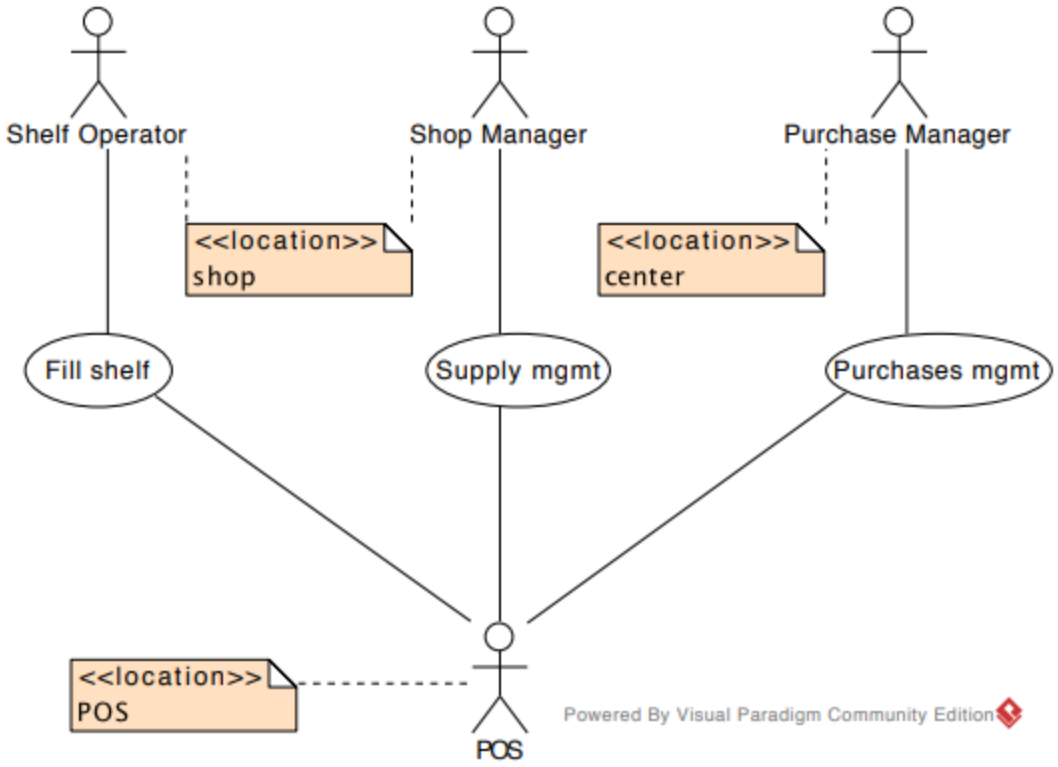
Lezione 5 17/10/2024

Where

Pensiamo che una **posizione** può essere sia **fisica** (coordinate xy rispetto a qualcosa, latitudine longitudine) oppure **virtuale** (spazio logico, organizzativo, per esempio in un organigramma la posizione delle persone).



Qui sto dicendo che lo shelf operator e shop manager sono nella posizione “shop”, però non mi interessa sapere in questo punto dove si trova questa posizione a livello di latitudine longitudine. Mentre invece il purchase manager si trova in una posizione centrale. Il POS può anche essere chiamato shop qua.

La cosa importante è che se c’è uno scambio di informazioni tra le attività svolte, mi devo rendere conto che tipi di collegamenti devo mettere in piedi.

Esempio: se ho il POS che sta nello shop, vediamo che il POS entra in gioco con l’use case purchases mgmt, il che significa che quando viene venduto un prodotto, viene innescata quest’attività. è probabile che quest’attività sarà

deployata su un hardware che è dove sta il purchase manager. Quindi ci deve essere una comunicazione tra il software del POS e il software del purchases mgmt.

Quindi ci interessa la locazione perchè ci può dare un'idea di come partizionare al meglio le mie componenti del sistema, per ottimizzare le qualità che per me sono importanti.

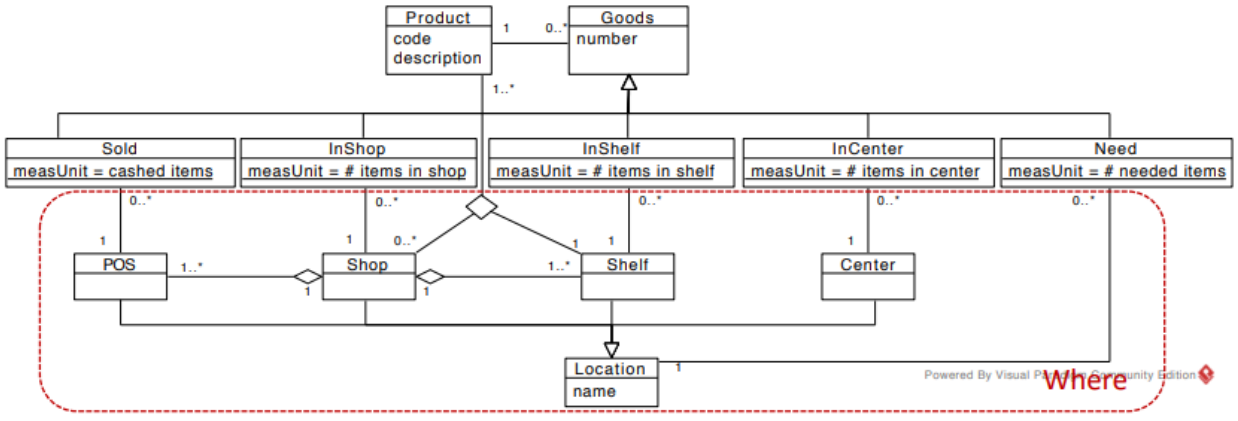
Io voglio che l'informazione della vendita arrivi con sicurezza, senza che il dato venga perso. Infatti nel "what" avevamo precision "strong".

Non mi interessa sapere la posizione di un prodotto nello shop in questo caso.

Per esempio in un sistema che rileva una caduta dai dati di accelerometri, non avrò una classe accelerometro in java ma avrò una classe "misura" che riceve i dati. Se però invece ricevo i dati dallo smartphone e da uno smartwatch, potrei volere distinguerli per gestirli in modo diverso (movimenti diversi) e quindi in questo caso, oltre ad avere l'accelerometro nel modello di dominio, sarà anche nel modello dei dati.

Tornando alla nostra immagine, tutti questi elementi non ce li ho all'interno del sistema, non mi interessa la posizione, mi interessa solo per ragionare sulla distribuzione delle funzionalità in componenti e poi sul deployment dei componenti sui dispositivi hardware.

Ci sono componenti dove la posizione è importante, tipo tripadvisor, deve sapere la posizione dell'utente.



Nel nostro caso questa modellazione non è necessaria perchè i nomi "sold, inshop, ..." già esplicitano la posizione dei prodotti. però se dovessi modellarlo,

vediamo che lo spazio non è inteso come coordinate xyz. Questo mi serve per capire come distribuire le responsabilità, la posizione poi non è usata all'interno del sistema.

Il collegamento tra need e location è un po' improprio.

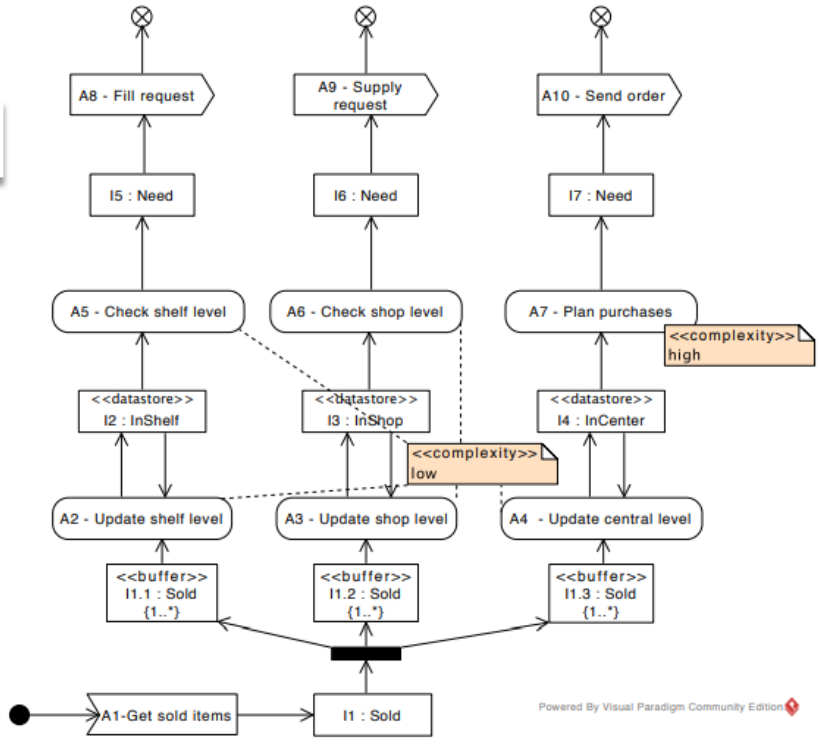
Riassumendo:

- Strutture dati che mi descrivono com'è fatto uno spazio e le locazioni al suo interno per poi localizzare il what
- Nel nostro caso devo poter rappresentare quali sono i supermercati, quali e quanti scaffali ci sono, per ogni prodotto in quel negozio in quale scaffale sta, ogni tipo di merce dove si trova

How

Io parto dagli use case che ho identificato, e cerco di srotolarli, provando a capire come ciascun use case può essere realizzato.

Solo condizioni necessarie (disponibilità dell'informazione)



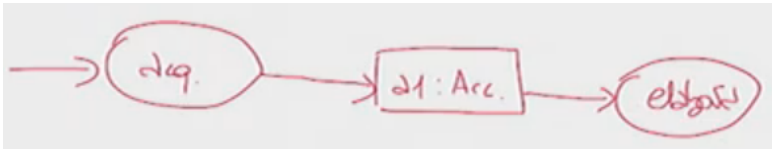
Il primo è relativo al mandare una richiesta di riempimento dello scaffale verso l'operatore. Il secondo è verso il manager dello shop e il terzo verso il centrale.

Qui vedo 3 catene esecutive, ogni volta che vendo qualcosa, quest'informazione va in pancia a 3 filoni paralleli. Io avrei potuto farlo anche sequenziale, in questo caso parallelo ha più senso.

In un diagramma di attività ho due concetti, il **dato** (rettangolare) e l'**azione** (tondo). Quando esprimo un dato ho "nome_dato : tipo". Nell'immagine vediamo :sold, :inshelf. Così si capisce esattamente di quale dato stiamo parlando, tra i dati che abbiamo identificato nel modello dei dati.

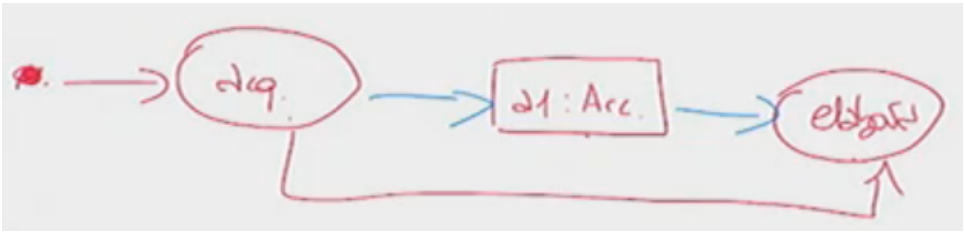
Usare uno strumento come visualparadigm aiuta a fare questi collegamenti.

Nell'esempio degli accelerometri:



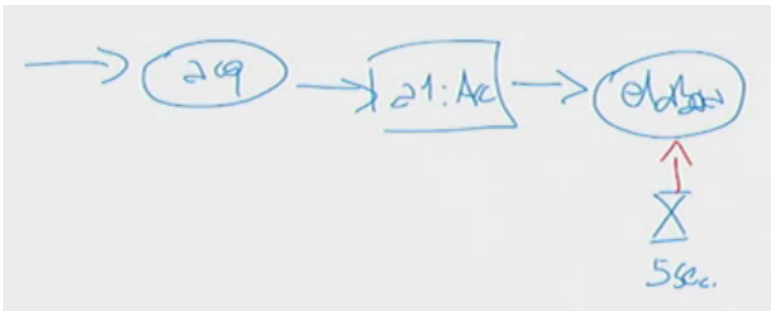
L'interpretazione di questo diagramma sarebbe che il flusso dati e il flusso controllo sono la stessa cosa.

Acquisizione quando termina genera a1, che viene dato ad elabora. Ma io dovrei fare questo:

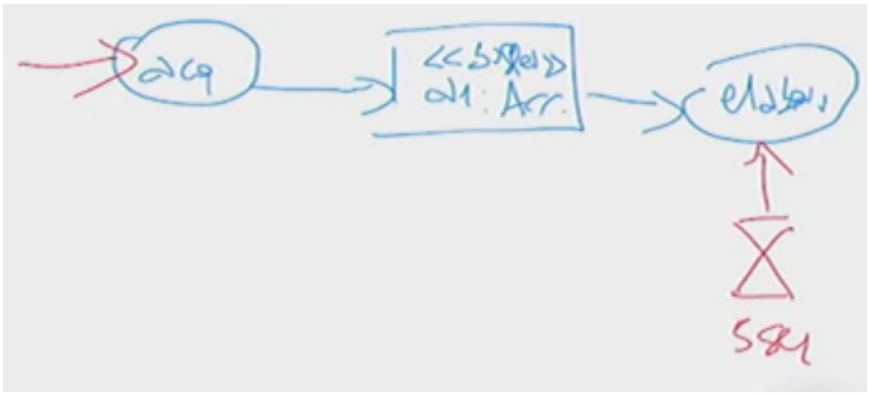


Con le frecce blu in questo caso esplicito un flusso informativo, mentre con quelle rosso il flusso di controllo, posso così dire che elabora viene eseguito a patto che a1 sia disponibile.

Alternativa:

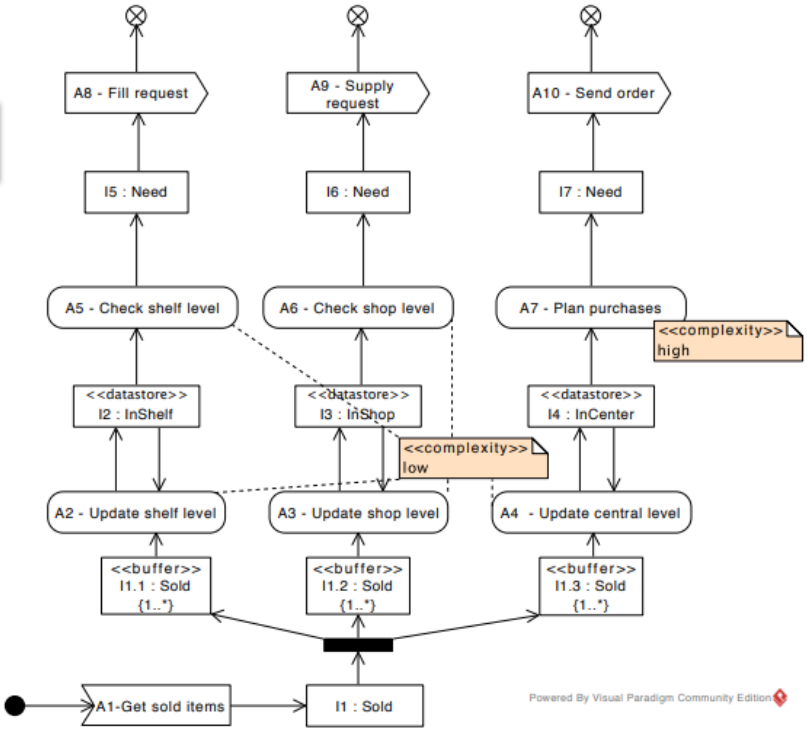


In questa situazione se acq produce dati più velocemente di 5 secondi, io lavoro solo sull'ultimo dato e quindi perdo dati, perchè elabora viene eseguito ogni 5 secondi.



Per fixare posso andare ad inserire un buffer. (il buffer c'è anche nell'immagine sopra dell'how, dove ci sono 3 buffer al posto di 1 di modo che le 3 attività possano partire a tempi diversi).

Solo condizioni necessarie (disponibilità dell'informazione)



Il box in basso a sinistra "A1- Get sold items" è un **evento entrante**, mentre "A8 - Fill request" e gli altri sono **eventi uscenti**. Mi aspetto che un evento entrante porti dentro un dato (a meno che è solo un trigger), in questo caso si porta dietro un "Sold".

Esempio: se abbiamo due azioni (tondi) collegati, senza un dato di mezzo, vuol dire che abbiamo fatto attività troppo piccole. Le attività dovrebbero produrre un dato in uscita, siamo a livello architetturale. Ovviamente ogni attività sarà poi fatta da più passaggi.

I **datastore**, che vediamo nell'immagine in mezzo, sono dei contenitori di dati (di tipo shelf o tipo inshop per esempio), è qualcosa di persistente, anche se il sistema viene spento/riavviato. Mentre invece il **buffer** ha una logica come se fosse in memoria ram, è un dato transiente, non per forza viene memorizzato.

Nell'immagine abbiamo un evento entrante (get sold items), vuol dire che avremo un pezzo di software che si interfaccia con il POS, a cui arrivano oggetti di tipo sold, un oggetto ogni 2 secondi (lo avevamo visto nelle assunzioni). A questo punto voglio rendere disponibile Sold ai 3 use case, quindi uso una fork. Questo è solamente un flusso informativo. Ogni buffer va in ingresso alla propria attività (A2, A3, A4), queste parlano ciascuna con un datastore che contiene gli elementi

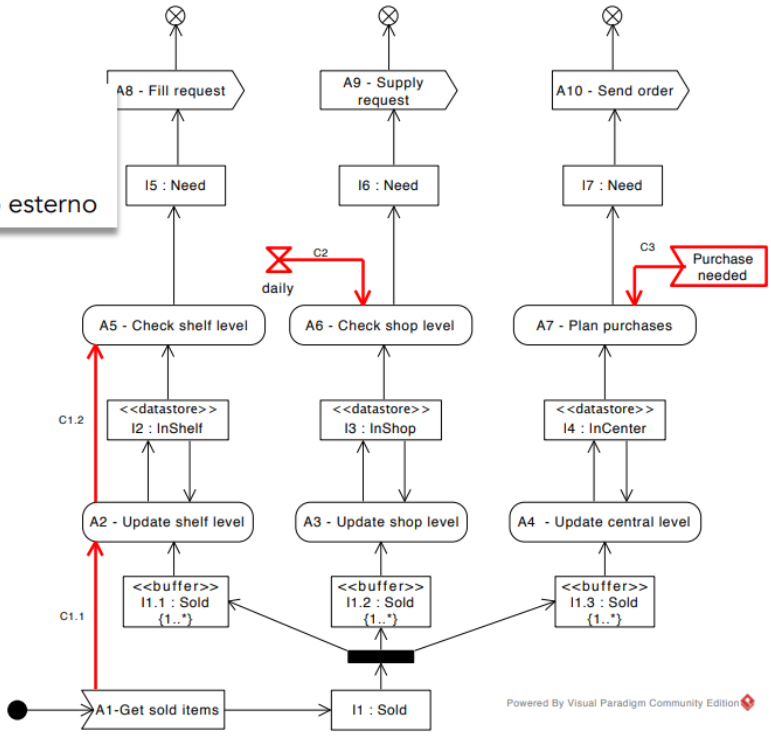
di tipo product. C'è la doppia freccia perchè c'è lettura, aggiorno il dato, e lo rimetto dentro. Il datastore è quindi usato in lettura da A5-A6-A7, dove viene modellato il caso che ci si aspetta (Need), ovvero quando mancano merci, non definisco una roba del tipo "se ci sono abbastanza merci allora... se mancano merci allora..." faccio solo il caso che mi aspetto. Il Need va all'evento uscente "Fill request" che è come se arrivasse una notifica sull'applicazione dell'operatore che deve adempire allo scaffale per esempio.

Abbiamo anche le note che dicono la **complessità** delle attività, per esempio A7 ha una complessità più alta perchè magari ho prodotti che durano poco, oppure stagionali. Il capire le complessità può pilotare il partizionamento delle responsabilità nei moduli, per evitare di mettere in uno stesso modulo delle attività che si occupano di cose diverse, o anche di cose uguali se hanno complessità alte.

Why

Si specifica il **perchè** una cosa succede tramite i **flussi di controllo**.

3 casi di innesco delle azioni:
C1: Attivazione 'sequenziale'
C2: Attivazione a tempo
C3: Attivazione a fronte di uno stimolo esterno



Quando "Get sold items" viene innescato, questo innesca anche A2 che a sua volta innesca A5.

C2, attiva il check shop level una volta al giorno.

A7 invece è attivata ad evento, è il manager che decide, oppure un algoritmo di AI.

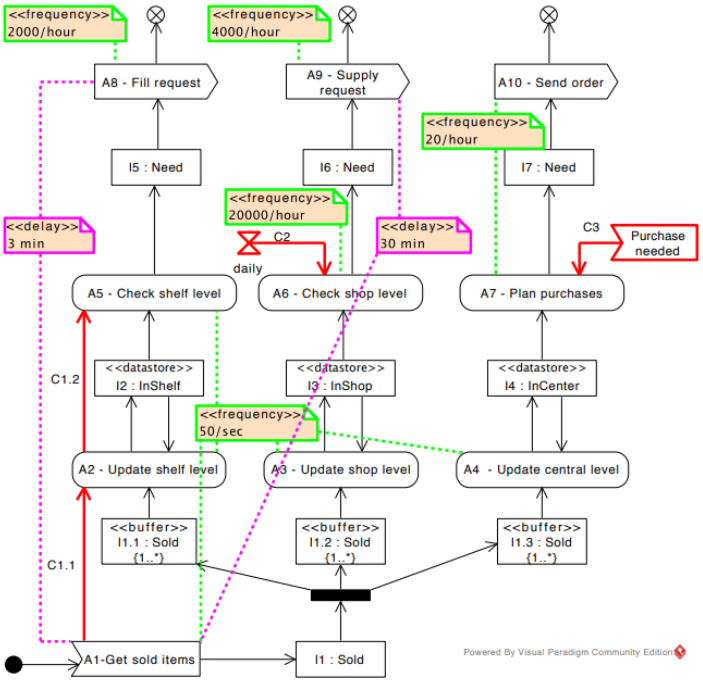
When

Il **when** è rappresentato dalle frequenze, che ci danno un'indicazione della **potenza di calcolo necessaria**, perchè specifichiamo con che frequenze le azioni vanno fatte.

Delay, timing, o tempo di risposta: entro quando un'attività deve essere eseguita

Frequenza: quando un'attività deve essere eseguita

La Frequenza mi da un'indicazione della potenza di calcolo necessaria



Per esempio vediamo che da quando viene venduto un prodotto, entro 3 minuti deve arrivare una notifica all'operatore, per evitare che lo scaffale sia non fornito. Avevamo il limite di 20 minuti per il refill, contiamo che l'operatore nei prossimi 17 minuti lo farà.

Avevamo detto che un Sold è generato ogni 2 secondi, poi abbiamo fatto un'altra assunzione, stiamo lavorando su 10 shop, 1000 prodotti, 1000 scaffali, vogliamo che uno scaffale garantisca la presenza di prodotti per 5 ore, e che ogni 5 giorni in media bisogna fare approvvigionamento di prodotti. é importante valutare il sistema nella sua complessità. Tipo le fill requests, 2000/h significa che sono 2000 all'ora per ogni scaffale, quindi se quel pezzo di software va in un modulo, magari istanzio quel modulo un certo numero di volte, per esempio tanti quanti

sono gli shop, per dividere le frequenze. 2000 è un calcolo fatto da un sold ogni 2 secondi, quindi mezzo al secondo, avendo 10 shop, dove ognuno ha 10 pos, quindi ho 100 pos, dove ciascuno ha mezzo sold al secondo, quindi 50 sold al secondo per tutto il sistema... poi il resto del calcolo.

Queste frequenze le calcolo considerando l'intero sistema, ma questo lo posso fare se e solo se riesco a fare delle assunzioni che sono sufficienti per ragionare sulle frequenze.

Le supply requests sono 4000 perchè sarebbero 2000 ma vanno ridotte (dalle assunzioni) nella mezz'ora dopo la chiusura.

Queste sono tutte cose da definire, per capire se vale la pena spezzettare i componenti. Per esempio se A3 la faccio 50/sec allora non vado a metterla insieme a chi fa A7 20/h, perchè sono due frequenze molto diverse. Cerco di mantenere coerente la distribuzione delle frequenze, perchè se sono frequenze così diverse non ha senso metterle nello stesso modulo.