra parte invia un messaggio. Il messaggio arriva da fuori il processo, che ha sempre un'origine esterna al processo e denota la consegna verso questo processo del messaggio. Per sottolineare si mette anche l'etichetta "Receive X" per chiarire che quello è

l'evento trigger. Un message flow può essere disegnato per rappresentare chi manda il messaggio.

Timer Start Event, rappresentato da un cerchio con un orologio. Il Timer può essere usato in due modi: fa partire un processo in modo schedulato in un punto specifico nel tempo oppure in modo ricorrente.

Partenze multiple, ci sono di due tipi: il multiple start, cerchio con esagono, che parte quando ogni volta che uno dei molteplici trigger associati allo start occorre, oppure il *multiple parallel*, cerchio con una croce, che inizia ogni volta che tutti i trigger collegati con lo start occorrono.

Con gli start alternativi associamo delle attività diverse a seconda di come parte il processo. Prima di gestire l'attività devo effettuare attività precedenti in base ai vari modi alternativi disponibili per partire. Abbiamo un trigger event alternativo per ogni start, quindi quando uno occorre gli altri sono disabilitati e viene associato un case all'evento che fa partire il processo.





| None start | Message start | Timer start | Multiple start | Multiple parallel start |
|---|---|---|---|---|
|  |  |  |  |  |

L'End Event è rappresentato da un cerchio dal bordo spesso, che corrisponde all'end generico, ossia il None End. Un processo o sottoprocesso termina quando l'esecuzione associata ad un thread raggiunge un end event e non ci sono altri thread associati alla stessa istanza ancora presenti come flusso di esecuzione. Quindi tutti i thread associati allo stesso processo devono terminare. Quando arriva il token ad un end viene attivato l'end e il token cessa di esistere. Possiamo anche avere più terminazioni diverse, basta che ne venga raggiunta una.

Possiamo avere la terminazione con messaggio, cerchio spesso con lettera nera, che significa che arrivo a fine processo e viene emesso un messaggio con anche indicazione del destinatario.

La terminazione ad occhio di buca, cerchio con cerchio nero all'interno, definisce che il processo termina completamente, anche se ci sono altri thread in esecuzione in quel momento. È una sorta di exit forzata. È utile in situazione di errore e si deve terminare qualsiasi cosa in corso.

La terminazione multipla è rappresentata dal pentagono nero e rappresenta il fatto che vengono emessi più eventi in output, quindi ci sono più cose che avvengono contemporaneamente alla stessa terminazione.

| None end | Message end | Occhio di buca end | Multiple end |
|---|---|---|---|
|  |  |  |  |

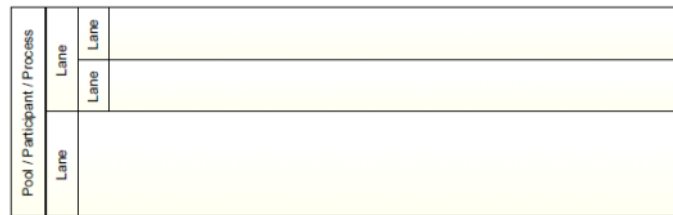
Il sequence flow determina il flusso di controllo all'interno di un processo. Tutte le attività, gateway ecc devono essere collegati tramite sequence flow. Un sequence flow confinato in uno specifico livello di processo, quindi non ci sono frecce che entrano nei sottoprocessi o altro, quindi i vari livelli non si intersecano con i flow, creando una chiara gerarchia.

I messaggi si collegano con la freccia tratteggiata con pallino che identifica da dove parte. Questa è usata solo per rappresentare data flow e non scambi di info all'interno dello stesso processo, ma tra processi diversi.

Il modello organizzativo

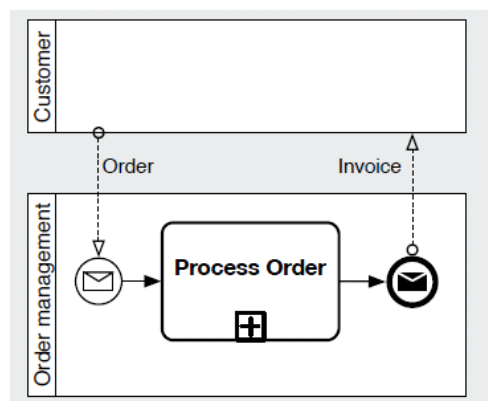
Il processo descrive come una organizzazione agisce, ma non abbiamo detto come organizzare le

risorse che fanno parte. A livello di modello non si specifica la risorsa particolare, ma ruoli o gruppi a cui sono attribuite le varie attività. Le risorse specifiche invece si annotano nelle specifiche delle attività. Per definire le risorse si usa il Pool.



In testa abbiamo il nome del processo o dell'entità, all'interno è strutturato su varie corsie e sottocorsie. Queste rappresentano classi oppure se sono singole, ruoli. Un pool esterno collassato non ha elementi all'interno. Gli elementi che si vanno ad inserire devono essere collocati nel ruolo a cui sono attribuite le attività, la ricezione dell'evento ecc.

Esempio:



In quello collassato non si vede da dove parte il messaggio ecc, all'interno invece è visibile se invece il pool è esteso.

Le frecce di message flow possono essere sia collegate a attività specifiche, attraversando il pool, oppure se non è specificato sono attaccate al bordo del pool.

Ci sono altri elementi: i DATI. Ci sono due categorie di simboli, quello del foglietto e quello rappresentato dal data storage. La differenza tra i due è che nel "data storage" le info sono persistenti, mentre il foglio porta informazioni volatili, quindi esistono limitatamente all'istanza del processo. Le informazioni nel data storage esistono indipendentemente dall'istanza.

| | |
|--|--|
| | Data object: local variable inside a process level, pointing to a temporary unit of information. |
| | Data object reference: refers to a data object in some state. |
| | Variable representing a collection of data objects. |
| | Data store reference: reference to persistent unit of information, manipulated by the process but also external entities. |

Le frecce puntinate definiscono l'I/O dei dati, quindi lettura con freccia che va dal dato all'attività, mentre l'output per aggiornare, creare ecc il dato ha la freccia che parte dall'attività e va verso il dato. Questo rappresenta il data flow e insieme agli altri due flussi determinano dei vincoli.

Ci sono anche le annotazioni, ossia descrizioni testuali che si attaccano alle attività per spiegarne l'uso o altro.

-----LINEE GUIDA STILISTICHE E METODOLOGICHE

Vedi slide

La parte analitica va ad espandere il livello visto prima con una reazione agli eventi, quindi la gestione di eventi, ovvero se qualcosa succede nel processo in uno specifico punto temporale.

Le reazioni che possiamo avere sono principalmente:

- Throwing an event, come il processo segnala che qualcosa è successo, e questo è rappresentato dal trigger
- Catching an event, come il processo risponde al segnale che qualcosa è avvenuto. Quando viene catturato un evento lungo un processo
- Boundary, catch che si attaccano ai confini delle attività e ce ne sono di due tipi, uno che interrompe l'attività (continua) a cui sono fisicamente attaccati e uno che non interrompono (tratteggiata)

I segnali sono icone all'interno di cerchi. Vedremo gli eventi intermedi, ossia che qualcosa è successo durante l'esecuzione di un processo ed è indicato con un cerchio a doppia linea.

THROWING

Il tipico uso è quello ad esempio di inviare un messaggio, quindi si genera il messaggio e poi il flow continua immediatamente. L'evento non implica risorse impegnate per eseguirlo o persone che devono metterci del tempo per farlo, e questa è la differenza dal definire invece un'attività per inviare il messaggio.

Il token arriva, scatta immediatamente e prosegue.

La ricezione del messaggio, ossia il catching di un evento intermedio, si rappresenta con doppio cerchio ma la letterina, nel caso del messaggio, è bianca. Non appena il sequence flow raggiunge l'evento, il processo aspetta che quando il messaggio arriva e a quel punto scatta l'evento e poi continua il processo. Ma l'evento di catching attende il messaggio prima di proseguire. Questo funziona per molti tipi di eventi ma non per l'errore.

Questi sono collegati da due frecce.

BOUNDARY

Questi sono dei catching che vengono disegnati sul bordo dell'attività, quindi non hanno frecce entranti. Determinano due possibili uscite: una, quella normale dell'attività, e una associata a un flusso d'eccezione. Questo significa che se durante l'esecuzione dell'attività non occorre l'evento segnalato nel boundary event, il token all'interno prosegue l'esecuzione normalmente, altrimenti se durante l'esecuzione di essa scatta l'evento indicato nel catching, il token principale viene annullato e parte un token associato al boundary event, che viene creato quando si entra nell'attività, e continua il flusso seguendo l'exceptional flow.

Ci sono boundary event tratteggiati. Questi non interrompono l'attività a cui sono collegati, ma partono se l'evento descritto occorre durante l'esecuzione dell'attività. In questo caso partiranno due token, quello del flusso normale e quello del flusso eccezionale.

TIMER

Possono specificare una durata di attesa oppure un punto specifico nel tempo che deve essere raggiunto. È importante sapere che i timer event non devono essere utilizzati per rappresentare la durata dell'attività. Le stime delle tempistiche sono degli attributi che possono essere aggiunti dagli strumenti.

Se vogliamo rappresentare in qualche modo un limite massimo per la durata, altrimenti facciamo qualcosa, questo può essere definito come boundary event, quindi attaccato all'attività. Se dopo un certo tempo una certa attività è ancora attiva, allora viene interrotta dopo il tempo che è appunto indicato dall'evento timer boundary. Usare quello tratteggiato è indispensabile se non deve essere interrotta l'attività.

BPMN definition for “message”

The content of a communication *between two participants*.

- Could represent even **material flow**, i.e., the delivery of a physical object.

A message has an **item definition** that specifies the message payload, i.e., an information or physical object.

Warning

Consider “send” and “receive” as BPMN keywords that denote the exchange of a message from the point of view of the emitter or destination.

Location of a message

A message could correspond to many distinct message flows, representing the message in different situations of the process.

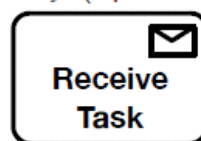
La send message può essere rappresentata anche da una attività che ha in un angolo una letterina. Questa rappresenta un task di invio. La differenza principale con invece l'evento di invio del messaggio, è che in BPMN gli eventi sono considerati istantanei e quindi non hanno riferimenti a tempistiche o su chi è l'esecutore. Se è rilevante la modellazione dell'esecutore, dei tempi di esecuzione o del costo dell'attività d'invio allora si usa l'attività. Eventualmente si potrebbe mettere prima dell'evento send una attività che prepara il messaggio.



La differenza con un invio normale di un messaggio è che si può associare un boundary event ad esso e si possono mettere dei decoratori. Il send message non si deve usare all'interno dello stesso processo, ma per comunicare con un altro processo esterno.

La receive ha una letterina vuota, che evidenzia il fatto che aspetta un messaggio. Questa attività blocca il processo fintanto che non arriva il messaggio, proprio come l'evento intermedio di ricezione del messaggio. Anche qui la differenza è che c'è un esecutore che aspetta il messaggio e che deve fare del lavoro per riceverlo.

Anche qui si possono mettere dei boundary event.



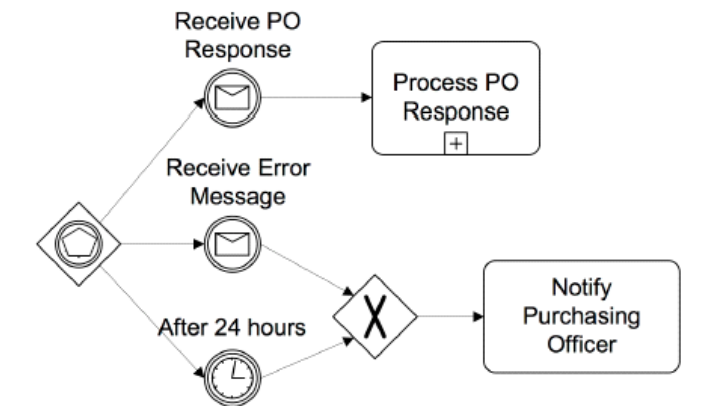
In generale abbiamo sia eventi intermedi che attività per mandare e ricevere messaggi. La comunicazione è implicitamente asincrona, cioè il mittente non blocca il processo per mandare un messaggio e non attende una conferma di ricezione.

Es: un processo di generazione di una carta di credito parte quando il cliente ne fa richiesta. In questo esempio non è prevista la gestione della non risposta del cliente. Si potrebbe mettere uno xor con un timeout che aspetta 7 giorni e poi scatta il tacito rifiuto. Ma a quel punto non avrebbe senso aspettare 7 giorni per verificare se c'è stata una risposta, infatti se risponde immediatamente sono tempi di attesa inutili. Una soluzione potrebbe essere aggiungere l'attività di ricezione, anche se solitamente la ricezione è automatica, quindi nemmeno questo va bene.

In questi casi quello che serve è una scelta guidata dagli eventi, ossia a seconda di quello che accade esegue una cosa piuttosto che un'altra. È simile allo xor ma invece di dipendere da una attività

precedente, dipende dall'occorrenza di un evento . Questo si usa quando la decisione non può essere presa prima di mettere la scelta stessa, ma da quello che occorre.

Se ci sono scelte che non possiamo compiere prima di mettere la scelta stessa, quindi prima. La decisione deve dipendere da quello che occorre successivamente. Si usa il seguente simbolo:



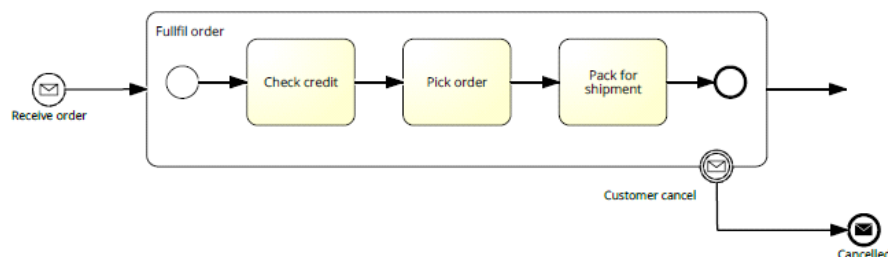
Dentro al gateway vi è un evento intermedio e dentro c'è il multiplo che significa "appena occorre un evento, parti". Il processo partiva appena uno di questi trigger scattava.

Si mette in attesa e non appena uno dei tre eventi occorrono, parte. Il processo si ferma nel gateway e poi prosegue solo in una delle alternative.

Con i token dal gateway event partono 3 token, uno per ogni evento. Il primo che occorre fa scattare il suo token e gli altri vengono invece eliminati.

Es: vedi slide 33

Il cliente può cancellare l'ordine mentre è completato o spedito e ci sono due tipi di gestioni di questi due casi. Quando non è stato ancora spedito durante il processo che potrà alla spedizione potrebbe esserci un caso di messaggio non sollecitato che indica quando il cliente effettua una cancellazione: non richiesto ma che potrebbe occorrere. Quello sollecitato è la risposta a una domanda posta da noi. Per dire che noi gestiamo questo messaggio in un certo modo se occorre in determinate attività lo si può segnare in questo modo:



Cioè si mettono quelle attività sensibili al messaggio non sollecitato all'interno di un sottoprocesso in modo che sia chiaro che il messaggio possa essere catturato in quel punto.

ERROR EVENT

L'evento di errore è un'eccezione alla fine normale di un processo. Soltamente lo si trova alla fine perché se c'è un errore interrompe il flusso. Nel caso non interrompa è una sorta di warning.

Se si intende errore una situazione che ha compromesso il flusso, si ha il termine.

È un end state ed indica un codice di errore che lo eslica. Il codice d'errore serve anche per individuare l'opportno handler. l'evento di errore lancia un "segnale" che potrà essere catturato da un handler. Per capire l'opportuno handler lo riconosco dal codice dell'errore.

Se l'errore occorre all'interno di un task io posso catturarlo con un boundary event attaccandolo all'attività in cui può verificarsi. Questo boundary event è sempre di tipo interruzione. Quello che invia l'errore sarà nero, mentre quello che fa il catch è bianco. Il simbolo è un fulmine. Possiamo anche averlo su un sottoprocesso, in questo caso allora l'interno del sottoprocesso deve contenere un evento di errore in un end state.



Semanticamente l'errore è a vari livelli e ogni livello rappresenta un "record di attivazione". Quando occorre un errore in un determinato livello di processo si interrompe l'esecuzione in corso. Tutti i thread paralleli di quello stesso livello del processo che ha l'errore vengono terminati. A questo punto il segnale di errore viene generato e propagato al livello sopra. Se il livello superiore ha un boundary che corrisponde all'errore, allora il segnale è catturato e si prosegue nel flusso che gestisce l'errore. Preferiamo l'errore allo XOR perch magari ci sono situazioni che devono esser evidenziate come errori veri e propri, da più enfasi.

ESCALATION EVENT

È rappresentato da una freccia che punta in alto. Questo è un warning, detta mild exception. Cioè è un avviso ed è diverso dagli errori perch possono essere messi all'interno del processo e non solo finale perchè non interrompe. Questo tipicamente è catturato da un catch boundary senza interruzione del processo principale, andando in parallelo.



SIGNAL EVENT

Rappresentato da un triangolo vuoto quando lanciato e triangolo nero quando viene catturato. È un event broadcast a tutti i thread attivi nel processo e a tutti i partecipanti attivi. All'interno del processo ci permette di effettuare comunicazioni a tutto il processo nei vari thread.

Es: se vi è un fallimento di una attività, si lancia un segnale che è catturato sui thread paralleli che interrompono le attività in corso in vista della segnalazione.



CONDITIONAL EVENT

Sono rappresentati da un foglietto con una lista e permettono di rappresentare condizioni di vario tipo sui dati o regole di essi.





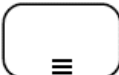
SUBPROCESS EVENT

È un handler che possiamo associare ad un processo per identificare che se occorre un certo evento durante il processo. È un processo attaccato che può scattare a causa dell'evento trigger ed è un processo che inizia con l'event che lo scatuisce e non quello none. Lo si mette all'interno del sottoprocesso in cui va a lavorare piuttosto che, con la gestione normale, ad un livello ulteriore. È un sottoprocesso tratteggiato che espanso ha l'evento trigger.



DECORATORI

Le decorazioni delle attività sono delle annotazioni che posso mettere centralmente in basso alla nostra attività.

| | |
|---|---|
|  | Basic task. |
|  | Compensation task. |
|  | Loop task (looping information attached to the activity). |
|  | Multi-instance task with parallel composition (expression attached to the activity to calculate the number of instances). |
|  | Multi-instance task with sequential composition. |

Il task di compensazione serve a correggere dei problemi che ci sono precedentemente. I task multiistanza significa che sono multiple, sia in parallelo che in sequenza.

La loop activity (freccia) è una che può essere ripetuta un certo numero di volte. La condizione di ripetizione può essere rappresentata in una annotazione agganciata ad essa (tipo Until X). Le iterazioni sono eseguite sequenzialmente e il numero di iterazioni è stabilito dinamicamente.

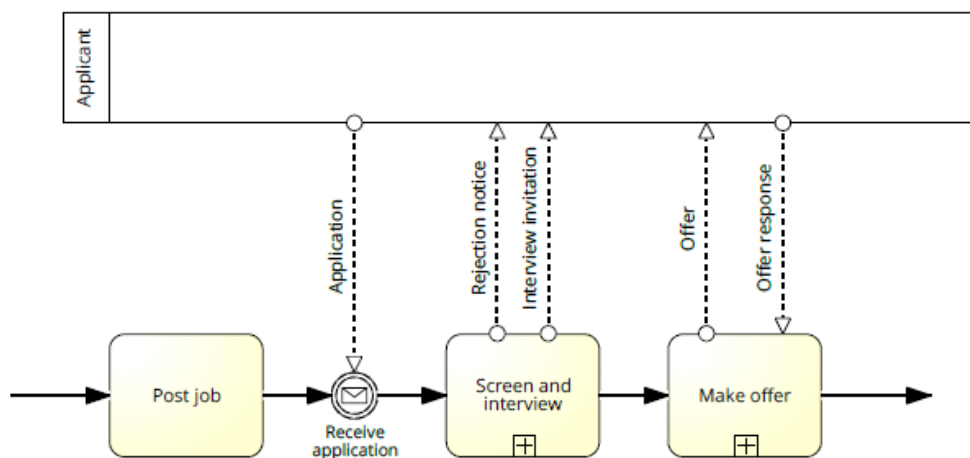
Corrisponde ad un gateway che ha il flusso che torna indietro.

Le multiistanza permettono di gestire attività molteplici volte, tipicamente associata a una attività ad un elemento di un insieme. Tipicamente il numero di istanze non è determinato dinamicamente, ma è nota a priori. Quello in sequenza è una sorta di for mentre quello in parallelo è autoesplicativo. L'attività è completa quando tutte le istanze sono terminate. Lo si fa così in modo tale che si possa gestire più facilmente un boundary event. Le istanze multiple permettono di rappresentare situazioni in cui ci servono più attività per gestire un caso solo.

LE MULTIISTANZA SONO IMPORTANTI perché alcuni processi abbastanza normali da pensare, in bpmn senza il multiistanza e senza alcuni accorgimenti non sarebbero possibili da trattare.

Es slide 58:

Il caso del processo è la copertura del posto vacante, non le interviste ai candidati. Questo è un problema one to many perché abbiamo un solo posto e molti candidati. La parte many applicants deve sincronizzarsi con la parte "un lavoro", ossia che ad uno solo verrà offerto quel lavoro e se viene assegnato il posto non si devono continuare le interviste.

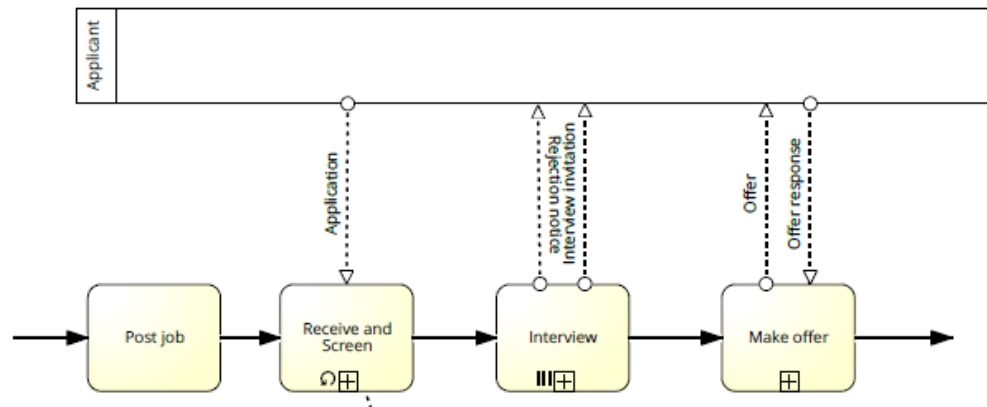


Questo processo è sbagliato perché stiamo trattando un solo candidato per volta e non è detto che quello stesso candidato sarà quello che avrà il posto. Considerando che il processo è per un posto di lavoro abbiamo bisogno di più processi di screening e interviste parallele.

Una prima soluzione a questo problema è utilizzare un sottoprocesso in loop. Viene effettuata la

screening e l'intervista in loop finchè non finiscono i candidati o finchè non viene trovato il candidato adatto. Il problema di questa soluzione è che non è veritiera perchè i candidati non vengono analizzati uno di seguito all'altro e di certo non si rimane in attesa dei candidati.

Un modo per migliorare è ottimizzare il processo andando a mettere una loop activity per l'analisi dei CV e poi si procede con le interviste in parallelo, quindi con una multistanza.



Il one to many all'interno di un pool ha bisogno di seguire questi passi:

Guideline

1. Identify all N:1 activities.
2. Surround them in a loop or MI activity or a combination thereof.

Main issues:

- If in a process phase we do not know the number of instances in advance, we can only use loop.
 - Pipelining impossible.
- If in a process phase we know the number of instances in advance, we can use MI.
 - Pipelining still impossible across phases.

Se non conosciamo il numero di istanze in precedenza possiamo solo usare un loop, invece se ne conosciamo il numero usiamo la multistanza. Non è possibile la pipeline, cioè non si possono fare loop e multistanza contemporaneamente. Nel caso dell'esempio prima si devono analizzare le candidature e poi si possono fare le interviste in parallelo.

Una soluzione per aumentare la pipeline e quindi rappresentare il processo che si svolge su più linee di esecuzione parallele che sono asincrone è usare un processo multipool. Un processo però dovrebbe essere associato ad un solo pool. Per fare ciò si dovrebbe suddividere il processo originale su più processi, andando a rappresentare il business goal su più processi.

Nell'esempio servirà per gestire i candidati, la parte many. Un altro processo invece verrà gestita la parte one, ossia il posto libero che aprirà l'offerta e poi la chiuderà al momento opportuno. Nella parte many sarà il processo che accoglie le offerte, fa lo screening, fa le interviste e fa l'offerta se il candidato è ritenuto adeguato. Questi due processi non sono indipendenti, ma anzi si devono sincronizzare.

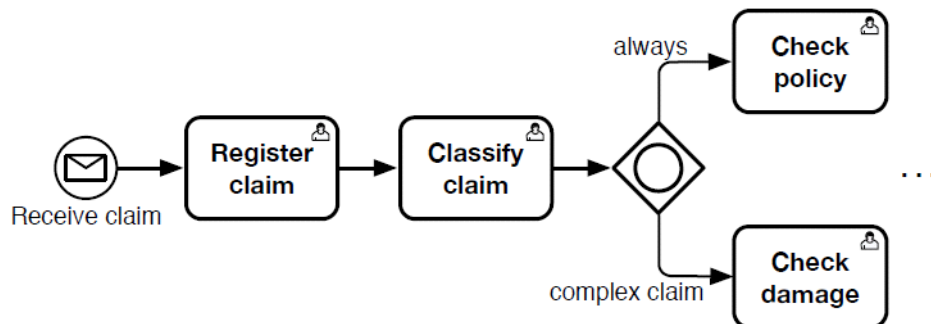
Per la sincronizzazione multiprocesso serve un datastore, di segnali che devono essere distrivuiti lungo il control flow per poter triggerare parti che stanno sui due processi e infine messaggi scambiati tra i processi per comunicare.

Es: viene rivisitato il processo per l'assunzione di un posto di lavoro con due process pool: Evaluate Candidate e Hiring Process. L'Hiring process inizia autonomamente quando si libera il posto o quando il posto viene assegnato lo si indica come riempito. L'Evaluate Candidate Process inizia quando arriva un CV per un posto specifico e se questo non è disponibile viene terminato il processo. Se invece il

una o entrambe.

In questi casi è utile l'OR-split, rappresentando da un cerchio all'interno del rombo del gateway. Il gateway OR è inclusive gateway che rappresenta scelte indipendenti. Almeno una deve essere vera e se più di una è vera quelle vere vengono attivate in parallelo. Può esserci l'altrimenti nel caso in cui nessuna delle porte precedenti sia vera, allora c'è la via d'uscita. Questa è fatta per modellare le opzioni.

Nell'esempio precedente si ha:



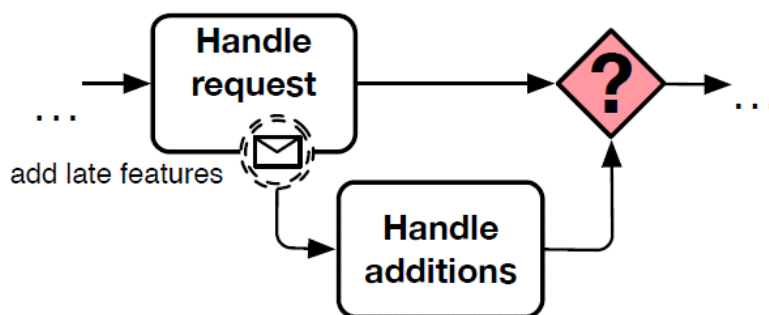
In alcuni casi funziona come un AND in altri come uno XOR.

Per indicare quando una strada è quella di default nel caso in cui non ci siano le altre si mette una sbarra sull'arco.

Con il token si ha che arriva sull'or e in base alle condizioni se ne include più di una i token vengono abilitati su tutte quelle attive e lavorano in parallelo. Il gateway join attende tutte quelle attivate prima di proseguire per poi unificarsi. Funziona come un and variable dove più di una può essere attivata ma si devono sincronizzare.

OR-Join è utile quando ci sono casi in cui si attiva una strada sola o entrambe, e in entrambi i casi deve proseguire e sincronizzare in caso di più strade. Solo L'OR può fare questa cosa.

Se faccio un boundary event che non interrompe, potrei avere l'attività principale e avere quella boundary opzionale che poi si deve fondere a quella principale e l'unico gateway per unificarle è l'OR-join perchè o una o entrambe devono poter essere gestite.

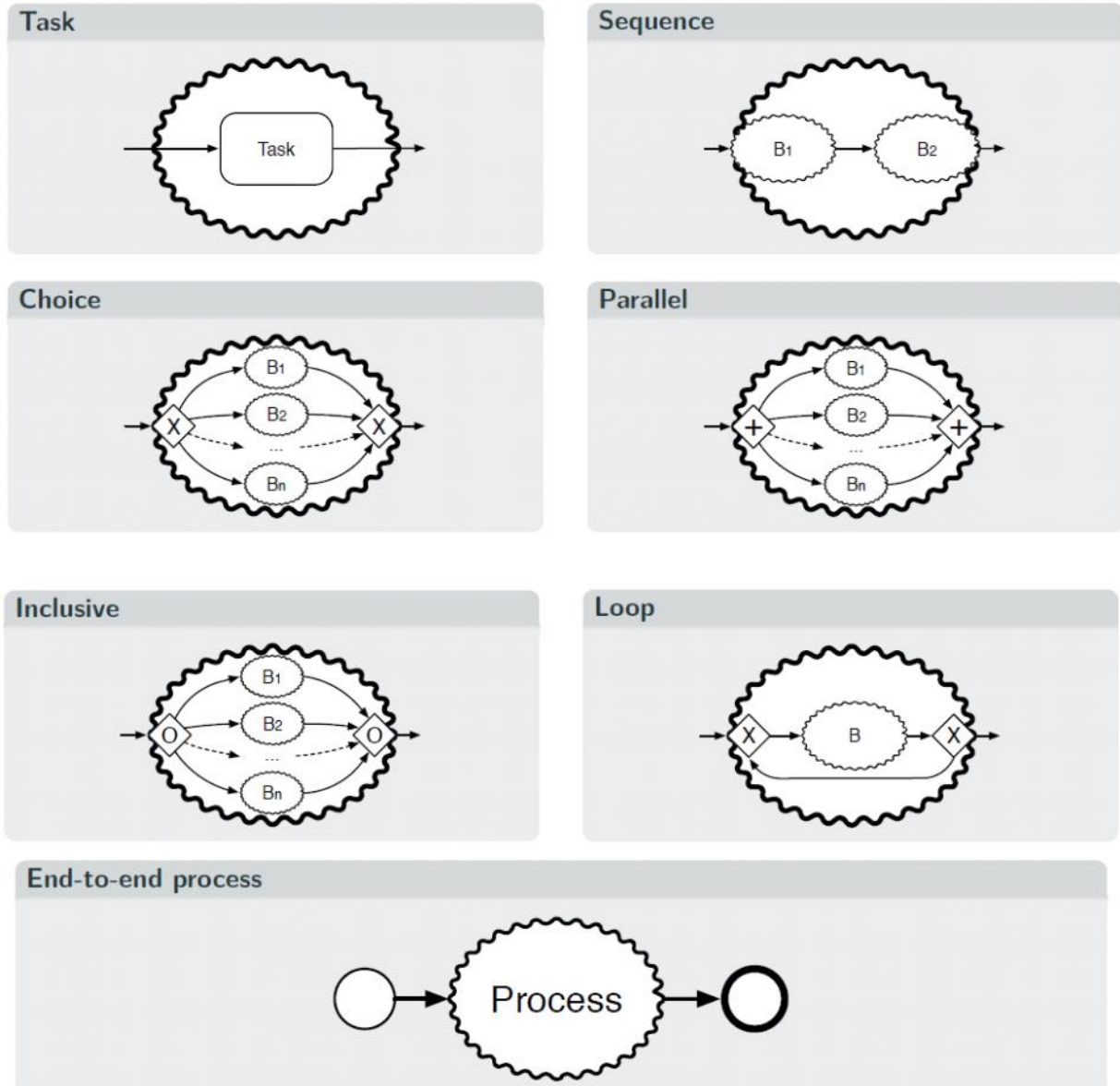


L'OR-Join quindi ha una semantica che deve riunire quelli attivati. Per XOR e AND basta sapere localmente cosa succede per aprire le porte perchè noto a priori. Il problema con l'OR invece è che bisogna conoscere cosa succede prima, dove si sono attivati i token, per sapere quali porte devono essere in attesa, ma non è sempre detto che si parta da uno split OR quindi serve sempre analizzare tutto il flusso precedente per capire come agire nel join e per sapere quanti thread ci sono.

L'OR è l'unica opzione quando ad esempio ci sono da riunire flussi d'eccezione, e che quindi non si sa se effettivamente si entrerà in eccezione o meno.

In queste casi la cosa migliore è strutturare rigidamente il processo, ma non sempre è facile farlo.

Strutturare vuol dire che ogni uso di gateway deve corrispondere un join e uno split agli estremi, perchè solo così si sa la porta che ha determinato e poi che ci sono blocchi con una sola entrata e una sola uscita, si riesce a legare l'OR join dove posso andare a leggere quanti ne vengono generati. Questi si chiamano Single Entry Single Exit blocks (SESE).



Anche con l'OR sempre un'entrata e un'uscita e idem per i loop. Questo è il modo usuale con cui alcuni testi suggeriscono di realizzare i processi.

Se c'è un SESE block, si riesce a fare una semantica semplice per l'OR join andando a legare lo split corrispondente e quindi sapere quanti token si devono attendere.

Altre forme di sincronizzazione possibili sono il discriminatore:

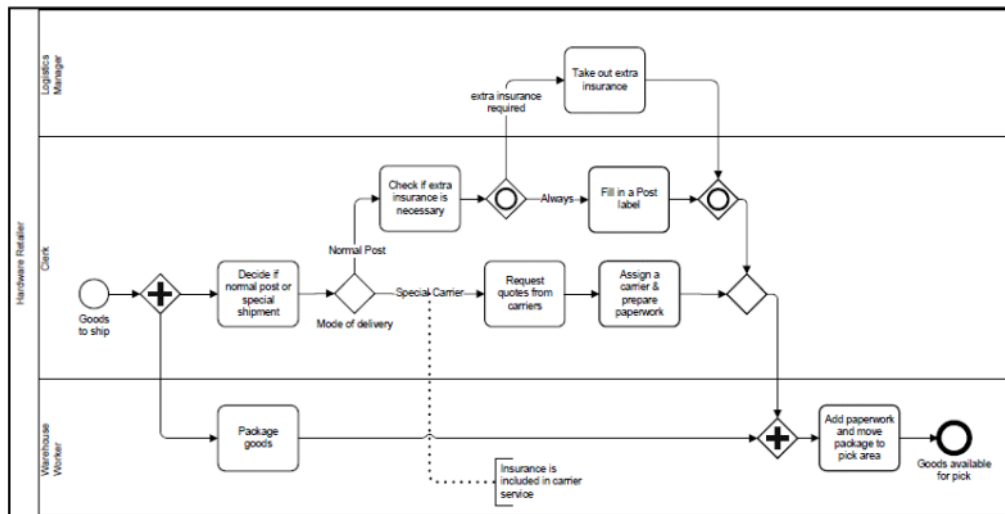
Il gateway di discriminazione serve a dire "il primo che arriva, passa e blocca gli altri" quindi seleziona il primo che passa ed è rappresentato da un asterisco in un rombo.

Esempi BPMN

martedì 30 novembre 2021 02:16

n0F))kB#txV3\$JBG password
bpmn

HARDWARE RETAILER

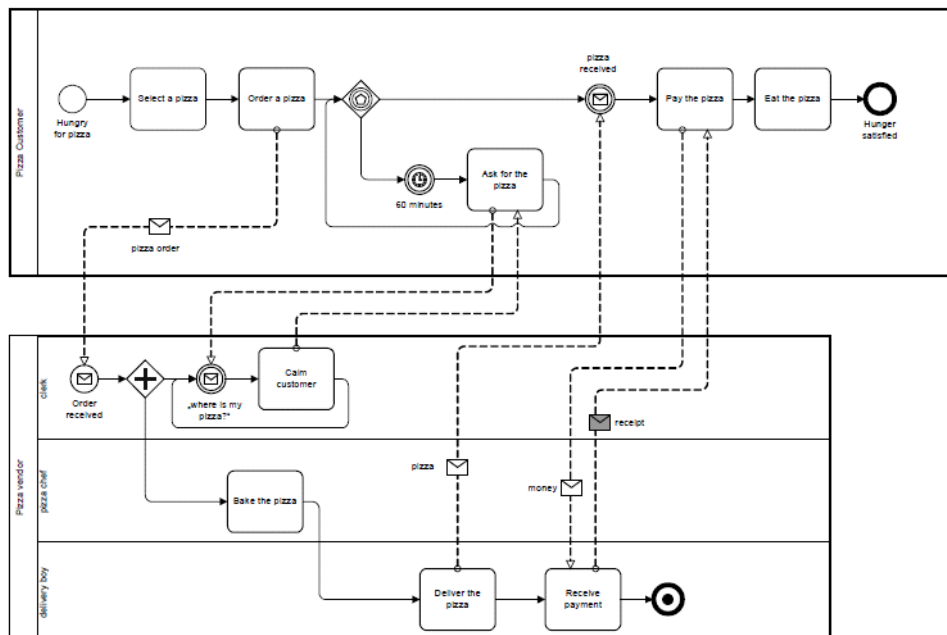


È su un pool che ha 3 ruoli.

Parte quando abbiamo della merce da spedire. Partono due processi in parallelo: uno deve decidere se inviare per posta normale o fare una spedizione speciale, intanto il magazzino impacchetta la merce. Se si fa la spedizione normale è in alternativa con la spedizione speciale e qualsiasi sia la spedizione si fanno le etichette per la spedizione, invece opzionalmente si fanno quelle dell'assicurazione. Infine ci si sincronizza con il magazzino e nel momento in cui il pacco è pronto si attaccano le etichette e si aspetta il corriere. Nel caso in cui si fa la spedizione speciale vi si aggiunge il pagamento in più.

Questo è un SESE block dove tutte le cose sono accoppiate con una sola entrata e una sola uscita.

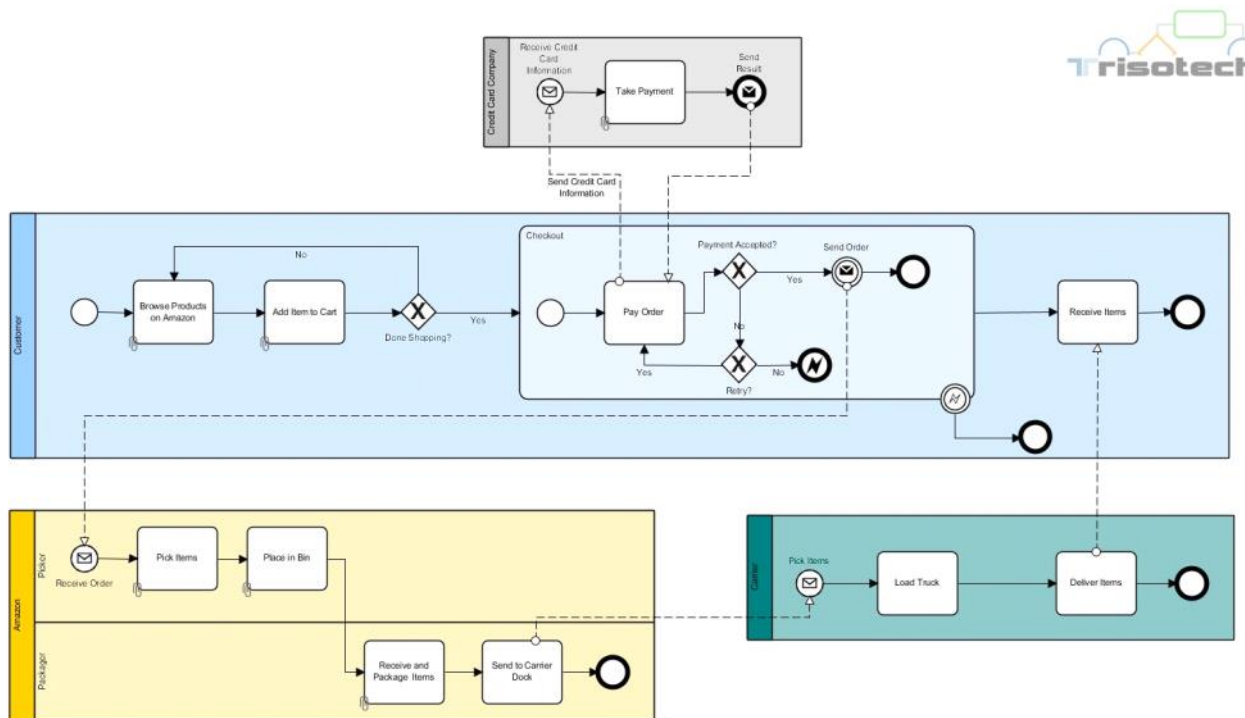
PIZZA



Abbiamo un'interazione tra due processi, il cliente e la pizzeria. Il cliente quando ha fame seleziona la pizza e la ordina, e l'ordine viene inviato alla pizzeria. Intanto questa riceve l'ordine e a partire da quel momento ha due thread paralleli. Uno è la preparazione della pizza a cui segue la consegna della pizza al cliente. Il cliente quando la riceve, la paga e poi la mangia. Però potrebbe succedere che la pizza non è stata ricevuta e sono passati 60 minuti. In questo caso si ha una even driven choice. Nel momento in cui sono passati i 60 minuti, avviene una telefonata alla pizzeria. Questa telefonata è gestita da un secondo thread che dà spiegazioni al cliente riguardo il ritardo e poi si rimette in attesa.

della chiamata. Anche il cliente si rimette in attesa della sua pizza. Se passano altri 60 minuti questa interazione si ripete.

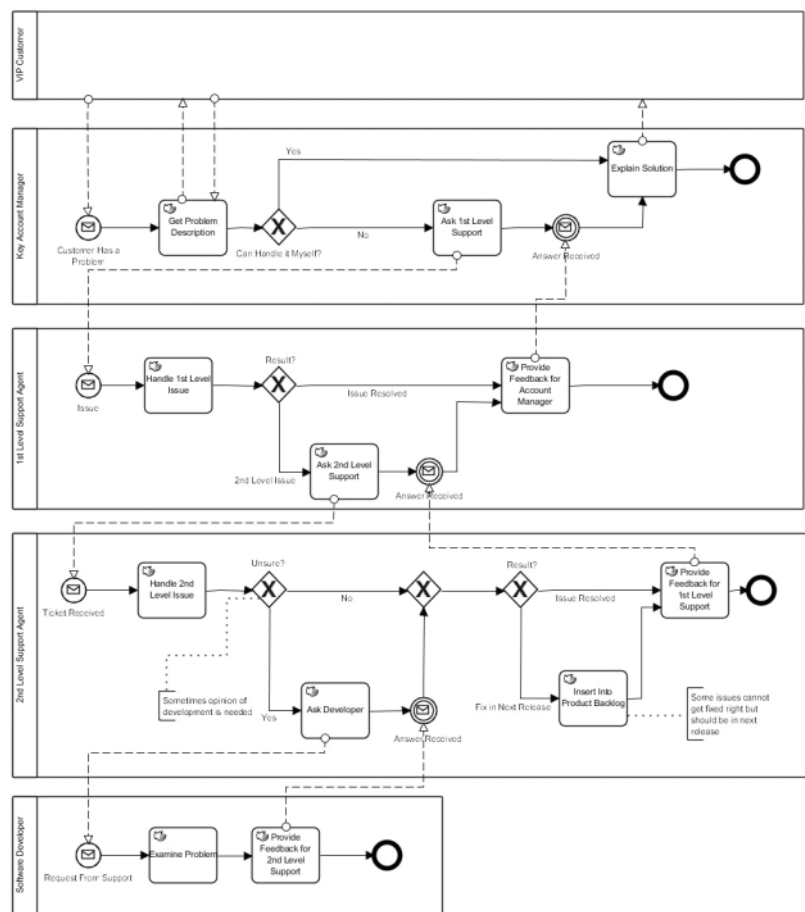
AMAZON FULLFILLMENT



Ci sono 3 pull: client amazon , compagnia di carta di credito e amazon stesso.

Il cliente cerca il prodotto, lo aggiunge al carrello e poi quando ha finito di fare shopping effettua il checkout, ossia un sottoprocesso che comunica con la compagnia della carta di credito. Vi è uno scambio di messaggi per l'esecuzione del pagamento e nel caso in cui si è accettato manda l'ordine ad amazon. Se il pagamento fallisce e non vuole riprovare vi è una terminazione per errore che viene catturata dal boundary event apposito che termina il processo. Se invece va a buon fine amazon gestisce l'ordine come in figura.

GESTIONE DI UN PROBLEMA TECNICO DI UN SW



Un cliente in base al problema telefona il centro assistenza. il ticket viene gestito su più livelli: l'azienda divide i ticket in base al livello tecnico. Il cliente manda una richiesta con un messaggio al quale viene chiesta una descrizione che il cliente deve dare. Se la soluzione è gestibile dal primo livello allora spiega la soluzione al cliente e termina. Se invece non è in grado si manda il ticket al secondo livello. Questi analizzano il problema e se lo risolvono mandano il feedback al primo livello e questo spiega la soluzione ottenuta al cliente. Se anche qui non vi è soluzione si manda al livello successivo. In questo punto si analizza il problema e se non si è certi della soluzione si manda un messaggio allo sviluppatore che poi dà una risposta e la soluzione sale fino al cliente. Se invece era necessario un fix viene effettuato e si manda la risposta al cliente.

Reti di Petri

martedì 30 novembre 2021 02:41

Bpmn ha una semantica precisa e rigorosa. Questa ci permette di vedere in molti casi che alcuni diagrammi non sono corretti in maniera automatica per quasi tutto. Un modo di valutare un processo è quello di eseguirlo, ma sarebbe una perdita di tempo. Si è pensato quindi alla simulazione, usando un modello: le reti di Petri.

Non tutti i grafici sono traducibili equivalentemente in reti di petri. Questo è dovuto a dei limiti di dettagli e elementi disponibili.

Le reti di Petri sono nate quando Petri cercava un modo semplice per descrivere i processi chimici in modo matematico, ma ha avuto successo principalmente in ambito di processi concorrenti. Questo perché la notazione è particolarmente adatto a questo ambito, e inoltre perché lo strumento è facile da implementare ed è efficiente.

Lo strumento poi è stato esteso con i colori per rendere ancora più chiaro il tutto.

Una rete di Petri è una quadrupla:

- **P** è un insieme *finito* di places e rappresenta uno stato, una possibile configurazione in cui si può trovare un sistema. Notazione: cerchio
- **T** è un insieme finito di transizioni, e P e T non hanno elementi in comune. La transizione rappresenta il passaggio da uno stato ad un altro a seguito di una azione. Notazione: quadrato
- **F** invece è l'insieme delle relazioni di flusso ed è un insieme delle possibili transizioni da un place a una transition e viceversa. Notazione: freccia
- Il peso **W**: data una transizione vi associa un numero positivo non nullo. Se è vuoto vuol dire 1 e sono associati agli archi.

Petri net

A Petri net is a tuple (P, T, F, W) , where:

- P is a finite set of places;
- T is a finite set of transitions, with $P \cap T = \emptyset$;
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs forming a flow relation;
- $W : F \rightarrow \mathbb{N} \setminus \{0\}$ is an (arc) weight function.

- Graphical notation: places = \bigcirc , transitions = \square , arcs = \rightarrow .
- Arc types:



La differenza con una macchina a stati finiti è l'esplicitazione della transizione, che qui ha un elemento vero e proprio, mentre nelle macchine a stati gli archi sono l'indicazione della transizione da uno stato all'altro. Con questo posso esprimere la condizione come ad esempio i processi concorrenti dove si arriva da due stati diversi. Non è naturale nelle macchine a stati rappresentare le istanze di esecuzione perché le esecuzioni sono analizzabili dalla lettura e il consumo dell'input, ma questo può portare al non determinismo. In reti di Petri si rappresentano sia thread concorrenti che più istanze del processo stesso.

Per comprendere il funzionamento serve una notazione: l'elemento di base per comprenderne il funzionamento è il **multiset**. È un insieme di elementi su un alfabeto che possono essere ripetuti. Quindi rappresentano un sottoinsieme dell'alfabeto con possibili ripetizioni. Queste ripetizioni possono essere esplicite (elementi effettivamente ripetuti) o compatte con la rappresentazione con potenza.

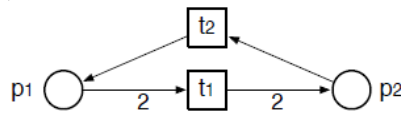
$$(a^{X(a)}): [a, a, a, b, c, b] = [a^3, b^2, c].$$

Posso semplificare ancora assumendo un ordinamento specifico dell'insieme ed esplicitando solamente le potenze.

Poiché sono array, le operazioni di calcolo si basano su questi array numerici e sono quindi operazioni tra matrici.

Un altro elemento importante sono i **postset/preset** e valgono sia per i place che per le transition. Si denota un puntino davanti al place o la transition e contano gli archi provenienti da una transition in entrata per un place nel caso del preset,

e contano gli archi uscenti verso una transition con il relativo peso per il postset di un place. La stessa cosa vale per le transition.



$$\bullet p_1 = [t_2]$$

$$p_1 \bullet = [t_1^2]$$

$$\bullet t_2 = [p_2]$$

$$t_2 \bullet = [p_1]$$

In questo esempio il preset di p1 è l'elenco degli archi entranti in esso, ossia t2. il postset di p1 è un arco di peso 2 verso t1, il cui peso si rappresenta con l'esponente.

La definizione formale è:

Il preset è l'insieme delle X con relativo peso, che possono essere transizioni nel caso dei place, dove il peso è definito e la coppia (x,p1) deve appartenere all'insieme delle relazioni di flusso possibili.

Petri net:

- $P = \{p_1, p_2\}$
- $T = \{t_1, t_2\}$
- $F = \{(p_1, t_1), (t_1, p_2), (p_2, t_2), (t_2, p_1)\}$
- $W = \{((p_1, t_1), 2), ((t_1, p_2), 2), ((p_2, t_2), 1), ((t_2, p_1), 1)\}$

Preset:

- $\bullet p_1 = [x^{W(x,p_1)} \mid W(x,p_1) \text{ is defined and } (x,p_1) \in F]$, that is
- $\bullet p_1 = [x^{W(x,p_1)} \mid x \in \{t_2\}]$, thus $\bullet p_1 = [t_2^1]$, i.e. $\bullet p_1 = [t_2]$, since $W(t_2, p_1) = 1$, i.e. $[t_2^1]$

Il postset è l'insieme delle y tali che il peso (p1,y) è definito e appartiene all'insieme dei flussi.

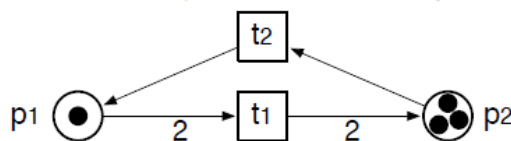
- $p_1 \bullet = [y^{W(p_1,y)} \mid W(p_1,y) \text{ is defined and } (p_1,y) \in F]$, that is
- $p_1 \bullet = [y^{W(p_1,y)} \mid y \in \{t_1\}]$, thus $p_1 \bullet = [t_1^2]$, since $W(p_1, t_1) = 2$, i.e. $[t_1^2]$

Il marking è un insieme di token distribuiti nella rete di Petri. Intuitivamente rappresentano istanze di processi, ossia vuol dire che se la reti di petri descrive il processo, i pallini rappresentano le varie istanze attive del processo e vengono rappresentati negli stati in cui si trovano. Si denotano elencando i place in cui si trovano i token e quanti token si trovano in ciascun place con l'esponente.

Marking

A marking M of a Petri net (P, T, F, W) is a multi-set over P : $M \in \mathbb{B}(P)$.

The marking identifies how many tokens are currently present in each place of the net.



$$M_0 = [p_1^1, p_2^3].$$

$$\text{Vector notation: } M_0 = (1, 3).$$

La firing rule permette di stabilire come trasformare un marking in un altro, ossia come avanzano le istanze di processo in una rete di Petri. Permette di computare un passo. In un sistema concorrente rappresentai token distribuiti nei vari stati, che rappresentano la computazione per una istanza e il token rappresenta dove è arrivata la computazione per quell'istanza.

In un momento qualsiasi potrebbero essere possibili più computazioni, ossia istanze che possono progredire. Abbiamo un sistema concorrente in cui più token possono progredire, ossia attraversare una transizione e quindi passare da uno stato a un altro. La regola di firing stabilisce quali, se ci sono, delle transizioni che possono essere eseguite e quale è il nuovo marking risultante. Determinare il nuovo marking è la situazione dei token e il loro stato a seguito di una computazione.

Data una rete di Petri e un marking M possibile, sia t una trasizione abilitata all'iterno della rete e denotiamo quindi con $(N, M[t >]$ il firing della transizione t .

Firing rule

Given a Petri net $N = (P, T, F, W)$ and a marking $M \in \mathbb{B}(P)$:

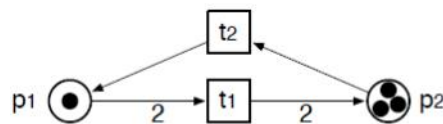
- a transition $t \in T$ is **enabled**, denoted $(N, M)[t]$, if and only if $M \geq \bullet t$;
- an enabled transition $t \in T$ can fire leading to marking $M' \in \mathbb{B}(P)$, denoted $(N, M)[t](N, M')$, if and only if $M' = (M - \bullet t) + t \bullet$.

Abilitata significa che il marking è \geq del preset di quella transizione. Il marking ci dice quali sono le distribuzioni di token nei place e quindi se assumiamo che elenca tutti i place in modo ordinato, ci basta mettere l'esponente. Il preset di t è il peso degli archi che entrano in t e arrivano da un place. Il \geq è valutato componente per componente.

La regola ci dice che se ho una transizione t e un arco con peso ad es 2 ci devono essere almeno due di token nel place entrante per attivare t . Se vi è un altro arco entrante da un place, con peso ad es 1 deve avere almeno un token. Prendi una transizione, guarda i pesi degli archi entranti, nei place ci devono essere almeno altrettanti token altrimenti non è eseguibile.

Per avere il nuovo marking sequi la regola di sopra (evidenziata), cioè al marking attuale togli il preset e ci aggiungo il postset.

Es :



$$M_0 = [p_1^1, p_2^3].$$

$$\text{Vector notation: } M_0 = (1, 3).$$

We have that:

- $\bullet t_1 = [p_1^2] = [p_1^2, p_2^0] = (2, 0)$
- $\bullet t_2 = [p_2] = [p_1^0, p_2^1] = (0, 1)$

per ogni transizione elenco i place e metto il peso dell'arco entrante e se non c'è metto 0

Thus: seguendo la regola

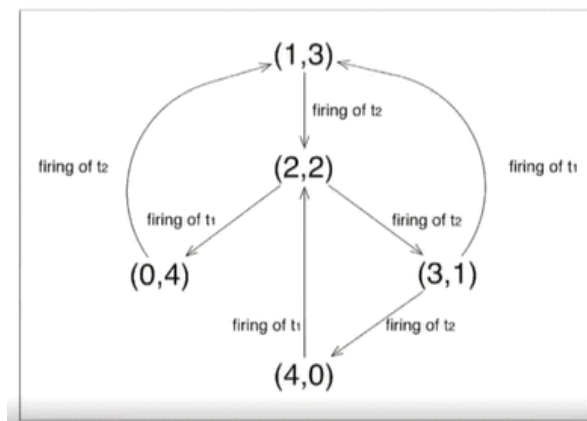
- $M_0 \not\geq \bullet t_1$, i.e. $(1, 3) \not\geq (2, 0)$, t_1 not enabled non è maggiore perchè 1 non è maggiore di 2, basta un po' per dirlo
- $M_0 \geq \bullet t_2$, i.e. $(1, 3) \geq (0, 1)$, t_2 enabled

Successivamente si applica la regola di firing. Succede che si consumano tanti token quanto è il peso che porta quelle transizioni e se ne generano tanti quanti dice la freccia uscente.

Nell'esempio otteniamo quindi:

$$M_0 - \bullet t_2 = (1 - 0, 3 - 1) = (1, 2), M_1 = M_0 - \bullet t_2 + t_2 \bullet = (1, 2) + (1, 0) = (2, 2)$$

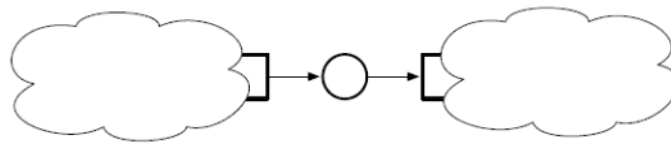
Andando a definire tutte le possibili transizioni e i marking ottenibili graficamente si ha il grafo di raggiungibilità.



Il modo naturale per concepire i place sono gli stati, che sono i punti intermedi tra le attività, mentre le transizioni possiamo identificarle come le nostre attività atomiche, o gli eventi. I token sono gli oggetti manipolati dagli oggetti, le istanze. Il marking è una snapshot dell'istanza di un processo. Il marking iniziale è lo stato iniziale dell'istanza quando parte il processo. Il firing è l'esecuzione di un passo del processo e il grafo di raggiungibilità rappresenta tutte le possibili

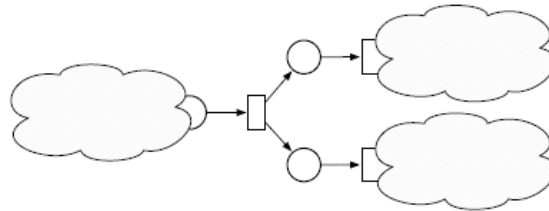
esecuzioni del processo. (vedi anche slide 15)

Il flusso, ossia la freccia che unisce due BPMN è data da un pallino, quindi uno stato. Deve terminare con una transizione e finire con una transizione necessariamente.

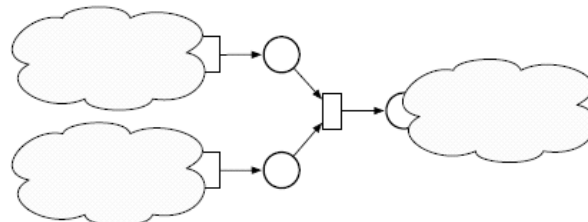


L'AND split è così rappresentato:

And-split



And-join

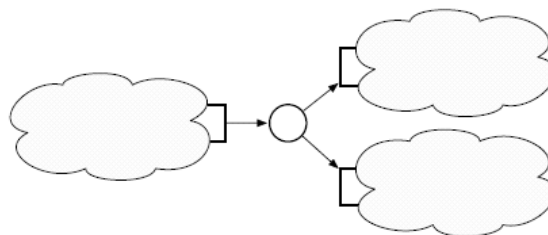


Se ho una istanza e una transizione, richiede un token per essere attivata. La transizione poi consuma questo token e ne produce altrettanti nei place in cui esce. l'accortezza è che entro con un place ed esco con una transizione.

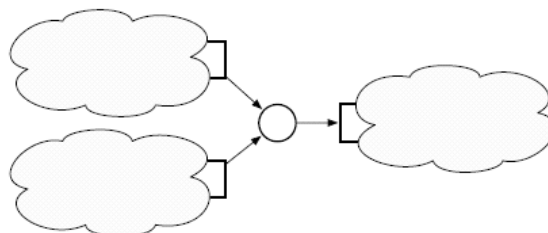
L'AND join: se ho un solo token entrante nella transazione, quando ce ne sono ad es 2 di archi in esso, la transizione in questione non è attiva finchè non arrivano ad essa entrambi i token. La transizione va poi ad unire l'esecuzione con un nuovo marking. Serve un token in ogni place tanti quanti sono i pesi degli archi entranti. In questo caso al contrario entro con transizioni ed esco con un place unico.

Lo XOR join e lo XOR split sono così definiti:

Xor-split



Xor-join



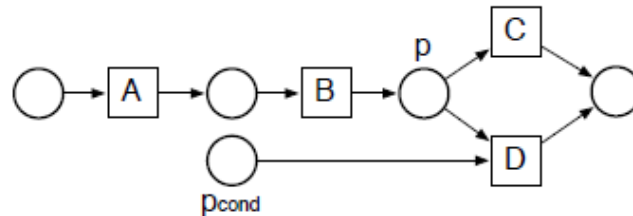
Quando una transizione si abilita e scatta, produce un token che a questo punto può seguire una delle due transizioni. La transizione che scelgo consuma il token presente nello stato, di conseguenza non è abilitato più lo stato da cui parte la diramazione perchè solo in una delle due transazioni viene consumato. In signavio sono abilitate entrambe le scelte, quando si sceglie una strada l'altra non scelta viene disattivata perchè non è più attraversabile.

Il join invece ha due transizioni entranti in uno stato e dunque quella finale di rcongiungimento è abilitata indipendentemente da quale delle due transizioni arrivi il token.

Si possono anche realizzare i cicli: se deve essere eseguito almeno una volta si mette uno stato a fine passaggio che permette di proseguire o di tornare indietro e rieseguire il passaggio; oppure anche zero volte dove il ciclo passa per lo stato e quindi si può proseguire direttamente o si entra nel ciclo.

FREE CHOICE NETS

È una rete particolare di Petri.



In questo esempio sembra esserci uno XOR split, ma p è condizionata. In questa situazione non abbiamo una libera scelta nello XOR. Questo significa che non tutte le reti di Petri si possono tradurre in PBMN, ma anzi aggiungono dettagli. Quindi non sempre le reti di Petri corrispondono a BPMN.

I workflow net sono dei particolari tipi diretti di Petri che sono effettivamente un workflow BPMN e lo sono quando ci sono due place speciali: input place (p_i) e un output place (p_o). Se unisco in modo fittizio questi due con una transizione t^* e se per ogni coppia di nodi che io prendo esiste un cammino all'interno della rete per andare da uno all'altro in modo che si attraversi sempre correttamente transizione-place, allora è un workflow associabile a un BPMN. Esso è detto fortemente connesso ed è indicato con N segnato.

Il workflow net sono stati studiati per le proprietà che possono eseguire.

Se esiste un k t.c. che per ogni sequenza di firing dal marking, questa sequenza di firing è $<k$ ed è detta terminante

È deadlock free se per ogni marking raggiungibile dal marking considerato, esiste una possibile transizione che permette di raggiungerlo.

È k -bounded quando per ogni marking raggiungibile da un altro, il numero di token in un place è inferiore a k , quindi non si accumulano troppi token in un place. La rete è k -boundend se per ogni place in N si ha questa regola.

È safe se è 1-bounded, se ogni place al più ha un token.

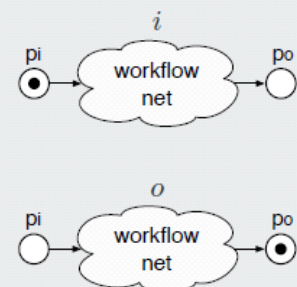
È live se per ogni marking raggiungibile, esiste un altro marking raggiungibile da M' tale che ci sia una transizione verso M'' .

Per ogni marking si può sempre andare avanti. È vivo in tutto se è vivo per ogni possibile transizione in N .

Input/output state

Given a workflow net N :

- The **input state** i is a marking that assigns only one token to the input place p_i of N .
- The **output state** o is a marking that assigns only one token to the output place p_o of N .



Un workflow net è **corretto(sound)** se e solo se è deadlock free, se dal marking iniziale l'output è sempre raggiungibile per ogni marking raggiungibile da quello iniziale. Quindi in qualsiasi situazione riesco a raggiungere quello finale senza blocchi. Quando raggiungo quello finale questo è pulito, ossia non ci sono altri token da altre parti.

Ci sono dei teoremi associati:

Theorem (van der Aalst, 1997)

A workflow net N is sound if and only if \overline{N} is live and bounded.

Theorem (van der Aalst, 1997)

For a free-choice workflow net it is possible to decide soundness in polynomial time.

N segnato è quello con la transazione t^* . Quindi si riduce la correttezza alla liveness e al bound, in tempo polinomiale. Questi sono più semplici da verificare perchè sono su transizioni, a differenza del deadlock che occorre l'analisi dei percorsi.

Per decidere se un grafo è corretto ha bisogno del grafo di raggiungibilità, che rappresenta tutti i marking e posso vedere liveness e bound. Per andarlo a vedere dovrei andare a costruirlo.

Esso è descritto così:

Construction algorithm

Given a Petri net N and an initial marking M_0 :

1. Label M_0 as the *root* and initialize set $New = \{M_0\}$.
2. While $New \neq \emptyset$:
 - 2.1 Select marking M from New .
 - 2.2 While there exists an enabled transition t at M :
 - 2.2.1 Obtain the marking M' that results from firing t at M .
 - 2.2.2 If M' does not appear in the graph add it to the graph and insert M' into set New .
 - 2.2.3 Draw an arc with label t between M and M' .
 - 2.3 Remove M from New .

Il problema è che non sempre questo algoritmo termina. Ci sono casi in cui non si abilitano transizioni, oppure che vanno all'infinito.

Unboundeness and coverability graph

Quando un marking è uguale ad un altro marking ma con +1 in un place, lo si mette a omega, ossia un nuovo simbolo e serve per dire "andare avanti è inutile perchè si accumulano i token". Riconosce quando incrementa sempre e sostituisce questo con un ciclo che porta ad omega. Quello che si ottiene non è un grafo di raggiungibilità, ma è un grafo di copertura, che si riesce a determinare in tempo polinomiale e finito perchè si tratta di una approssimazione. Questo algoritmo termina sempre ed è polinomiale. I due grafi non sono la stessa cosa ma, se quello di raggiungibilità è finito, coincide con quello di copertura.

Il grafo di copertura può essere usato per determinare se quello di raggiungibilità è bounded o no. In questo modo si dimostra se un grafo è corretto o no. Nel caso in cui risultasse bounded, la verifica della liveness è fattibile con il grafo di raggiungibilità.

In pratica invece di usare il grafo di raggiungibilità ne costruisce uno di copertura e i due sono equivalenti quando nella costruzione non si utilizza omega. Se è bounded si percorre il grafo e cerca gli omega in tempo polinomiale. Se non è bounded lo dice subito, se invece lo è allora anche i due grafi coincidono e quindi cerca nel grafo che ha già costruito se è live, anche questo fattibile in tempo polinomiale.

Business Process Management II

martedì 7 dicembre 2021 11:43

Costruire e nalizzare un modelo di processo è un progetto articolato e complesso. Il processo ha bisogno di tecniche (metodologie), strumenti descrittivi (linguaggi) e di supporto (tool).

oltre ad identificare eventi e attività, è importante capire quali sono i partecipanti al processo che vengono specificati in base ai ruoli che giocano. Questi sono considerati come risorse così come lo sono anche gli strumenti utilizzati.

Oltre alle attività, si devono definire i tempi di svolgimento delle attività. Ogni attività è stata studiata e definita una distribuzione di probabilità delle tempistiche impiegate. In questa fase di raccolta si devono effettuare delle osservazioni sulle esecuzioni delle attività.

Per le simulazioni si usano i generatori casuali di eventi che descrivono le transazioni in ingresso al processo

Sviluppo di competenze in ambito della modellazione dei dati e della conoscenza , nell'analisi dei flussi informativi e dei processi decisioni, nell'apprendimento automatico e nella risoluzione automatica di problemi.

PROGETTO

sabato 5 febbraio 2022 17:31

Sono coinvolti sicuramente i clienti, che hanno o meno un account e per poter effettuare un acquisto devono per forza essere in possesso di un account e può essere fatto anche al momento dell'acquisto.

Per effettuare l'acquisto c'è un limite di tempo.

Il sistema ha una selezione di artisti e di tour associati all'artista. Per ogni artista ha segnate le date dei concerti e le località. Ogni località in cui l'artista effettua il concerto ha una sezione di acquisto dei biglietti a se stante.

Per ogni data poi ha il numero di settori, il codice dei settori, il prezzo per ogni settore e ciascun settore ha dei posti specifici con indicazione se sono liberi o meno. Ogni posto in base al settore di appartenenza ha un prezzo diverso. I biglietti sono a nominativo e se si deve effettuare la modifica c'è una tariffa da pagare.

Quando si effettua l'acquisto si deve quindi selezionare l'artista, si deve selezionare il tour (solitamente è solo uno per volta) e si deve selezionare la data con località associata (per ogni data c'è una località, ma la stessa località può essere associata a più date). L'acquisto dei biglietti è disponibile dalla data indicata dall'artista fino ad esaurimento biglietti. Ci possono essere anche i casi in cui vi è prevendita e poi vendita. Nel caso della prevendita solo la metà dei biglietti vengono resi disponibili. Per ogni data ci sono anche i pacchetti vip, che hanno tutta una serie di agevolazioni e cose in più, tra cui posti riservati con visibilità privilegiata che non sono disponibili per il resto della vendita (tipo posti nelle prime file di un determinato anello o parterre). Per ogni pacchetto, perché può essere più di uno vi è la lista dei privilegi. Questa selezione avviene con una sezione secondaria per la stessa data, e non tra le vendite generali.

Quando è aperta la vendita il cliente, anche senza account, può iniziare la procedura di selezione della data, del settore e del posto nella data. Ci sono due modalità di scelta del posto, o il migliore disponibile ed è selezionato dal sistema in base a quelli disponibili senza limitazione di prezzo (vedere nel caso se si può inserire la questione dell'assegnazione casuale del posto andando a definire i vari settori che si stanno considerando per restringere il range di biglietti tra cui assegnare), oppure quella della scelta su mappa dove sulla mappa vengono indicati i posti liberi e quelli occupati e il cliente cliccando sul posto desiderato può selezionare il/i posti desiderati. Ciascun cliente può acquistare un massimo di 6 biglietti per volta. Se si acquista in fase di prevendita si ha un costo aggiuntivo specifico che vale per chiunque acquisti i biglietti in quella data indipendentemente dal posto scelto, mentre in fase di vendita generale questo costo non viene addebitato. Una volta effettuata la selezione del posto si passa alla scelta della tipologia di biglietto: vi è l'e-ticket che alla fine di tutto il processo produce un pdf con codice a barre da stampare a casa, oppure vi è il biglietto fisico che invece viene spedito. Se si sceglie l'e-ticket non ci sono costi aggiuntivi, per la spedizione invece sì e ci sono varie possibilità di scelta tra i corrieri e i tempi di invio etc, questi dipendono dal corriere specifico che al momento della scelta viene informato della futura spedizione (vedere effettivamente se si deve fare qui o in seguito si deve dare l'informazione). Si deve inserire l'indirizzo di fatturazione in ogni caso e nel caso del biglietto spedito si deve anche indicare l'indirizzo. Anche i prezzi variano indipendentemente dalla scelta. Con l'aggiunta di queste agevolazioni il totale viene aggiornato in tempo reale. In seguito vi è anche la scelta dell'assicurazione, che è facoltativa. Se si sceglie l'assicurazione ci sono una serie di agevolazioni tipo cambio del biglietto possibile, così come rimborso in caso di problemi che impediscano la presenza del cliente all'evento o protezione in caso di smarrimento. Se si mette l'assicurazione il sistema attiva di conseguenza una serie di cose che poi il cliente al termine dell'acquisto può effettuare fino a 30 giorni dalla data dell'evento. Dopo la scelta dell'assicurazione si inseriscono i nominativi dei partecipanti all'evento. Non tutti gli eventi sono a nominativo e questo permette di definire se poi è reso possibile il cambio di nominativo per il determinato evento (tramite pagamento supplementare) o se non è necessario.

Successivamente si passa alla fase di pagamento: il sistema permette di pagare con carta di vari circuiti che indicherà. Il cliente dovrà inserire i dati della carta e poi confermare. Al momento della conferma viene rimandato alla pagina dell'ente bancario di riferimento che si occuperà di chiedere conferma dell'acquisto al cliente e una volta effettuati tutti i procedimenti dello specifico sistema informerà il sistema della presa a carico del pagamento a nome del cliente e rimanderà sulla pagina dell'acquisto. Se l'acquisto con la banca è andato a buon fine il sistema avviserà il cliente della corretta riuscita del pagamento e invierà un riepilogo dell'acquisto sia in tempo reale che via mail, sia rendendo visibile sulla pagina personale del cliente l'evento e il suo acquisto. Tutto questo procedimento, dalla scelta del posto alla ricezione della conferma di avvenuto pagamento, sono con limite di tempo (solitamente 15 minuti) al cui scadere viene resettato tutto e si torna alla selezione della località e della data, annullando tutte le scelte fatte fino al momento specifico. Se ci sono problemi con le transazioni il cliente può riprovare l'inserimento delle carte etc fino alla fine del tempo.

Sulla pagina del cliente sarà quindi poi possibile, in base alle selezioni effettuate, avere a disposizione i privilegi dell'assicurazione e quello del cambio di nominativo.

Se si sceglie il pacchetto vip e questo prevede dei gadget in aggiunta al biglietto e altro si deve gestire anche la questione della spedizione di essi insieme al biglietto, quindi il flusso del biglietto VIP deve essere gestito diversamente in alcune parti.

Se si cambia nominativo chiaramente il processo è controllato e può essere effettuato con un pagamento aggiuntivo a carico del cliente, per il rimborso viene emesso sul conto bancario indicato per effettuare l'acquisto iniziale.

La scelta dei biglietti è bloccata prima dell'effettiva apertura delle vendite e se vi è molta richiesta nello stesso momento il cliente viene inserito in una coda virtuale al termine della quale potrà eseguire il procedimento. In caso di sold out i biglietti non potranno essere selezionati quindi il procedimento dalla scelta in poi è bloccato.

Person(ruolo cliente) ha Account
Account ha Username e Password
Username è univoco per ciascun Account
Username e password hanno il vincolo di uguaglianza
Se non ha Account non può acquistare Biglietto
Account ha MetodoDiPagamento (obbligatorio)
MetodoDiPagamento ha Intestatario e NumeroConto che lo identificano inoltre
MetodoDiPagamento è gestito da Banca
Cliente possiede Biglietto (min 0, max 6)
Cliente acquista Biglietto con MetodoDiPagamento (solo uno tra quelli presenti nell'Account)
mettere la relazione Cliente ha metododipagamento e poi mettere la terna cliente compra biglietto con metodo di pagamento e poi metter evincolo di sottoinsieme tra cliente/pagamento e cliente/biglietto
Biglietto ha nominativo (mettere nome come value)
Biglietto si riferisce a Concerto
Concerto ha data
Concerto è di Person(ruolo Artista)
Concerto ha Luogo
Concerto è oggettificazione di Artista, Data, Luogo(?) NO
Il Concerto per quell'Artista in quella Data in quel Luogo devono essere univoci
Per una Data e un Artista c'è solo un Concerto in un determinato Luogo
Per uno stesso Luogo possono esserci più Date dello stesso Artista
Per un Luogo in una Data può esserci un solo Artista

Owner della carta è una Person?

cliente acquista biglietto con metodo di pagamento e quindi client-Ticket non sono in relazione tra di loro e basta oppure ticket è stato acquistato con metodo di pagamento?

Togliere metodo di pagamento?

Cliente
Account (ossia username o password)
Biglietto (max 6 a cliente) e ha nominativo
Biglietto ha assicurazione
Tipo di biglietto (e-ticket o fisico)
Artista
Tour(?)
Concerto con data e luogo
Luogo ha settori
Settore ha posto e posto ha prezzo
Il posto ha uno stato: libero o occupato (relazione unaria)
Pacchetto vip / tipo di biglietto (?)
Corriere(?) che è legato solo al tipo di biglietto fisico
Costo spedizione legato al corriere
Banca (legata al cliente) (?) oppure cliente ha conto con codice ecc ed è di una banca specifica
Potrei mettere collegato a ciascun account più di un metodo di pagamento e posso mettere che in fase di pagamento del biglietto debbo necessariamente sceglierne solo uno per pagare (quindi mettere il vincolo esterno di exclusive or)
Si può acquistare il biglietto solo se esiste un metodo di pagamento e il metodo di pagamento si può inserire al

Concerto è oggettificazione di Artista, Data, Luogo (1,1,1)

Il Concerto per quell'Artista in quella Data in quel Luogo devono essere univoci

Per una Data e un Artista c'è solo un Concerto in un determinato Luogo

Per uno stesso Luogo possono esserci più Date dello stesso Artista

Per un Luogo in una Data può esserci un solo Artista

Luogo ha Settore (>1)

Settore ha più Posti

Biglietto si riferisce a Posto (uno per ogni Biglietto)

Posto fa parte di Settore

Posto può essere inSpalti o InParterre

Se è inParterre non ha indicazione del posto a sedere

Settore ha prezzo

Settore ha più Posti, Posto ha un solo Settore

Posto è libero o occupato (relazione unaria booleana)

Posto può essere acquistato solo se è libero

Biglietto è di Tipo e-ticketo fisico

Se Biglietto è fisico ha un Corriere associato per la spedizione(costo spedizione?)

Corriere ha costi di spedizione di Prezzo

BigliettoFisico sottotipo di Biglietto

Biglietto ha assicurazione(unaria)

Togliere
metodo di
pagamento?

Devo mettere
l'account
associato ad
una persona o
l'account non
serve come
entità?

Biglietto deve necessariamente designare una Data per
pagare(quindi mettere il vincolo esterno di exclusive or)

Si può acquistare il biglietto solo se esiste un metodo di
pagamento e il metodo di pagamento si può inserire al
momento dell'acquisto o può essere preesistente
sull'account(se si mette il metodo obbligatorio sull'account è
ovvio che non avendo l'account non si può acquistare il
biglietto e nell'account è obbligatorio mettere un metodo di
pagamento)

Potrei mettere la cosa dinamica che i dati del biglietto
possono essere presenti per il cliente solo se è avvenuto il
biglietto (tipo relazione Cliente ha biglietto per Concerto)

Concerto è una oggettificazione di artista con data e luogo !!

- MODELLAZIONE CONCETTUALE

La modellazione concettuale ha lo scopo di modellare il dominio e descrivere formalmente, tramite un linguaggio preciso, aspetti fisici e sociali della realtà ai fini di un certo obiettivo. Deve essere fatto in modo non ambiguo e quindi non si può usare il linguaggio naturale. Il modello è un'astrazione della realtà secondo una determinata concettualizzazione. Lo scopo principale della modellazione concettuale è definire degli information systems che siano di supporto al lavoro degli umani, così come serve a capire la comunicazione tra gli umani. Ciò che viene modellato sono i dati, ossia dei fatti relativi a un particolare stato del dominio studiato, e i processi che sono collezioni di task che descrivono come il lavoro è strutturato e come vengono manipolati i dati. Un altro aspetto importante sono le risorse che vengono utilizzate dagli attori per modificare ed evolvere i dati.

- DEF DI CONCETTUALIZZAZIONE

Il concetto è un'idea o astrazione derivante da esempi o istanze della realtà e di cui si estraggono le proprietà e il significato a seguito di una conoscenza acquistata dall'analisi dei suddetti esempi (capisco che quello è un cane perchè ho tanti esempi di cani e associo le caratteristiche comuni per poterli identificare e generalizzare). Il concetto è una rappresentazione intensionale della realtà , cioè esprimo le caratteristiche rilevanti per il contesto astrando dall'elemento reale. Una concettualizzazione è un processo di astrazione e generalizzazione da esperienza e percezioni su quello che è il dominio in analisi: si tratta di una parte di realtà rilevante come percepita e organizzata da un agente estraendo da una specifica situazione in base al contesto specifico usando il linguaggio formale adeguato. Il linguaggio usato per la concettualizzazione ha delle interpretazioni e queste devono coincidere con il significato delle cose per poter considerare la concettualizzazione corretta in modo tale che il modello che si costruisce a partire dalla concettualizzazione sia per lo meno completamente incluso da quello inteso.

- DEF DI ONTOLOGIA

Un'ontologia è lo studio filosofico della natura e della struttura degli esseri e della realtà, delle loro categorie e delle relazioni tra loro. Si dice modellazione ontologica perchè modella la realtà. Con i concetti possiamo avere influenze nella definizione date dall'agente, invece l'ontologia è neutra e non ha quest'influenza.

- INFORMATION SYSTEM E SUE FUNZIONI!!

Un sistema informativo è un sistema che modella, colleziona, immagazzina, processa e distribuisce informazioni sullo stato del dominio considerato per facilitare la pianificazione, il controllo e la coordinazione di un processo decisionale in una organizzazione. Si riferisce allo stato del dominio.

Le funzioni dell'IS sono:

*Memoria: mantenere una rappresentazione dello stato del dominio andando a memorizzare sia le istanze di dati che la loro semantica. Le informazioni sono aggiornate su richiesta dell'utente o autonomamente

*Informazioni: fornire informazioni sullo stato del dominio tramite interrogazioni o fornendole automaticamente

*Attività: eseguire azioni che cambiano lo stato del mondo e permette all'IS di agire sia automaticamente che sotto richiesta dell'utente

È importante per un sistema informativo avere conoscenza del dominio e delle funzioni che deve poter eseguire. Inoltre il dominio evolve nel tempo quindi lo schema concettuale che rappresenta le funzioni deve comprendere aspetti sia statici che dinamici.

- QUERY INTENSIONALI ED ESTENSIONALI

Le query vengono poste al sistema per avere informazioni sullo stato di esso. Deve essere definito un linguaggio comune per poter essere compreso sia dal sistema che dall'utente che ne fa uso. Possono appunto essere di due tipi:

*Intensionali: il sistema risponde con delle proprietà associate ai dati di cui si chiedono informazioni

*Estensionali: il sistema restituisce le istanze di dati che corrispondono alla richiesta

- *DEF VINCOLI E TIPOLOGIE*

- DIFF TRA UML, E-R E ORM

- PUNTI DEL CSDP

- RELAZIONE N-ARIA DEF E PERCHÈ CONTROLLO SULLA CHIAVE MOTIVI

- CRITERI PER CAPIRE SE UNA RELAZIONE È PRIMITIVA O MENO

- DEF FATTI PRIMITIVI E IMPORTANZA

I fatti primitivi sono unità atomiche di informazioni, ossia non sono scomponibili ulteriormente e di cui non vi è possibile applicare modifica in solo parte di essi. Il fatto elementare è un'asserzione che coinvolge uno o più oggetti che sono in relazione tra loro e ciascuno di essi gioca un ruolo imprescindibile nella relazione.

- VINCOLI TRA RUOLI GIOCATI DALLO STESSO ENTITY TYPE

- OBJECT TYPE DIPENDENTE E INDIPENDENTE

Si può dire indipendente quando tutti i ruoli che gioca non sono mandatory ed è rappresentato dal !.

- VINCOLI AD ANELLO

- COME GESTIRE I SOTTOTIPI IN RMAP E VANTAGGI/SVANTAGGI

- ALGORITMO GENERALE RMAP E VARIANTI

- *APPROCCIO SISTEMICO*

-

- DEF PROCESSO AZIENDALE

Un Business Process consiste in un insieme di attività eseguite in maniera coordinata in un ambiente organizzativo. Quando modelliamo un processo modelliamo un goal da raggiungere: ossia come voglio raggiungere un certo stato e quindi la definizione dell'obiettivo e le regole che devono essere seguite.

- FASI DI VITA DI UN PROCESSO/CICLO DI VITA DEL BPM

- PILASTRI DEL BPMN

- OBIETTIVI DEL BUSINESS PROCESS

- DIFF TRA MODELLO STRATEGICO, ORGANIZZATIVO E FUNZIONALE

Sono strumenti utili per definire le caratteristiche dell'azienda. Strategico serve a definire i fini gli obiettivi e i mezzi per raggiungere determinati obiettivi descritti nel BMM Business Motivation Model che permette di definire la strategia aziendale.

- RELAZIONE TRA PROCESSI E ORGANIZZAZIONE FUNZIONALE

- CATENA DEL VALORE DI PORTER

- COME SI TERMINA UN PROCESSO

- VARIE CATEGORIE DI EVENTI

- MULTIPLE START EVENT E GLI ALTRI TIPI

- EVENTO INTERMEDIO BORDER E INTERNO

- TIPI DI EVENTI INTERMEDI

- EVEN DRIVEN CHOICE

- SESE BLOCK

- DEF RETI DI PETRI

- REGOLA DI FIRING E QUANDO UNA TRANSIZIONE PUO' ESSERE ESEGUITA

- REGOLA DI FIRE CON OMEGA

- DEF MARKING

- ALGORITMO GRAFO DI RAGGIUNGIBILITA'

Lo stato è un certo marking della rete. Si parte da un initial marking...

- DIFF TRA GRAFO DI COPERTURA E GRAFO DI RAGGIUNGIBILITA'
 Approssimazione non deterministica del grafo di ragg in quanto nelle reti di petri c'è il problema della boundness e i grafo di raggiungibilità in una rete di petri infinita non termina, in quello di coeprtura aggiungiamo omega dove verranno messi i token che si accumulano. Sono equivalenti quello di copertura e di raggiungibilità solo se è bound.
- Quali vincoli coinvolgono ruoli diversi giocati dallo stesso object type, su relazioni diverse?
 Vincoli insiemistici (sottoinsieme, esclusione, exclusive or, uguaglianza)
- Che vincoli posso mettere tra due ruoli di ot diversi che sono due ruoli in una relazione con ruolo giocato da uno stesso object type?
 Vincolo di unicità esterno o di frequenza
- Teorema di wardlaffjd e quindi roba di bound e live e cosa sono i workflow net:
 Hanno start e end place. Una volta raggiunto l'end tutti gli altri token non devono essere attivi. l'end place deve essere sempre raggiungibile dallo start place. Live è una rete di petri in cui tutte le transizioni sono live e quindi tutte sono raggiungibili da un marking.