Lezione 9 05/11/2024

L’altra volta abbiamo visto la problem architecture, che aveva come obiettivo la modellazione dal punto di vista dell’utente.

Architettura logica

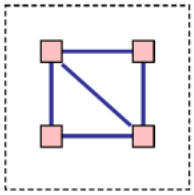
Ora prendiamo le funzionalità che abbiamo identificato, e cerchiamo di comporle in componenti logici, che poi si tradurranno in componenti concreti, che sarà l’ultima prospettiva che andremo ad affrontare.

L’obiettivo dell’**architettura logica** è quello di definire **cluster di attività** chiamate **componenti logiche**. Quindi un componente logico è un insieme di attività, non sono le componenti software, non mi preoccupo ancora di questioni realizzative (come sono fatti dentro i componenti e le modalità di interazione fra i componenti).

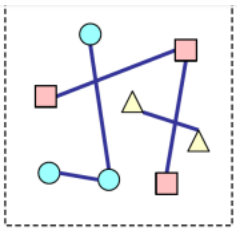
Cerco di raggruppare le attività con lo scopo di ottenere componenti:

- **Compatti** (o Coesi, Omogenei): è bene raccogliere in un componete funzionalità che sono **omogenee** fra loro in termini di requisiti **non** funzionali.
    - Esempio, non metto insieme funzionalità che distacca la rete elettrica nel caso in cui ci sia un picco e la funzionalità che calcola le bollette (timing e complessità molto diverse). Quindi metto insieme le componenti che hanno la stessa frequenza di esecuzione.
- **Isolati** - (o Disaccoppiati): è bene che i flussi informativi e la condivisone di risorse fra componenti siano il più limitato possibile. I componenti devono cercare di avere quanti meno servizi richiesti ad altri componenti sia possibile.
    - In altre parole, il componente non dipende da altri componenti
    - Vantaggi: minimizzo l’overhead nella comunicazione, flessibilità nel deployment distribuito, facile manutenibilità

La coesione è una misura della forza delle relazioni tra le responsabilità di uno specifico modulo – ovvero, dell'"unità di scopo" del modulo

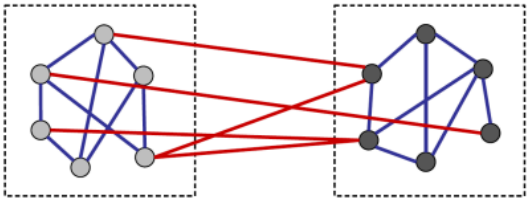


coesione più alta

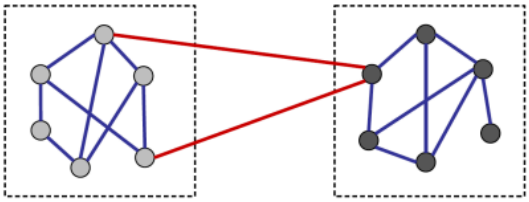


coesione più bassa

accoppiamento più alto



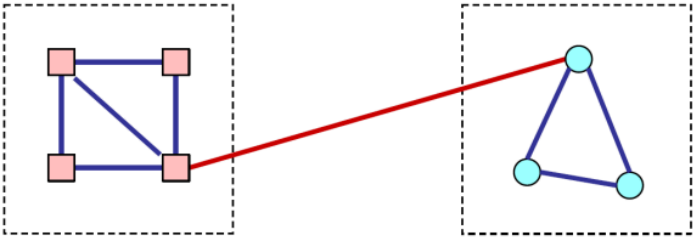
accoppiamento più basso



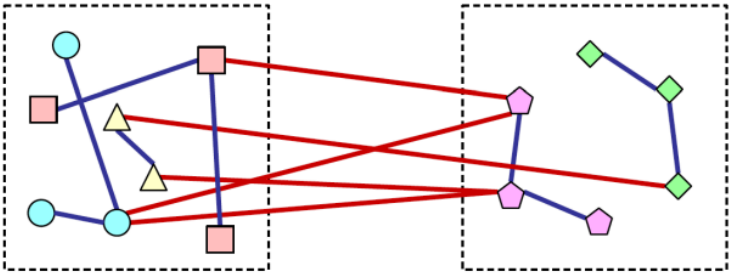
L'accoppiamento è una misura della forza delle dipendenze tra moduli

Alta coesione
Basso accoppiamento

Alta coesione
Basso accoppiamento



Bassa coesione
Alto accoppiamento



La prima soluzione qui è quella ideale.

Non esiste una regola generale per partizionale le attività in componenti… è più legato all'esperienza, e all'applicare pattern ben noti.

Non esiste l'Architettura corretta: occorre scegliere il miglior compromesso rispetto alle dimensioni rilevanti del problema, perchè abbiamo delle **dimensioni** di interesse diverse.

- Ho poca banda e mi serve restringere le comunicazioni
- Ho complessità computazionali diverse fra loro

Dimensione

Dimensione: è una proprietà che vogliamo considerare e il suo insieme di valori assumibili. Esempi:

- Complessità {alta, media, bassa}
- Frequenza temporale {1s, 1m, h, 1g}

Dimensione	Descrizione	Livello	Tipologia
Delay	Tempo massimo richiesto all'elaborazione	Architetturali	Strutturali Proprietà che sono dovute a caratteristiche intrinseche della soluzione
Frequenza	Cadenza con cui le attività vengono svolte		
Livello di astrazione	Dell'informazione trattata (what)		
Complessità	Complessità computazionale		
Location	Posizione fisica/virtuale		
…			
Goods	Tipo di prodotto	Domain-dependant	
…			
Extra flows	Intensità dei flussi informativi tra le istanze dei componenti e l'ambiente esterno (cioè gli attori)	Architetturali	Dinamiche Proprietà che dipendono dal numero di istanze dei componenti e degli attori che esistono a run-time
Intra flows	Intensità dei flussi informativi tra le istanze dei componenti		
Sharing	Intensità della condivisione di informazioni (tipicamente dati persistenti) con altre istanze dei componenti		
Control flows	Intensità dei flussi di controllo tra le istanze dei componenti		

Queste sono le dimensioni più comuni a cui si ha a che fare.

Nel nostro progetto per esempio abbiamo per l'accelerometro il valore letto per i 3 assi, poi ho dei dati grezzi e filtrati. Poi ho una finestra temporale di valori, che potrebbero avere all'interno dei dati interpolati. (e arrivo a 3 dati). Poi abbiamo che

la persona ha fatto un'attività. Così ho usato 4 livelli dell'informazione, a diversi livelli di astrazione, da quello più grezzo. Tutti mostrano la stessa informazione, ma in modo diverso.

Il livello di astrazione può anche essere quello relativo alla funzionalità o all'insieme di funzionalità che ho identificato lavorino su un certo dato.

Su quelle nella prima parte (prima della linea rossa) c'è più libertà di interpretazione, quelle sotto sono più standardizzate.

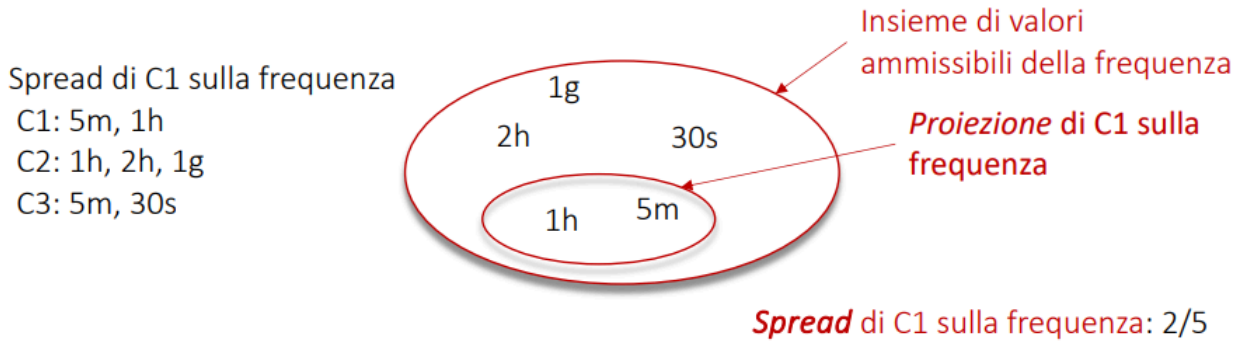
Quelli sopra sono strutturali, ovvero i raggruppamenti dei componenti sono una decisione sulla struttura. Devo mettere insieme quelle con frequenze e complessità computazionali simili.

Quelle sotto sono dinamiche perché

Una volta che ho scelto un partizionamento su una dimensione, come valuto se una soluzione adottata è accettabile?

Proiezione e spread

- **Proiezione** di un componente su una dimensione è l'insieme di valori assunti dalle sue attività in quella dimensione
- **Spread** di un componente su una dimensione è una stima quantitativa di quanto è ampia la proiezione di una dimensione sulla dimensione stessa



Supponiamo di avere C1, C2 e C3 che sono 3 componenti. Ciascuna ha le frequenze scritte. Se C1 ha 5m e 1h la sua proiezione è 1 ora e 5 minuti. Lo spread è quanto mi allargo sui valori possibili. Un alto spread è negativo, vuol dire che

stiamo raggruppando in un’entità tante funzionalità con frequenze diverse, che non è la situazione ideale.

Overlap e interferenza

- **Overlap** di un componente su una dimensione dinamica è l’unione delle sue intersezioni con le altre componenti. Ovvero, quanta comunicazione c’è tra i componenti. Negli esempi visti prima, in quello a alta coesione e basso e basso accoppiamento abbiamo un’intersezione piccola, nel secondo caso invece è grande.
    - Aspetto strutturale → il numero di interfacce/risorse richieste che quindi creano dipendenza con altri componenti
    - Aspetto dinamico → l’intensità dei flussi **contemporanei** (runtime)
        - invocazione ad esempio della stesse interfaccia contemporaneamente da più istanze di componenti
- **Interferenza** di un componente su una dimensione è una **stima** quantitativa del suo suo overlap su quella dimensione (complessivamente, in generale, senza concentrarsi su una singola componente). La stima dell’interferenza va fatta considerando le istanze dei componenti contemporaneamente in esecuzione, non sul singolo componente.

Le definizioni date permettono di determinare se i componenti sono **compatti** e **isolati.**

Compattezza significa che lo **spread** di un componente su tutte le dimensioni **strutturali** è **basso**. Lo spread è legato alla compattezza poiché indica l’omogeneità del componente che è legata a proprietà strutturali

Isolamento significa che **l’interferenza** di un componente su tutte le dimensioni **dinamiche** è **basso**. L’interferenza è legata all’isolamento poiché indica il livello di interazione fra componenti

Dimensione	Livello	Tipologia	Influenza	Indicatore
Delay	Architetturali	Strutturali	Compattezza	**Spread** indica l'omogeneità del componente su una dimensione
Frequenza				
Livello di astrazione				
Complessità				
Location				
…				
Goods	Domain-dependant			
…				
Extra flows	Architetturali	Dinamiche	Isolamento	**Interferenza** indica il livello di 'interazione' (flussi) tra i componenti/attori/dataStore
Intra flows				
Sharing				
Control flows				

Per quelle strutturali posso fare una formula matematica

$$spread_D = \frac{\sum_1^n spread_i}{n}$$

mentre per calcolare l'interferenza occorre considerare il numero totale di istanze di componenti e di attori che sono a regime.

È una stima di cui non esiste una formula ben precisa. Occorre considerare la fotografia del sistema in un determinato istante (quali e quante istanze sono in esecuzione) e stimare i valori di ogni dimensione.

Esempi di interferenza

- Si supponga
  - Di avere due tipi di componenti: Client (Cli) e Authentication (Auth)
  - Che contemporaneamente si hanno in esecuzione
    - 10 istanze di Cli ed 1 istanza di Auth
    - -> l'interferenza relativa all''intra-flow' è elevata
  - Che contemporaneamente si hanno in esecuzione
    - 10 istanze di Cli ed 5 istanze di Auth
    - -> l'interferenza relativa all''intra-flow' è medio-bassa
- Si supponga
  - Di avere due tipi di componenti: Profile Manager (PM) e Authentication (Auth)
  - Che contemporaneamente si hanno in esecuzione
    - 1 istanza di PM e 10 di Auth
      - Se le 10 di Auth accedono a profili doversi
        - -> l'interferenza relativa allo 'sharing' è bassa
      - Se le 10 di Auth accedono allo stesso profilo
        - -> l'interferenza relativa allo 'sharing' è alta

Lo sharing si applica ai datastore, è la condivisione delle informazioni. C'è anche una verifica dei ruoli per vedere se la persona è autorizzata ad accedere ai dati.

Footprint

Il **footprint** è l'insieme di tutti gli **spread** e le **interferenze**. L'obiettivo è quello di ottenere un footprint il più piccolo possibile su tutte le dimensioni, che porta quindi a compattezza e isolamento.

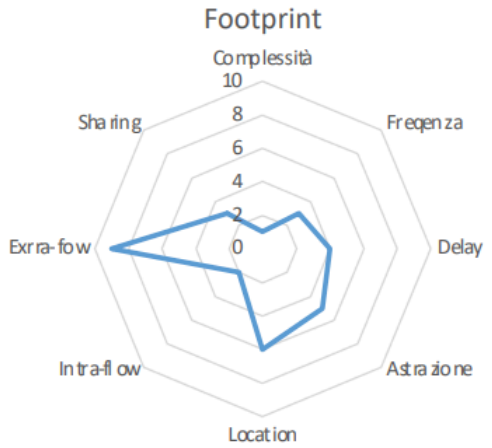
- Modo alternativo di formulare il ben noto criterio di alta coesione e basso accoppiamento

Procedimento

- Identifico un criterio di partizionamento
  - Per dominio applicativo, per dimensione, etc.
- Per ogni componente ottenuto:
  - **Valuto** ciascuna dimensione di interesse identificando la **proiezione** (per le statiche) o l'**overlap** (per le dinamiche)
  - **Quantifico** le proiezioni e l'overlap assegnando un valore in un range definito (esempio, [0 – 1], [0 – 10], o [0 – 100]) ottenendo rispettivamente **spread** e **interferenza**
- Uso quindi un diagramma Radar per visionare la bontà della soluzione
- Come valuto la bontà di un componente?
  - Se un componente ha attività omogenee dal punto di vista di una dimensione statica (ad esempio la complessità), avrà uno spread basso per quella dimensione
    - Un valore verso il basso del range scelto
  - Se un componente ha poche interazioni (con attori, altre componenti, o datastore), avrà un'interferenza bassa per quella dimensione
    - Un valore verso il basso del range scelto

Quindi potrei fare un primo partizionamento per tipologia di dato, e poi ripartiziono usando un altro criterio (come la frequenza).

strutturali	spread	Complessità	1
		Freqenza	3
		Delay	4
		Astrazione	5
		Location	6
dinamiche	interferenza	Intra flows	2
		Extra fows	9
		Sharing	3
		Control flows	2



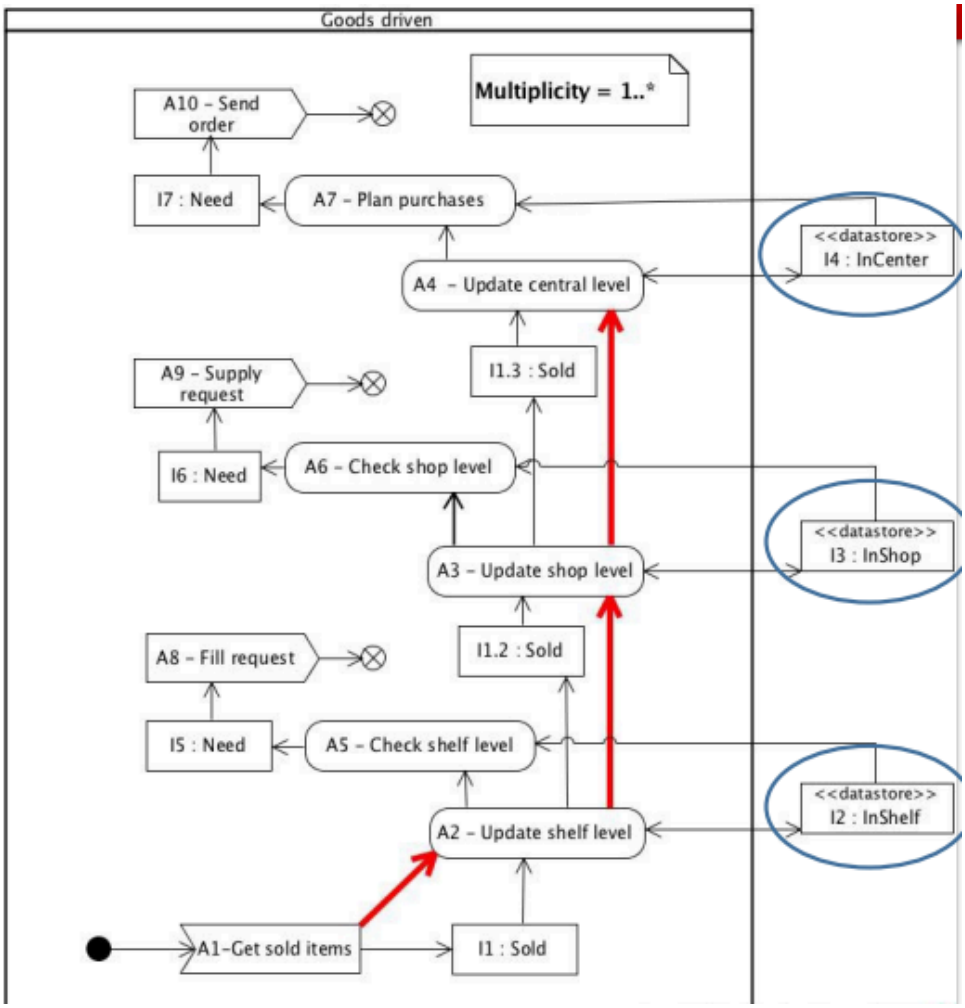
Soluzione buona: ho tutti i valori in centro
Una soluzione è tanto migliore, quanto è piccolo il footprint

Se ho un extra flow così alto, posso dire che faccio in modo di garantire l'availability e la velocità degli attori.

Logical Architetture applicata al caso di studio

Partizionamento Goods driven

Questa è una rivisitazione della soluzione che abbiamo identificato nella problem architecture.



C'è un componente che fa tutto.

La rivisitazione prevede che tutto venga fatto in maniera sequenziale. Quando ho un evento di vendita, viene generato un sold, che va ad update shelf level, ...

Quindi la moltiplicità 1..* arriva dal fatto che ogni volta viene istanziato un componente.

Qua abbiamo risorse condivise nei datastore, che è unico, ma è usato da più attività sia in lettura che in scrittura, ed è esterno.

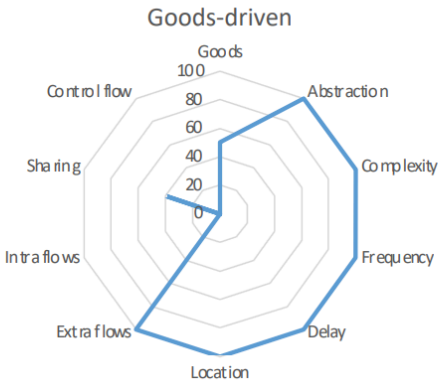
Posso avere più istanze dello stesso componente attive contemporaneamente: ogni volta che viene venduto un bene, viene generata una nuova istanza

Il sistema è quindi costituito da un insieme di istanze che hanno lo stesso comportamento e struttura. Questo stile di partizionamento viene comunemente definito **verticale**: ogni componente tratta tutti i layer architetturali.

Vediamo come abbiamo calcolato il tutto. Proviamo a dare una quantificazione (values) per ciascuna dimensione.

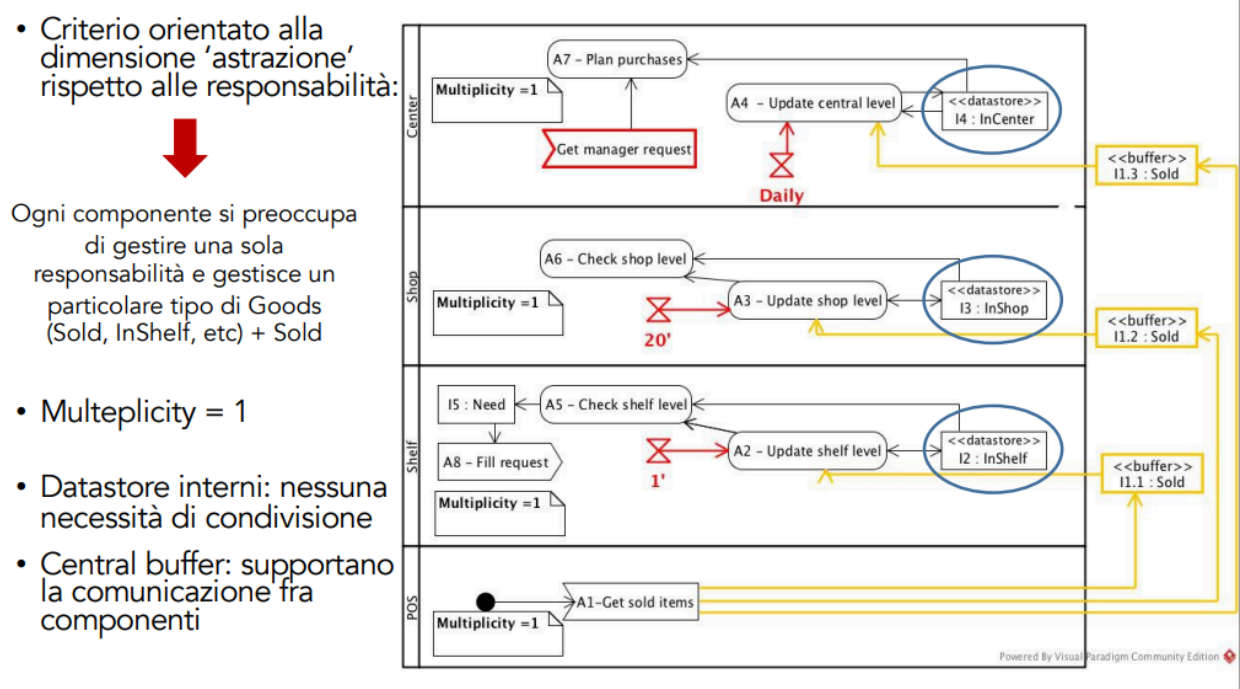
		Dimension	Values	
statiche	Proiezioni	Goods	medium	tratto un solo tipo di Goods (Coca cola, Pane, etc.) per istanza, am deve esesre in grado di gestirli tutti
		Abstraction	all	tratto tutti i tipi di astrazione dell'informazione: da Sold a inCenter
		Complexity	all	il componente include tutte le attività, quindi anche tutte le complessità identificate
		Frequency	all	il componente include tutte le attività, quindi anche tutte le frequenze identificate
		Delay	all	il componente include tutte le attività, quindi anche tutte i delay identificati
		Location	all	il componente include tutte le attività, quindi tutte le locazioni possibili degli attori e dei Goods
dinamiche	Overlap	Extra flows	high	il componente è unico e quindi tutte le sue istanze comunicano con tutti gli attori esterni
		Intra flows	-	Il componente è unico, le istanze fra loro non scambiano informazione
		Sharing	medium/low	I datastore hanno singole istanze per ciascun tipo di prodotto e per ciascuna astrazione di prodotto: la probabilità che due istanze accedano contemporaneamente alla stessa informazione media/bassa
		Control flows	-	Il componente è unico, le istanze fra loro non si scambiano messaggi di controllo

		Dimension	Values
statiche	spread	Goods	50
		Abstraction	100
		Complexity	100
		Frequency	100
		Delay	100
		Location	100
dinamiche	interferenza	Extra flows	100
		Intra flows	0
		Sharing	40
		Control flows	0



Partizionamento Abstraction-driven

Il grafico alla fine non è bello, e non riusciamo a sistemarlo con delle tattiche, quindi dobbiamo rivisitare la soluzione. Provo a fare un altro tipo di partizionamento, in questo caso abstraction driven. Tengo in considerazione il tipo di attività che viene fatta. Nel primo gestisco la vendita, nel secondo l'aggiornamento dello shef, poi dello shop e poi del sold. Quindi abbiamo messo insieme le attività che hanno lo stesso obiettivo (catturare l'evento di vendita, gestire il livello di shop, ...)



Questo è più legato alla funzionalità e meno al dato. Ho un sold che va a tutti i livelli.

Come da specifica, ho tempi diversi, in questo modo quando ricevo un sold, lo vado a mettere nei 3 diversi buffer, di modo che quando scatta ciascuna, svuota il buffer e legge il prodotto. Però ogni volta che una cassa vende qualcosa, deve comunicare con il center, con lo shop... ho delle comunicazioni enormi.

Poi la singola istanza di POS deve gestire TUTTI i sold (e ne arriva uno ogni 2 secondi), quindi il componente deve essere molto veloce. Stessa cosa lo shelf, nel momento in cui è attivato dovrà svuotare il buffer con tutti i sold che sono attivati in un minuto.

Quindi POS ha un sacco di richieste, mentre i buffer devono essere dimensionati in maniera adeguata.

- Una istanza di componente attiva per ogni tipo di astrazione
- Il sistema è quindi costituito da 4 istanze ciascuna creata a partire dal componente e che hanno quindi comportamenti e strutture diverse
- Il POS scambia info con tutti: informazione attraversa più layer
    - no modificabilità: se cambio l'interfaccia dell'elemento logico POS, devo cambiare tutti i componenti che dipendono da questa

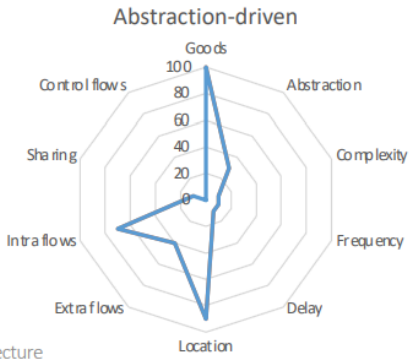
- no uso corretto del pattern layer
- traffico elevato anche ai livelli alti

- Buffer devono essere opportunamente dimensionati perché i tempi di attivazione dei componenti sono diversi
- Questo stile di partizionamento viene comunemente definito orizzontale: ogni componente tratta un solo layer architetturale

  In rosso i valori che sono cambiati:

		Dimension	Values	
statiche	proiezioni	Goods	all	Ogni componente tratta più tipi Goods alla volta
		Abstraction	medium/low	Ogni componente tratta un solo tipo di Goods per costruzione (o Sold, o InShelf, ...) + il Sold che è su tutti I livelli
		Complexity	low	Ogni componente integra attività con livello omogeneo di complessità
		Frequency	low	Ogni componente integra attività con livello omogeneo di frequenza
		Delay	low	Ogni componente integra attività con livello omogeneo di delay
		Location	circa all	Ad eccezione di del componente inCenter, gli altri componenti hanno a che fare con attori sparsi in diverse location. Esempio inShop deve interagire con attori che solo localizzati nei propri Shop
dinamiche	overlap	Extra flows	medium/low	Interazione con i 100 POS e con i 10 operatori diversi ma con tempistiche lasche (vedere assunzioni)
		Intra flows	**medium/high**	Il componente POS interagisce con tutti i componenti (scambia informazioni con tutti)
		sharing	low	Non c'è condivisione: i datastore sono interni, c'e' comunque scambio dati (Sold)
		control flows	-	I componenti fra loro non interagiscono

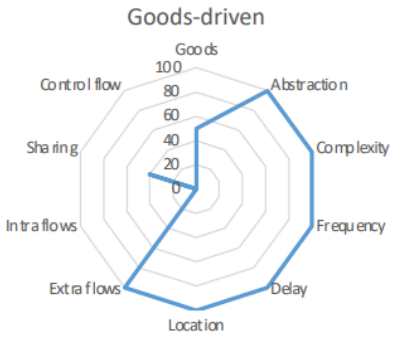
		Dimension	Values
statiche	spread	Goods	100
		Abstraction	30
		Complexity	10
		Frequency	10
		Delay	10
		Location	90
dinamiche	interferenza	Extra flows	40
		Intra flows	70
		Sharing	10
		Control flows	0



Software Architecture

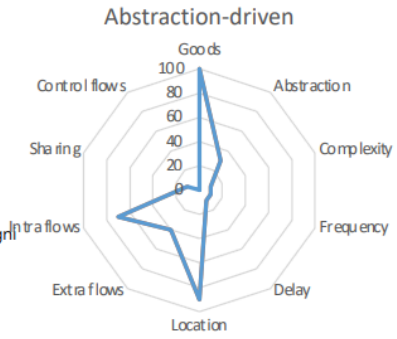
- Il footprint di una partizione ottimale dovrebbe avere un'area piccola al fine di garantire compattezza e isolamento

- Partizionamento verticale Goods-driven
  - Pros:
    - Struttura interna semplice
    - Intra-flows inesistente che permette di istanziare molti componenti senza produrre overhead nella comunicazione -> alto livello di scalabilità
  - Cons:
    - Componenti monolitici-> bassa compattezza
    - Il componente interagisce con attori distribuiti:
      - Costi di comunicazione elevati
      - Criticità della infrastruttura di comunicazione



- Partizionamento orizzontale Abstraction-driven
  - Pros:
    - Buon grado di compattezza in termini di abstracion, complexity e timing
    - Buona base per la distribuzione anche su piattaforme eterogenee
  - Cons:
    - Location e Intra-flow suggeriscono delle complicazioni dal punto di vista
      - dell'overhead nelle comunicazioni (occorre gestire la localizzazione di ogni attore) e
      - nella struttura interna che può diventare complicata (occorre gestire la concorrenza)

Software Architecture



Partizionamento Abstraction-driven with sharing

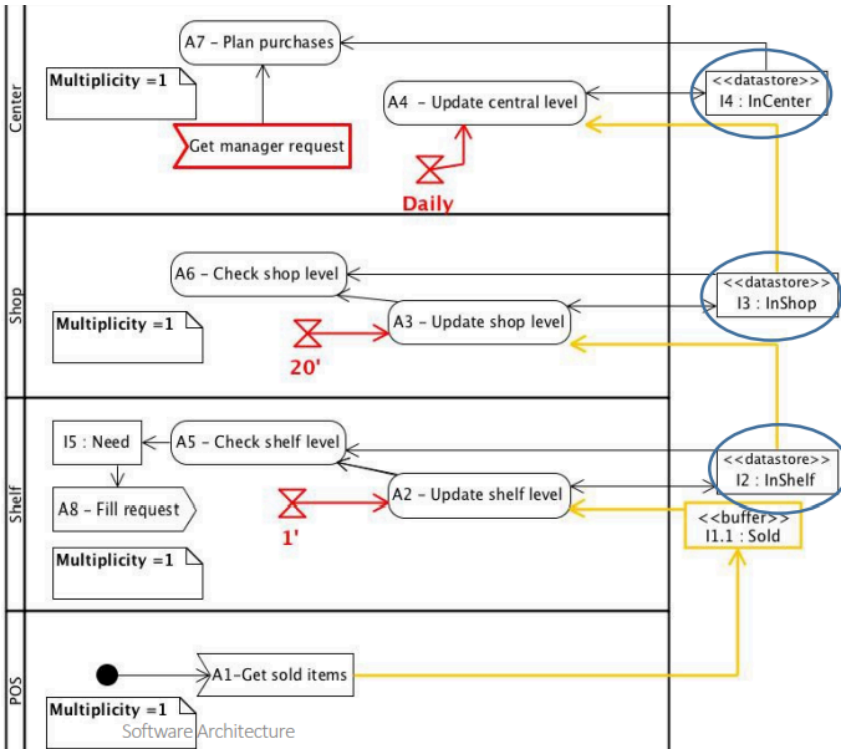
Facciamo una miglioria mettendo fuori il datastore.

Questa soluzione però può portare ad un'overhead della computazione, perchè il livello shop non sa più quali sono i sold, perchè li prende dal datastore. Quindi si dovrebbe fare un datastore un po' diverso che separa i dati vecchi e quelli nuovi.

- Criterio orientato alla dimensione 'astrazione' con sharing dei dati:

I componenti più alti interagiscono attraverso sharing dei dati e non via flussi informativi realizzati con buffer

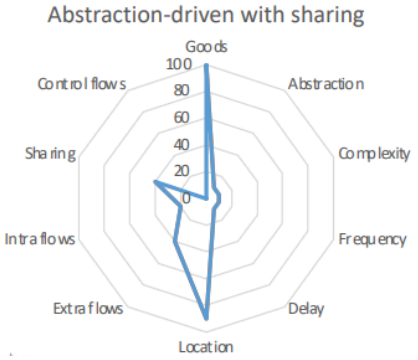
I datastore non sono più interni ed ho un solo buffer



Questa soluzione diminuisce gli intra-flows, mentre lo sharing è aumentato.

		Dimension	Values	
statiche	proiezioni	Goods	all	Ogni componente tratta ogni tipo di Products (Coca, Fanta, etc.)
		Abstraction	low	Ogni componente tratta un solo tipo di Goods per costruzione (o Sold, o InShelf, ...)
		Complexity	low	Ogni componente integra attività con livello omogeneo di complessità
		Frequency	low	Ogni componente integra attività con livello omogeneo di frequenza
		Delay	low	Ogni componente integra attività con livello omogeneo di delay
		Location	circa all	Ad eccezione di inCenter, gli altri componenti hanno a che fare con attori sparsi in diverse location. Esempio inShop deve interagire con attori che solo localizzati nei propri Shop
dinamiche	overlap	Extra flows	medium/low	Interazione con I 100 POS e con operatori diversi ma con tempistiche lasche (vedere assunzioni)
		Intra flows	medium/low	il componente POS scambia informazioni solo con Shelf
		Sharing	medium	Condivisione di datastore
		Control flows	-	I componenti fra loro non interagiscono

		Dimension	Values
statiche	spread	Goods	100
		Abstraction	10
		Complexity	10
		Frequency	10
		Delay	10
		Location	90
dinamiche	interferenza	Extra flows	40
		Intra flows	20
		Sharing	40
		Control flows	0



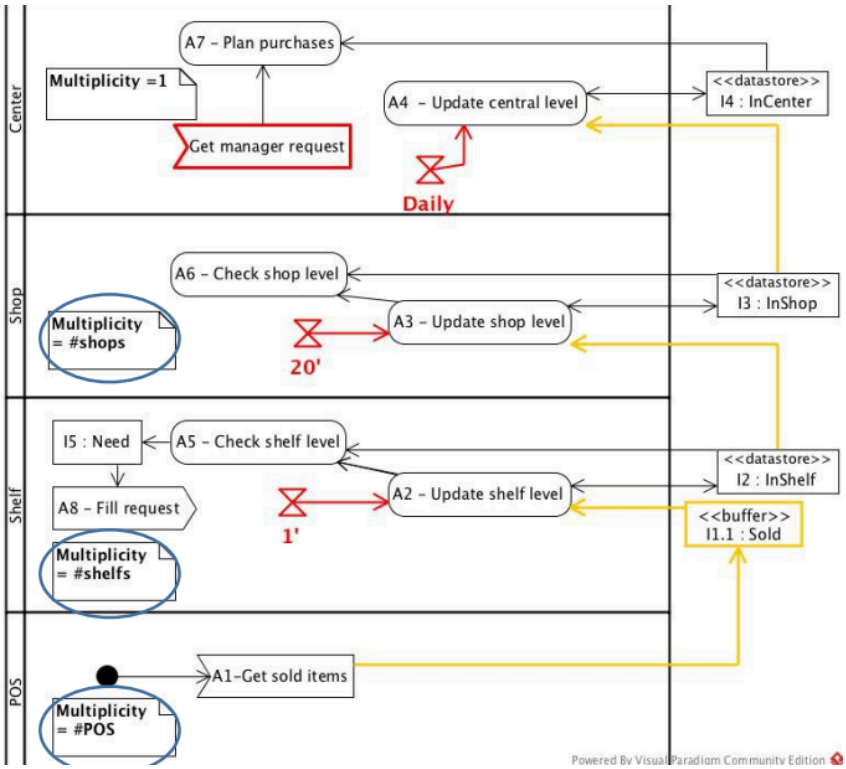
Partizionamento Multidimensional-driven

L'ultima soluzione è il **partizionamento multi-dimensionale**.

- Criterio orientato alla dimensione '**astrazione**' con **sharing** dei dati, ma considerando la '**location**':

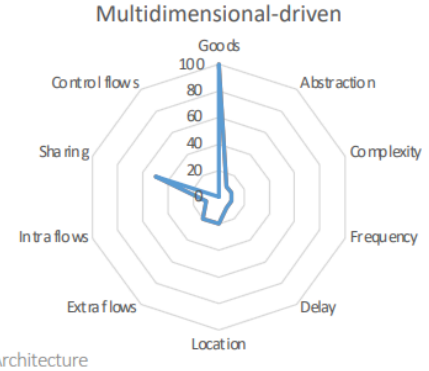
Uguale alla precedente, ma con multiplicity cambiati

- Un componete per ogni shop, uno per ogni shelf, uno per ogni POS



		Dimension	Values	
statiche	proiezioni	Goods	all	Ogni componente tratta ogni tipo di Products (Coca, Fanta, etc.)
		Abstraction	low	Ogni componente tratta un solo tipo di Goods per costruzione (o Sold, o InShelf, ...)
		Complexity	low	Ogni componente integra attività con livello omogeneo di complessità
		Frequency	low	Ogni componente integra attività con livello omogeneo di frequenza
		Delay	low	Ogni componente integra attività con livello omogeneo di delay
		Location	low	Ogni componente ha a che fare con attori colocati
dinamiche	overlap	Extra flows	low	I componenti interagiscono con un solo attore
		Intra flows	medium/low	il componente POS scambia informazioni solo con Shelf
		Sharing	medium	Condivisione di datastore
		Control flows	-	I componenti fra loro non interagiscono

		Dimension	Values
statiche	spread	Goods	100
		Abstraction	10
		Complexity	10
		Frequency	10
		Delay	10
		Location	20
dinamiche	interferenza	Extra flows	20
		Intra flows	10
		Sharing	50
		Control flows	0



Software Architecture

Risultato: architettura che si presta bene alla distribuzione

- Le istanze dei componenti Shelf e Shop posso essere decentralizzate su computing node periferici e si gestiscono i propri vincoli di timing (riduco i costi di comunicazione e le criticità del sistema di comunicazione)
- La componente Center che effettua computazioni complesse ma non time-critical è ospitata in un high-performance computing node

La dimensione Goods è ancora non ottimale, ma può non essere un problema se ogni
tipo di prodotto viene trattato allo stesso modo

In generale, i partizionamenti verticali ed orizzontali possono essere combinati:

- I partizionamenti orizzontali sono soggetti a partizionamenti verticali per migliorare la scalabilità

Altri criteri di partizionamento

- Tipi di prodotti se sono gestiti con strategie diversi.

In generale non esiste la soluzione corretta, ma esistono diverse soluzioni che sono giuste rispetto a questioni di dominio applicativo, costi, risorse disponibili

- L'architettura logica deve definire una specifica soluzione di progettazione software, indicando i tipi componenti software e le relative istanze

- In particolare si deve:
  1. definire i tipi dei componenti, in termini di raggruppamenti delle attività individuate nella Problem Architecture
  2. per ogni tipo di componente, definire quante istanze saranno presenti nel sistema (molteplicità)
  3. valutare il footprint della soluzione proposta, identificando i pro e contro essenziali della soluzioni