UNIVERSITÀ DELLA CALABRIA
il Campus per eccellenza

Domain-Driver Design

Mario Alviano

Domain-Driven Design (DDD)

Un approccio allo sviluppo di software complesso

- Focalizzato sul dominio principale
- Esplora i domini attraverso una collaborazione fra esperti del dominio e sviluppatori software
- Parla un linguaggio onnipresente all'interno di un contesto esplicitamente limitato

Non ci accontentiamo che il nostro sistema funzioni.

Vogliamo realmente capire **cosa** stiamo costruendo.

Modelli di dominio

- La base fondante di DDD
- Definiscono in modo inequivocabile e rigoroso cosa il sistema deve fare
- Il sistema non deve consentire altri usi
- I modelli di dominio sono costituiti da oggetti valore (value objects) ed entità (entities)
- Strutture più grandi sono rappresentate da aggregati

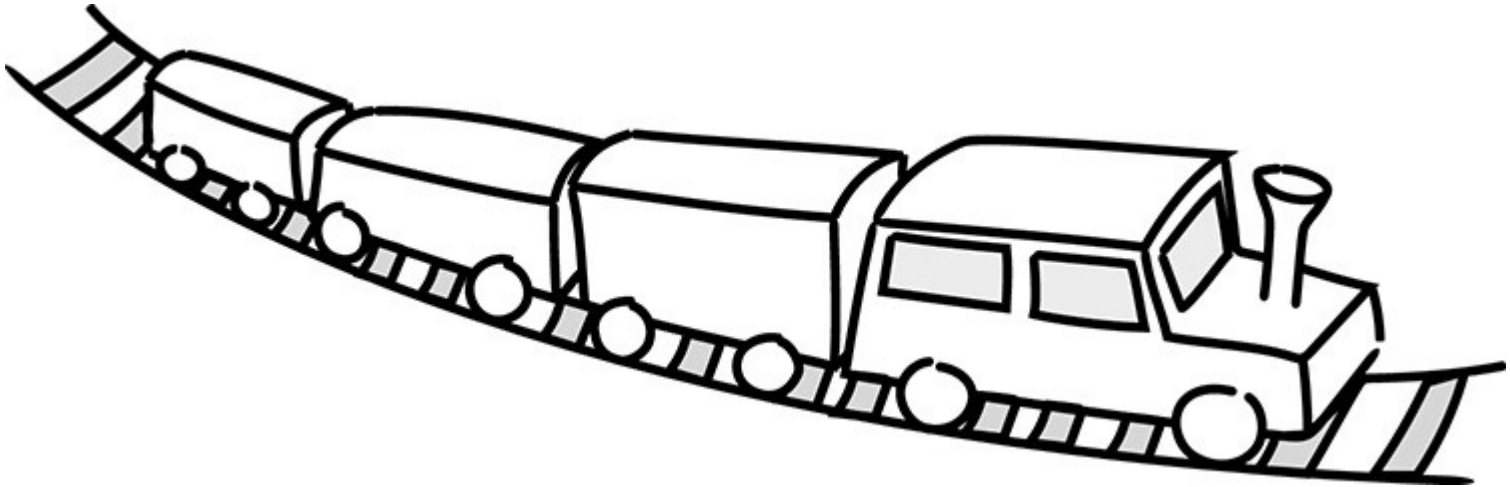
Integrazione di più sistemi

- Contesti limitati (bounded contexts) per ogni sistema (o modulo)
- Mappature di contesto (context mappings) per lo scambio di informazioni fra diversi sistemi
- Questa divisione facilita lo sviluppo sicuro del software

I modelli sono semplificazioni (1)

- Tutto ciò che è irrilevante nel contesto viene rimosso
- Ad esempio, check-in di bagagli all'aeroporto
  - Il peso del bagaglio è rilevante
  - Il numero di scarpe del passeggero non lo è
- Un modello è un'astrazione della realtà
- Diverse rappresentazioni (testuali, grafiche) per uno stesso modello

I modelli sono semplificazioni (2)



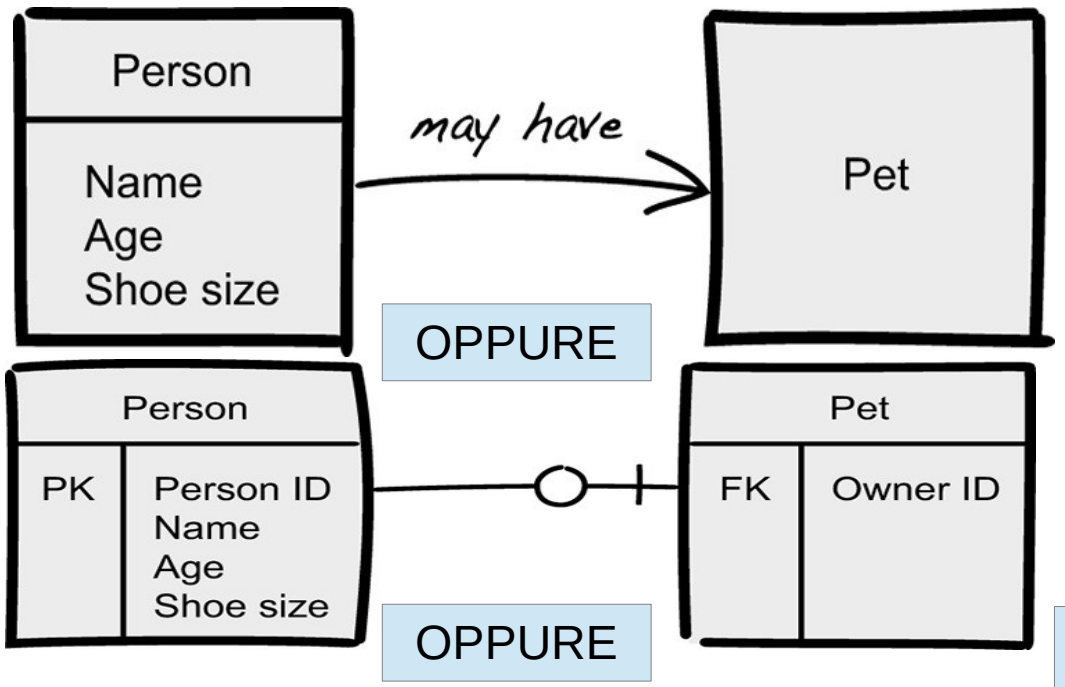
Un modello è una semplificazione della realtà
che sia ancora accettabile come
valida rappresentazione di qualcosa di reale

Modello di treno

Ci aspettiamo che alcune caratteristiche siano in comune con il treno vero: colore, dimensione relativa, forma, movimento su rotaia

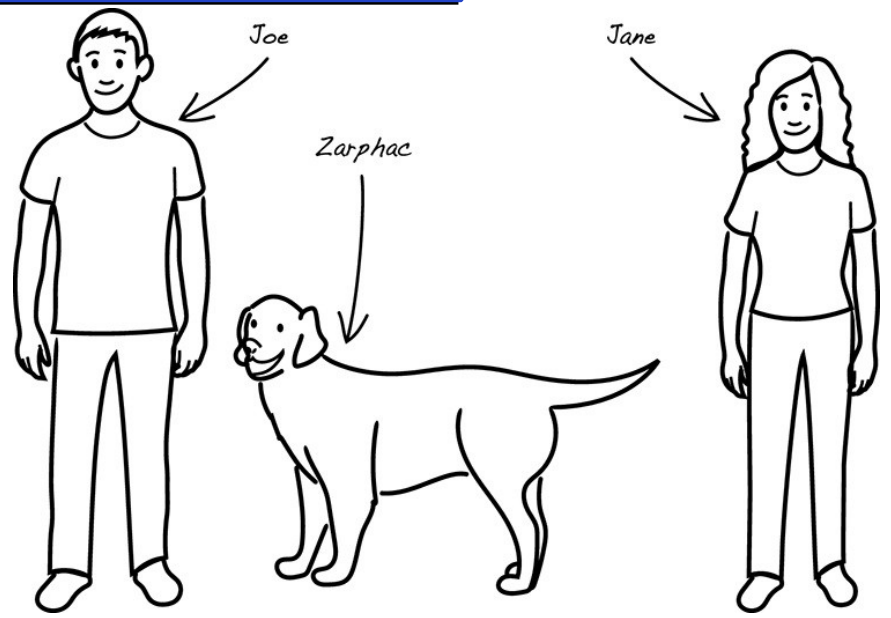
Ci sono molte caratteristiche diverse: materiale, dimensione assoluta, peso, propulsione, curve delle rotaie

I modelli sono semplificazioni (3)



descrizione testuale, pseudo-codice, ...

KISS, Keep It Simple and Stupid



Joe, age 34, shoe size 9, and his dog Zarphac, together with
Jane, age 28, shoe size 6, no pet
(altezza, genere e altre caratteristiche non rilevanti per questo modello)

I modelli sono rigorosi (1)

- Le semplificazioni consentono di essere precisi e formali
- Non usare più termini per riferire lo stesso concetto
- Ad esempio, pessima modellazione di bagagli in aeroporto
  - Al check-in: “number of bags”
  - Al gate: “baggage count”
  - Tablet del personale: “luggage”
  - Inconsistenze anche nei numeri: bagagli sul nastro inclusi o no?
  - Bisogna sempre tenere in considerazione la sorgente del dato

I modelli sono rigorosi (2)

- Altro errore comune è semplificare eccessivamente
- Se tutto è rappresentato come un oggetto, generico, ci saranno sicuramente problemi di sicurezza
- Bisogna prestare attenzione agli esperti di dominio
- Se non abbiamo un’idea chiara del dominio, dobbiamo chiedere a loro
- Se non lo facciamo, ce ne pentiremo

I modelli sono rigorosi (3)

- “Molte persone hanno solo un animale domestico.”
- Dobbiamo rendere il requisito formale
- Chiedi se “È possibile avere più di un animale domestico?”
- “Oh, è veramente insolito.”

1) Meglio usare una lista di animali domestici
   - Il codice è più complesso e magari inutilmente
2) Consentiamo solo un animale domestico
   - Prima o poi capita una persona con più animali domestici

I modelli sono rigorosi (4)

- Continua a chiedere finché il requisito non è formalizzato
- “Dovremmo consentire più animali domestici? O inseriamo una restrizione sull’averne solo uno?”
- Questa è una decisione di business, non tecnica
- Compete all’esperto del dominio, non allo sviluppatore

I modelli sono rigorosi (5)

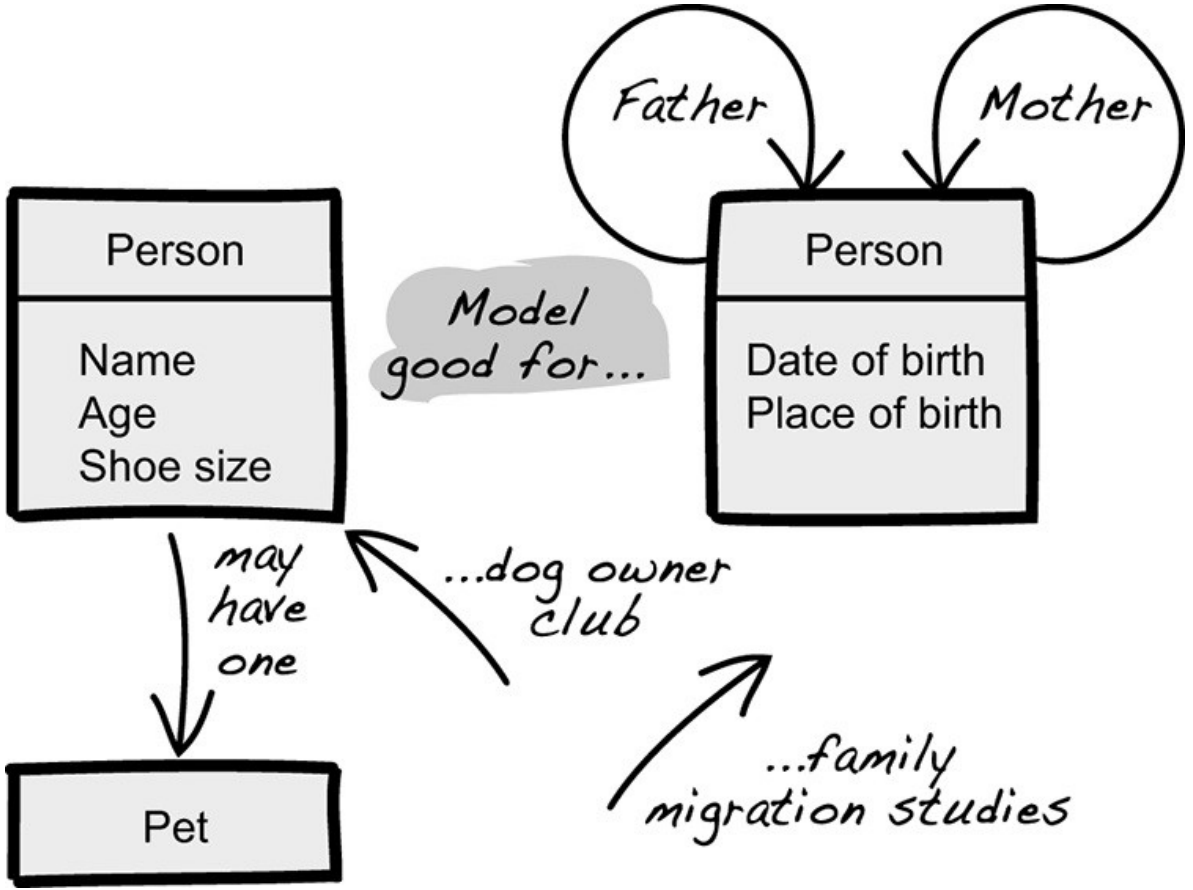
```
class Person {     ①
    private String name;

    private int age;
    private int shoeSize;
    private Animal pet;
    void growOlder() {
       this.age++;
    }
    void swapPetWith(Person other) {
     ...
    }
}
```

Il modello concordato impatta la realizzazione del software

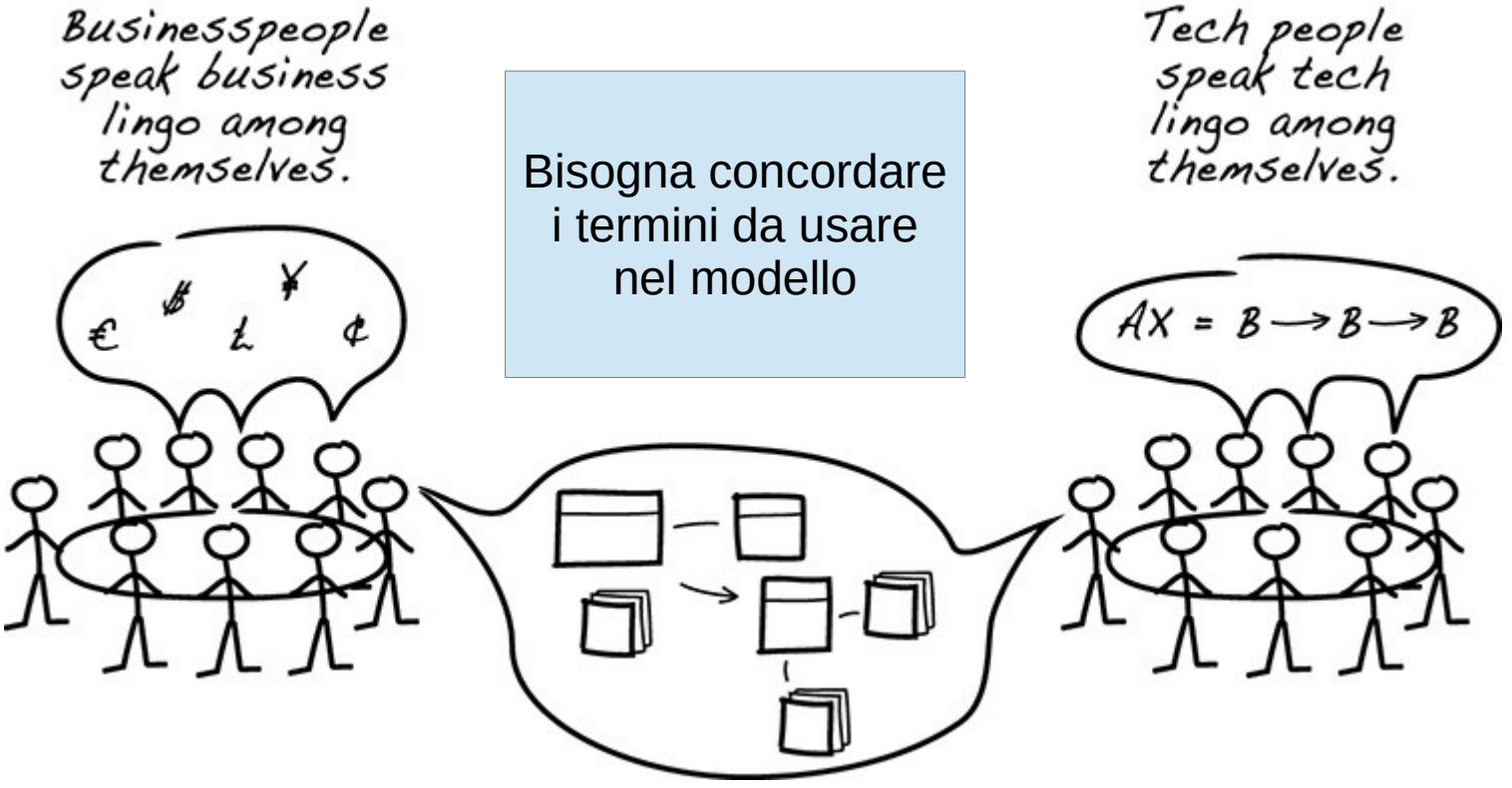
Il metodo per scambiare gli animali domestici di due persone dipende fortemente dal requisito sul numero di animali domestici (in questo caso solo uno)

Questa classe ha diversi problemi di progettazione. Riesci a individuarli? Discutine con gli altri

Fare un modello significa sceglierene uno

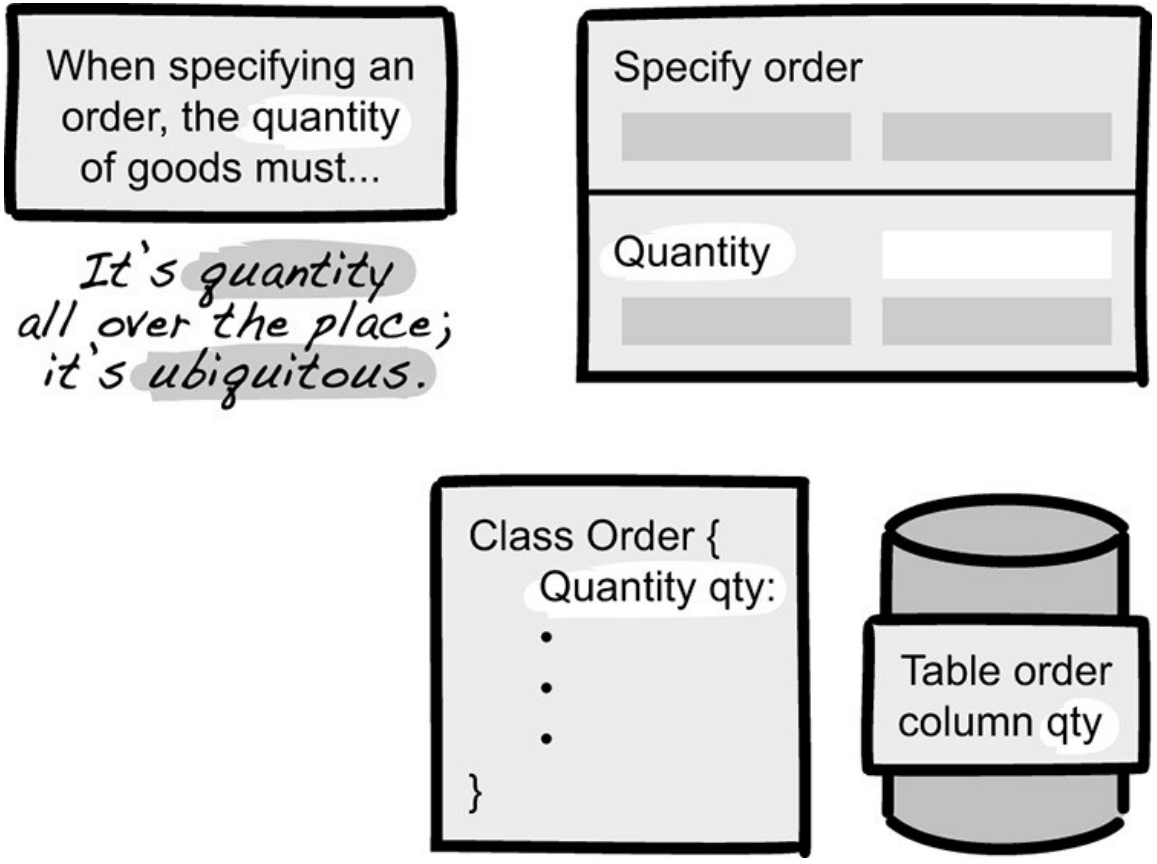


Introdurre un linguaggio onnipresente (1)



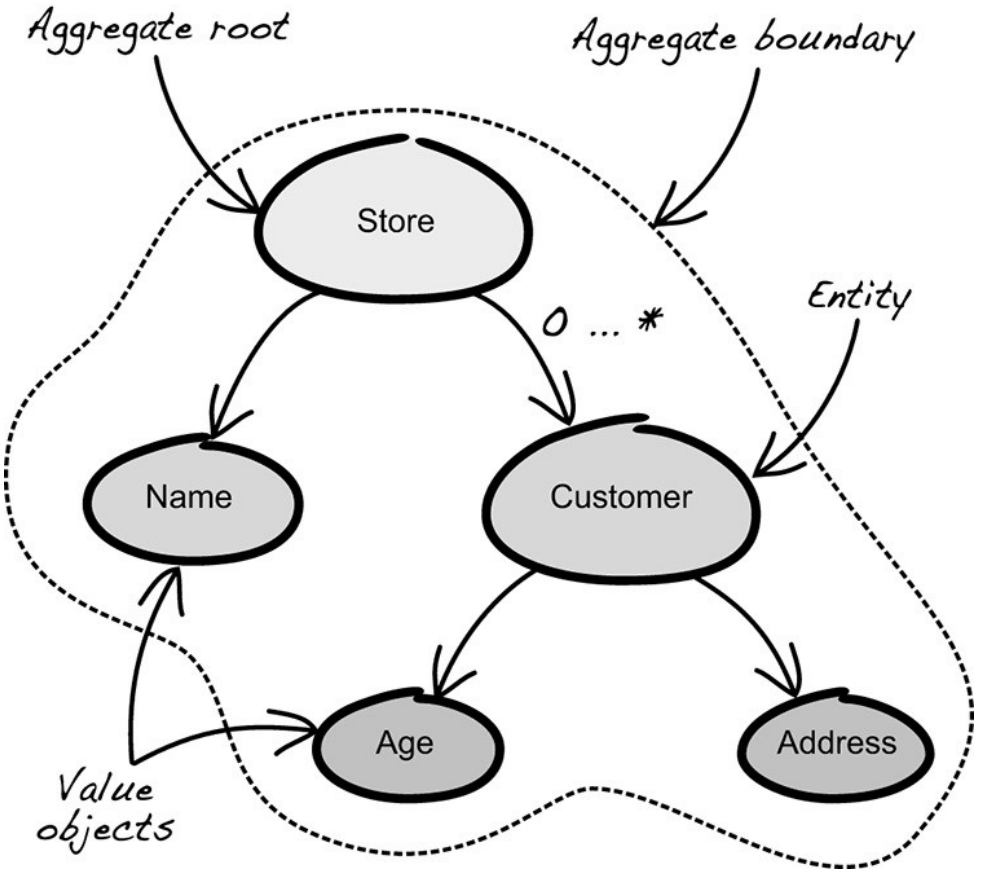
Bisogna concordare i termini da usare nel modello

Introdurre un linguaggio onnipresente (2)



I termini adottati nel modello devono essere usati in modo consistente in tutti i documenti del progetto: specifiche, diagrammi, codice, manuale, database, file di log, ...

I costrutti del modello



- Entità
- Oggetti valore
- Aggregati
- Contesti limitati

Entità (1)

- Ha un’identità (tipicamente un identificatore) che la definisce e la distingue dalle antre entità
- L’identità è consistente durante tutta la vita dell’entità
- Può contenere altri oggetti, come altre entità o valori oggetto
- È responsabile per la coordinazione delle operazioni sugli oggetti posseduti

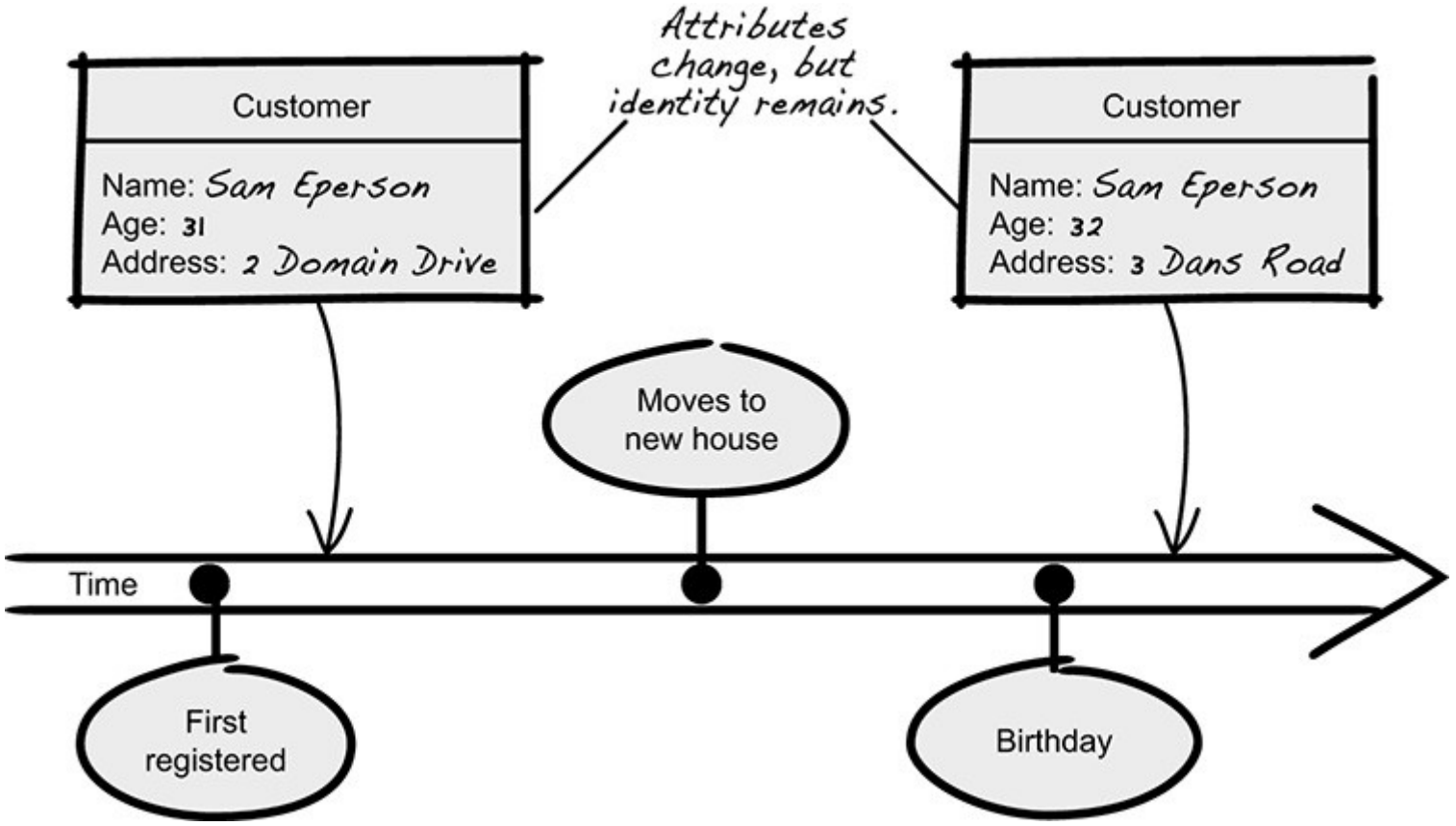
Il confronto fra due entità considera solo l’identità

Due entità sono uguali se hanno la stessa identità, anche gli altri attributi sono diversi

Entità (2)

- Il confronto fra due entità considera solo l'identità
- Due entità sono uguali se hanno la stessa identità, anche se gli altri attributi sono diversi
- Ad esempio, un'automobile
  - Molte parti possono cambiare nel tempo
  - Ma l'automobile è sempre quella
  - La identifichiamo con il numero di telaio o la targa
- L'identificatore potrebbe essere univoco solo in alcuni contesti
  - Globale vs locale
  - Tipicamente, si usa un intero progressivo autogenerato

Entità (3)

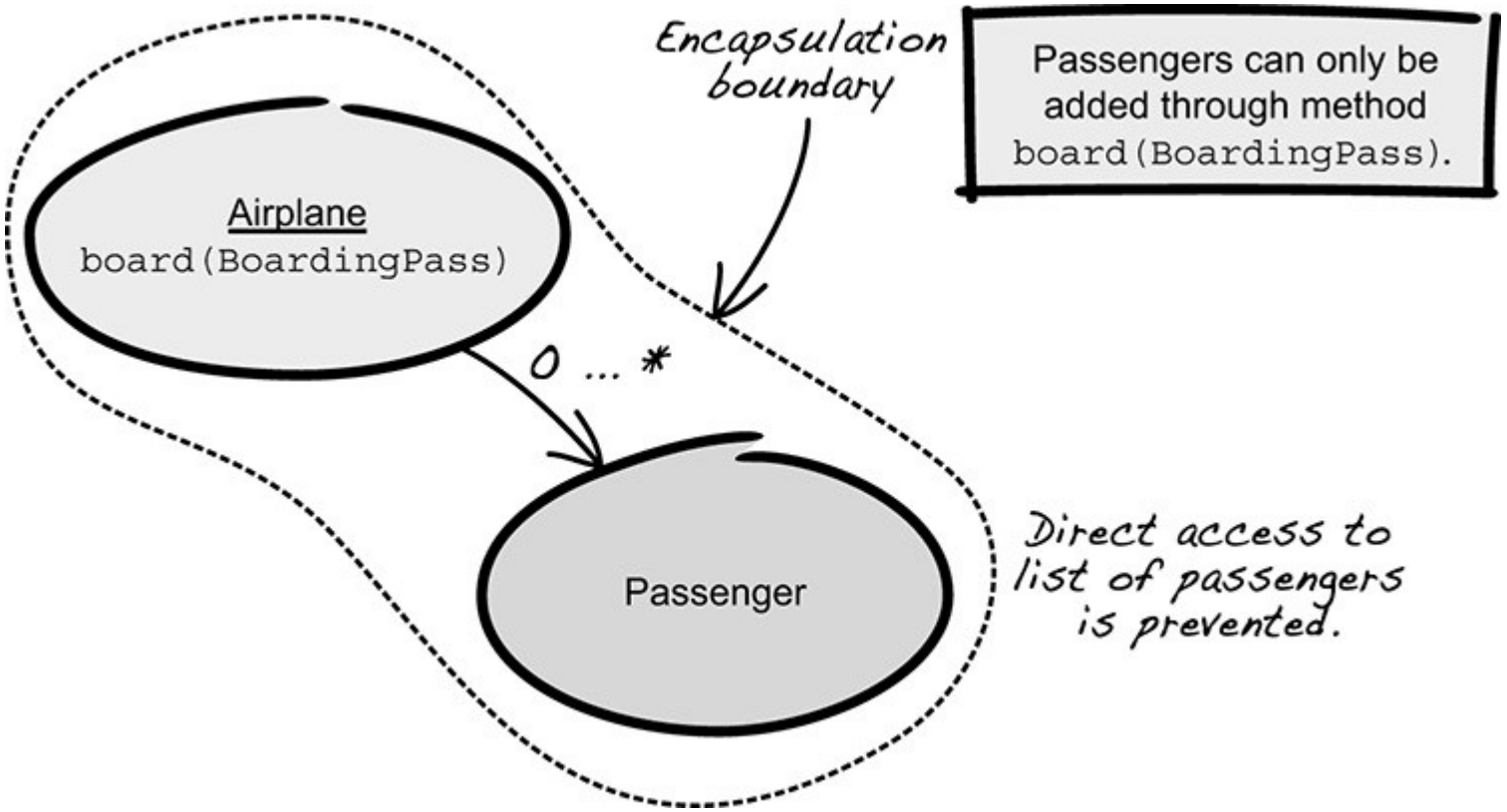


Potremmo usare il codice fiscale come identificativo, ma…

Se ho inserito un CF sbagliato?

Se devo inserire uno straniero senza CF?

Entità (4)



L'entità Airplane è responsabile per l'aggiunta di Passenger (entità posseduta)

Solo in questo modo possiamo garantire invarianti del sistema

Object value (1)

- Non ha un’identità, è definita dal suo valore
- È immutabile
- Dovrebbe formare un insieme concettuale
- Può riferire entità (non contenerle)
- Definisce e impone vincoli importanti
- Può essere usato come un attributo di entità e altri oggetti valore
- Può essere di breve durata

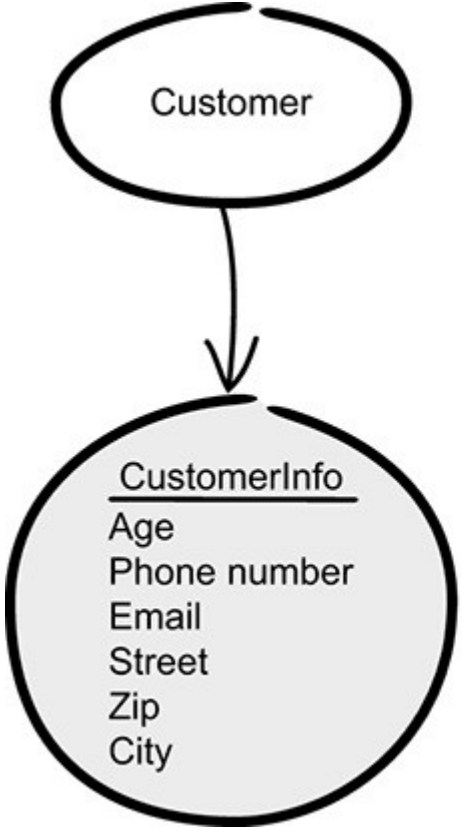
Object value (2)

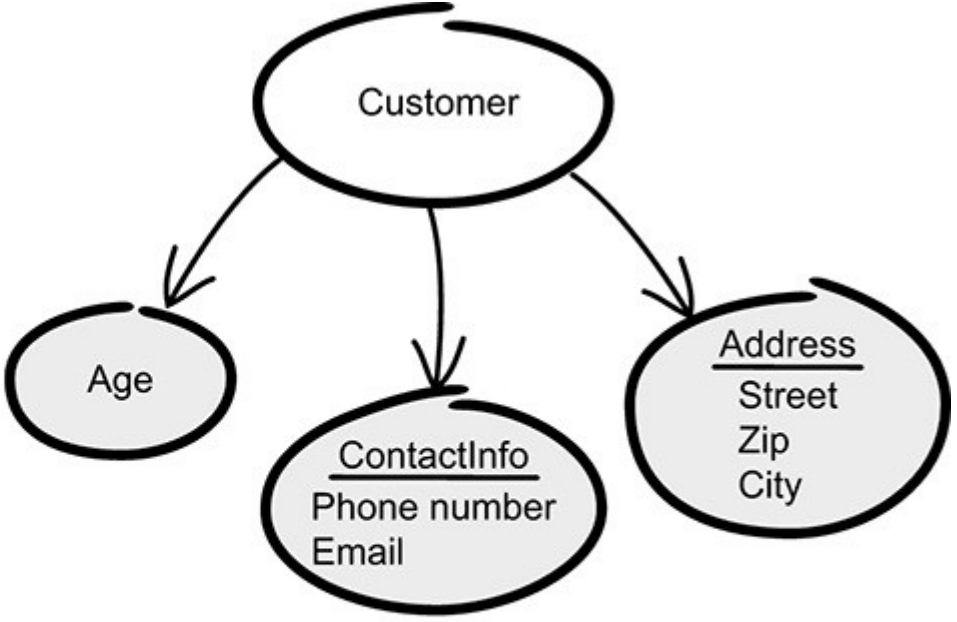
- Definito dal suo valore
  - Una banconota da 5 euro, per quanto mi riguarda, è uguale a tutte le altre banconote da 5 euro
  - Non faccio neanche molta differenza fra due banconote da 5 euro e una banconota da 10 euro: sono 10 euro!
  - Per la Banca Centrale Europea è diverso: ogni banconota è diversa, ha un numero di matricola

Object value (3)

- Immutabile
  - Nelle entità l’identificativo è immutabile
  - Gli oggetti valore sono identificati dal proprio valore
  - Il valore non deve poter cambiare

Value object (4)



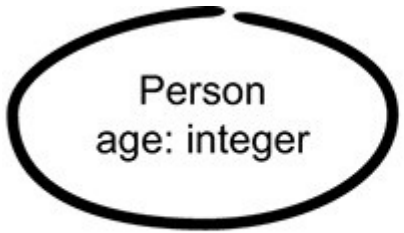


Un insieme concettuale, non un semplice raggruppamento di attributi

La modellazione sulla destra chiarisce meglio quali sono gli oggetti del dominio

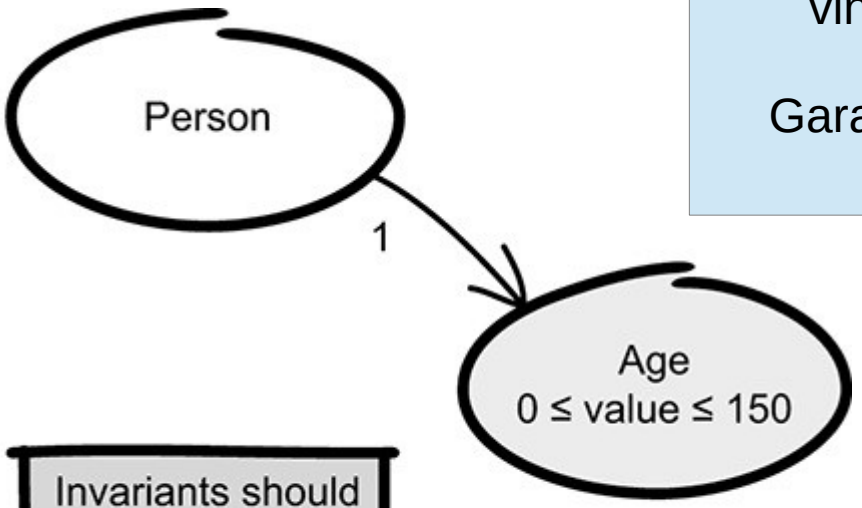
Object value con un solo attributo non sono poi così rari

Value object (5)



age can be
$[-2^{31} \rightarrow 2^{31} - 1]$

Age is a primitive and is missing invariants.



Invariants should be enforced by the value object.

Definisce e impone vincoli importanti

Garantisce invarianti del dominio

Aggregato

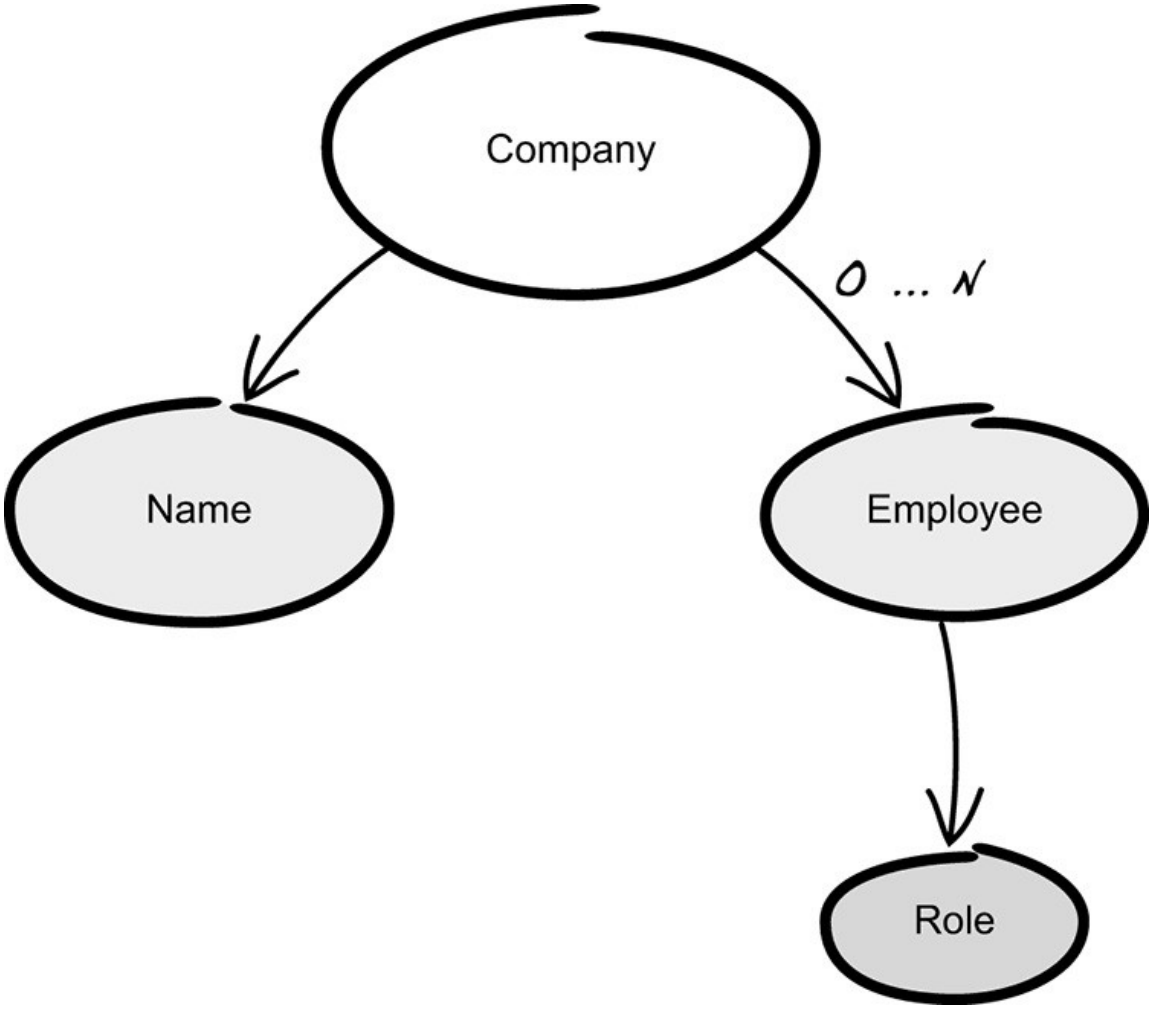
- Un confine concettuale usato per raggruppare parti del modello
- L’aggregato è trattato come una unità durante i cambi di stato del sistema
- Non viene scelto a caso o su basi tecniche
- Viene selezionato con cura dalla comprensione del modello

Aggregato: regole (1)

- Ha un confine e una radice
- La radice è una specifica entità contenuta nell’aggregato
- Solo la radice può essere riferita persistentemente all’esterno dell’aggregato
  - La radice ha identità globale
  - La radice controlla tutti gli accessi agli oggetti nel confine
  - Le altre entità hanno identità locale
- La radice può passare riferimenti di entità interne ad altri oggetti, ma questi non devono essere conservati
- La radice può passare riferimenti di value objects interni ad altri oggetti

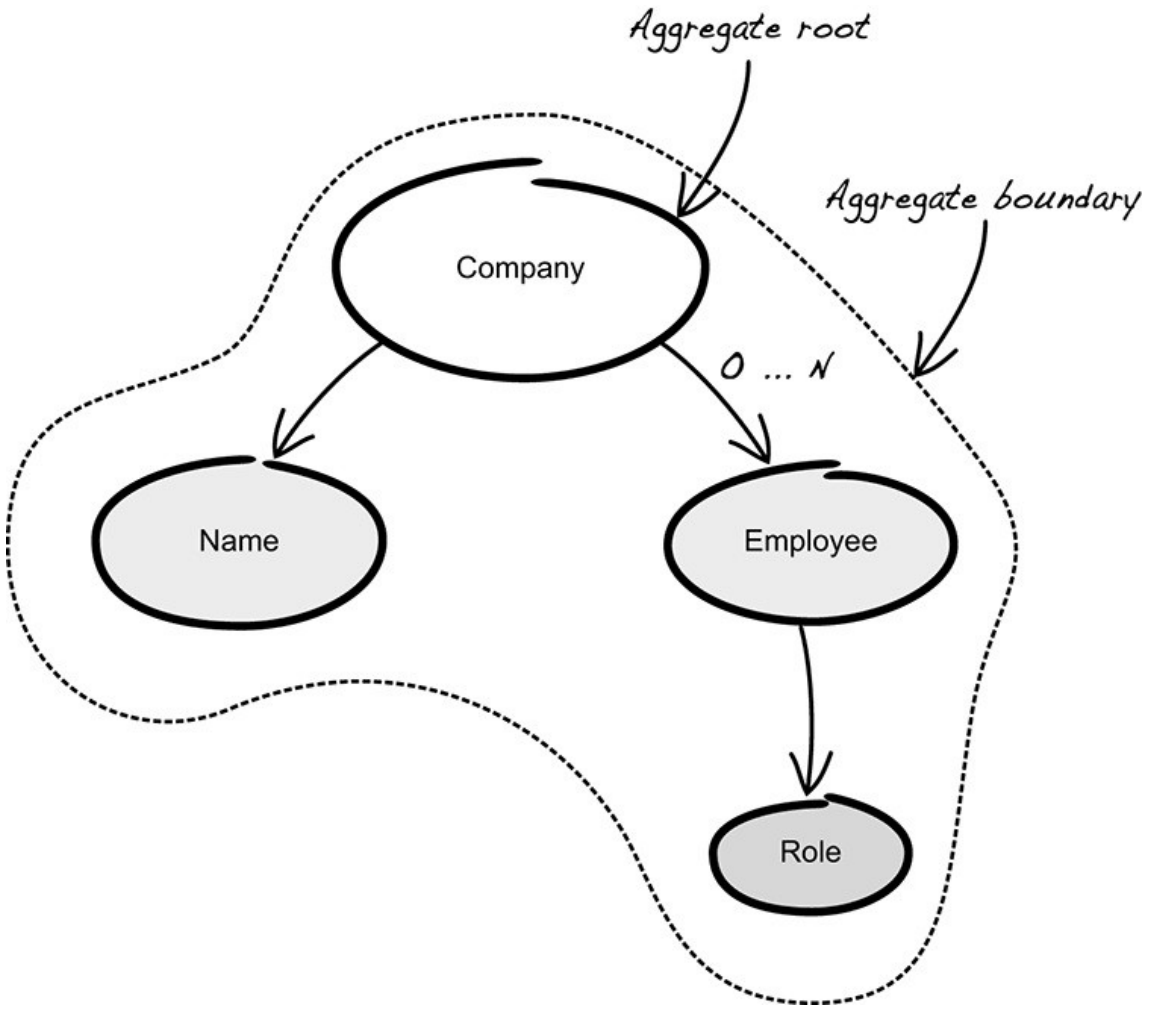
Aggregato: regole (2)

- Invarianti fra i membri dell’aggregato sono sempre imposti a ogni transazione
- Invarianti che riguardano più aggregati non possono essere sempre consistenti, ma lo devono diventare (eventually consistent)
- Oggetti dell’aggregato possono avere riferimenti ad altri aggregati (ovvero alle loro radici)



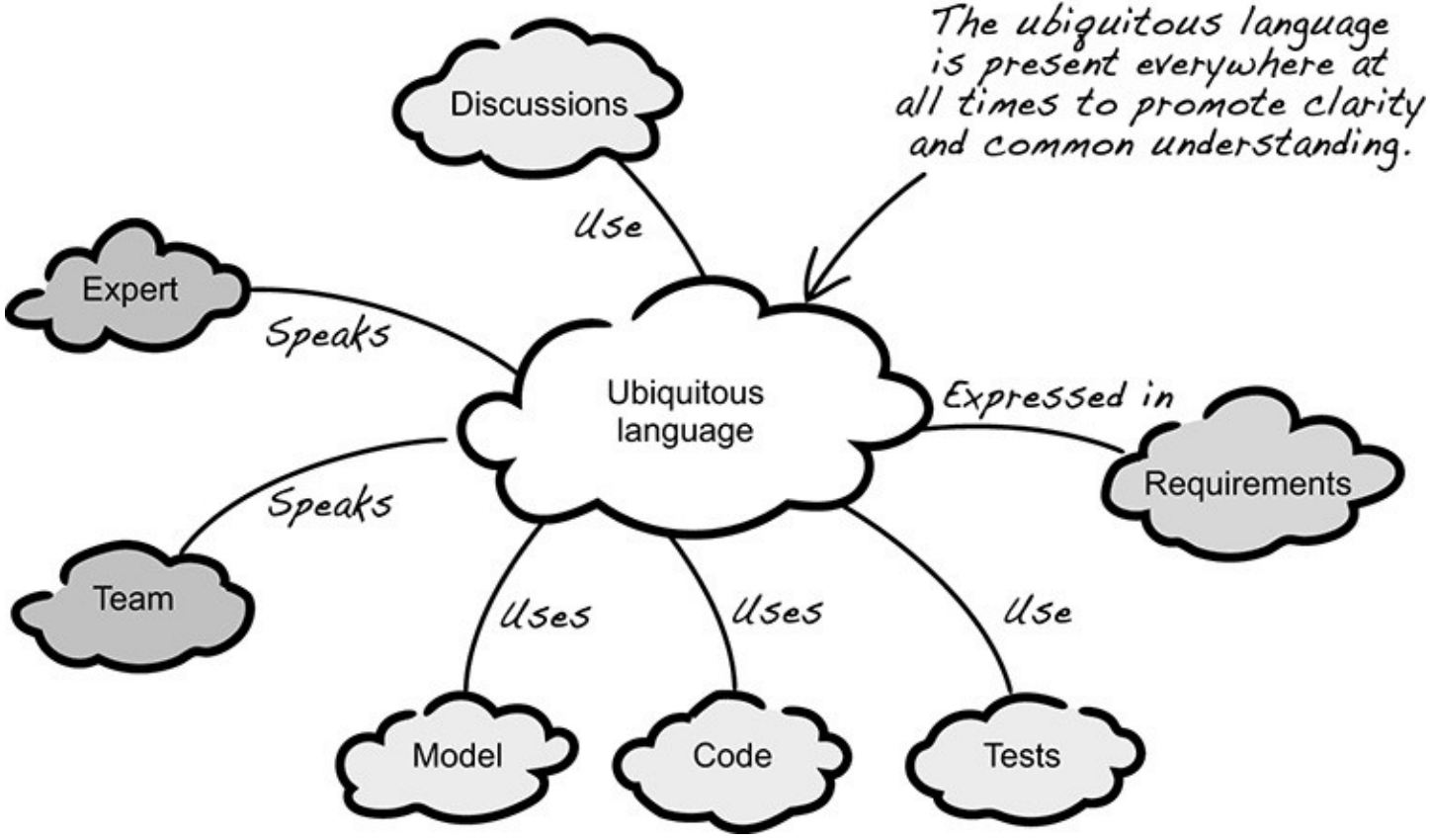
Example

Our example model consists of a company and its employees. We'll make **the company an entity** because it has a clear identity and, because our system can handle many companies, it also needs to be **globally identifiable**. The company has a name, but **the name is merely a value**, so we'll make it a value object. It also has employees who work at the company. **An employee definitely has an identity**, so it's also modeled as an entity. An employee always belongs to a company, so it becomes a child entity of the company. Each employee will have a specific role, but that's also a value, so it becomes a value object.



After discussing the nature of an employee together with the domain experts, you realize that **an employee doesn't have to be identifiable outside of the company**. The employee object can have local identity. You also realize that when roles are assigned to employees within the company, **there are certain roles that can only be held by one person at a time**. There can, for example, only be one CTO at any given point. The same goes for manyother roles. **To uphold these required invariants, the company entity should control the assignment of roles to employees.** This leads you to the insight that the company, together with its child objects, should be modeled as an aggregate. You make the company the root of the aggregate.

Contesti limitati



Usiamo un linguaggio onnipresente, ma…

lo stesso termine può avere significati diversi a seconda del contesto

package per il reparto spedizioni
vs
package per sviluppatori

Dove i termini cambiano la propria semantica, lì è il confine fra due contesti limitati

Developer: “What characterizes an order?”

Expert: “Well, an order contains products that are sellable and nonsellable items.”

Developer: “Not sure I understand. What do you mean by nonsellable products?”

Expert: “Nonsellable products are items that are bundled with sellable products when shipped as a package to their destination.”

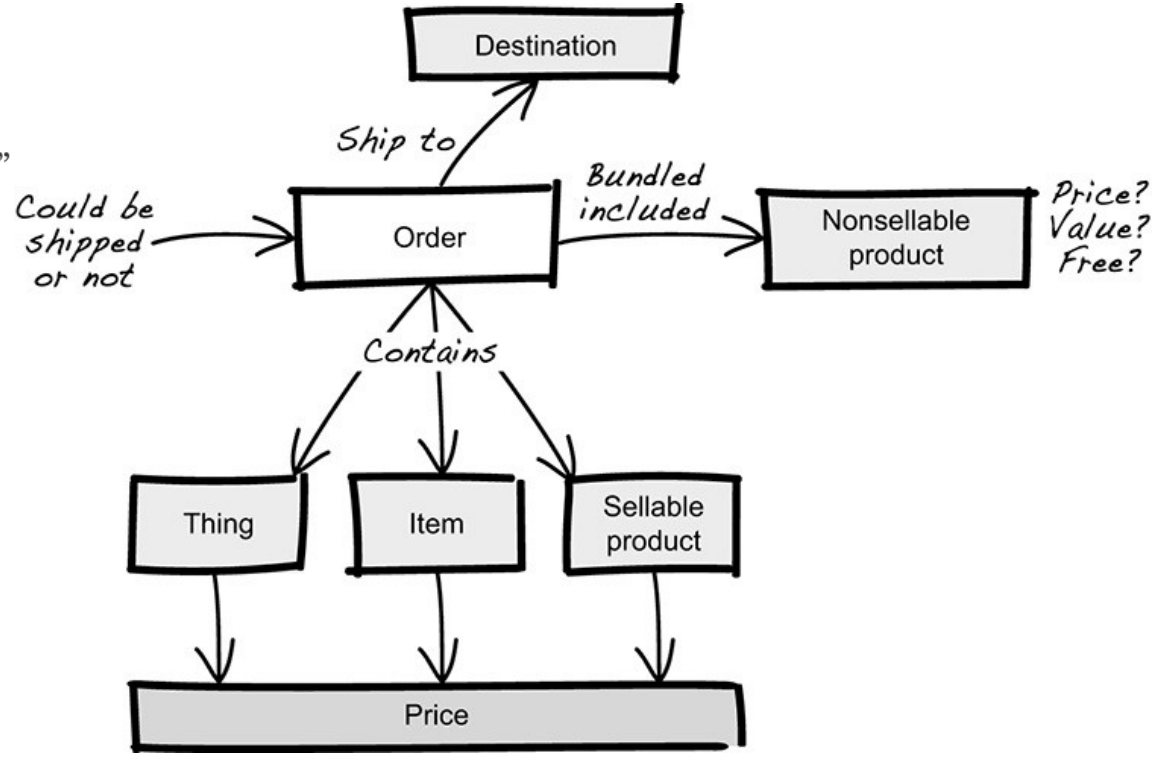
Developer: “Oh, I see. Nonsellable items are products without a price?”

Expert: “No no, all products have a value, but bundled products have a price of zero, so they get included for free.”

Developer: “Hmm, OK, I guess that makes sense.”

C’è molta confusione: non è chiaro cosa è un prodotto, un oggetto, una cosa

Possiamo definire un modello,
ma è necessario approfondire la discussione



Developer: “I’m a bit confused about the terminology. Could we agree on using some of the terms?”

Expert: “Sure, any particular ones in mind?”

Developer: “It seems we only have products—is it OK to stop using words like items, things, nonsellable, and sellable?”

Expert: “OK, that makes sense. From now on, we’ll use the term product for all of these.”

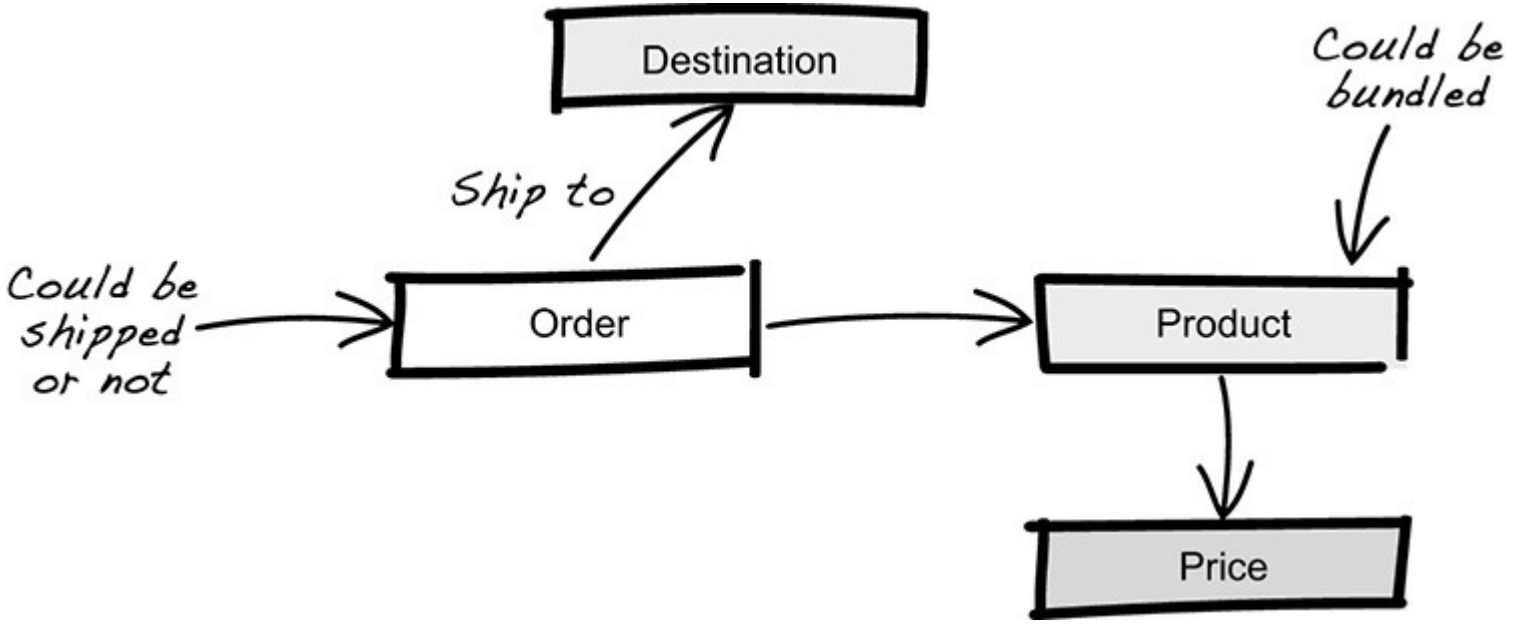
Developer: “Included and bundled mean the same thing, right?”

Expert: “Yes, so let’s only use bundled.”

Developer: “What about price and value?”

Expert: “Same thing. Let’s use price.”

Developer: “Why do we need to care whether a product is free or not?”Expert: “You’re right. We don’t. Let’s not use free.”



La discussione porta ad identificare termini univoci

Il modello diventa più semplice e preciso

Quanti prodotti in un ordine?

Developer: “An order can have one or more products?”

Expert: “Yes, that’s correct. But an order without products isn’t much of a package.”

Developer: “Package?”

Expert: “Oh, sorry. Yes, a 'package' is what we call the box in which we ship everything.”

Developer: “OK, makes sense. But how do we know how many products we need to ship in a package?”

Expert: “Well, the quantity of each product is specified in the order.”

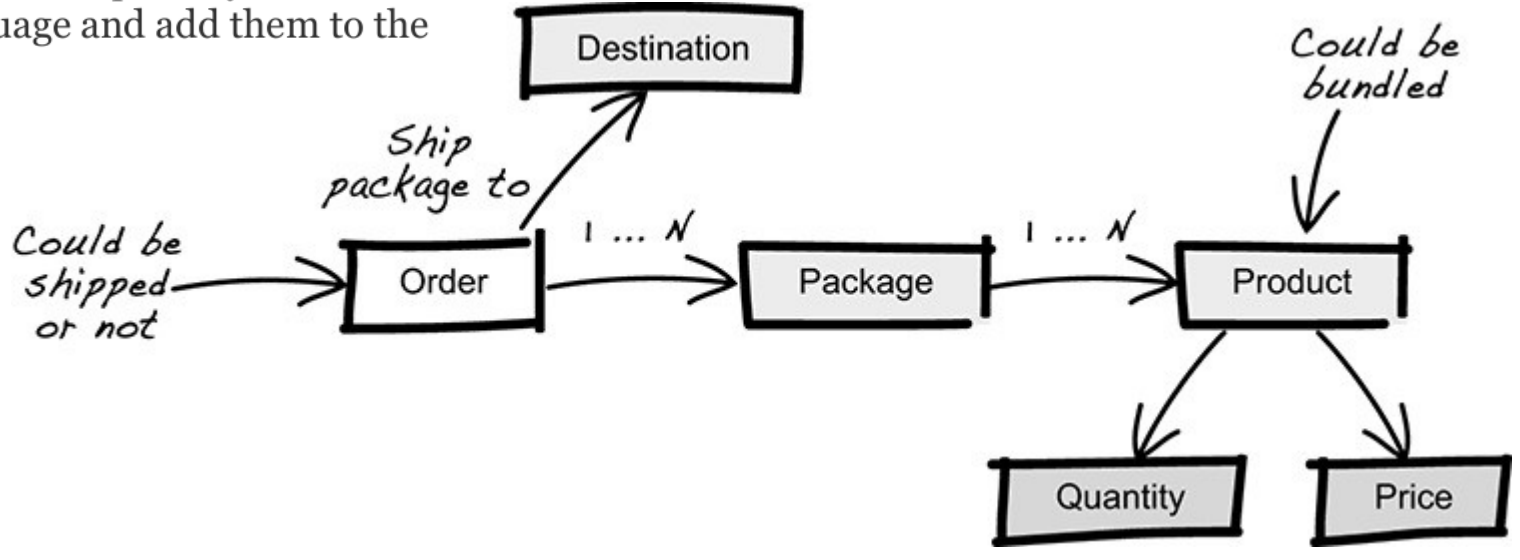
Developer: “Ah, I see. Let’s introduce 'quantity' and 'package' in our ubiquitous language and add them to the model.”

La discussione rivela nuovi termini

Accordarsi sull’uso di questi termini nel modello

Se esperto e sviluppatore sono d’accordo, chiedere agli altri dipartimenti

Il confine del contesto è dove il modello non funziona più



Developer: “Could you please have a look at our model of an order?”

Finance Expert: “Sure. The model makes sense, but you seem to miss a lot of important concepts.”

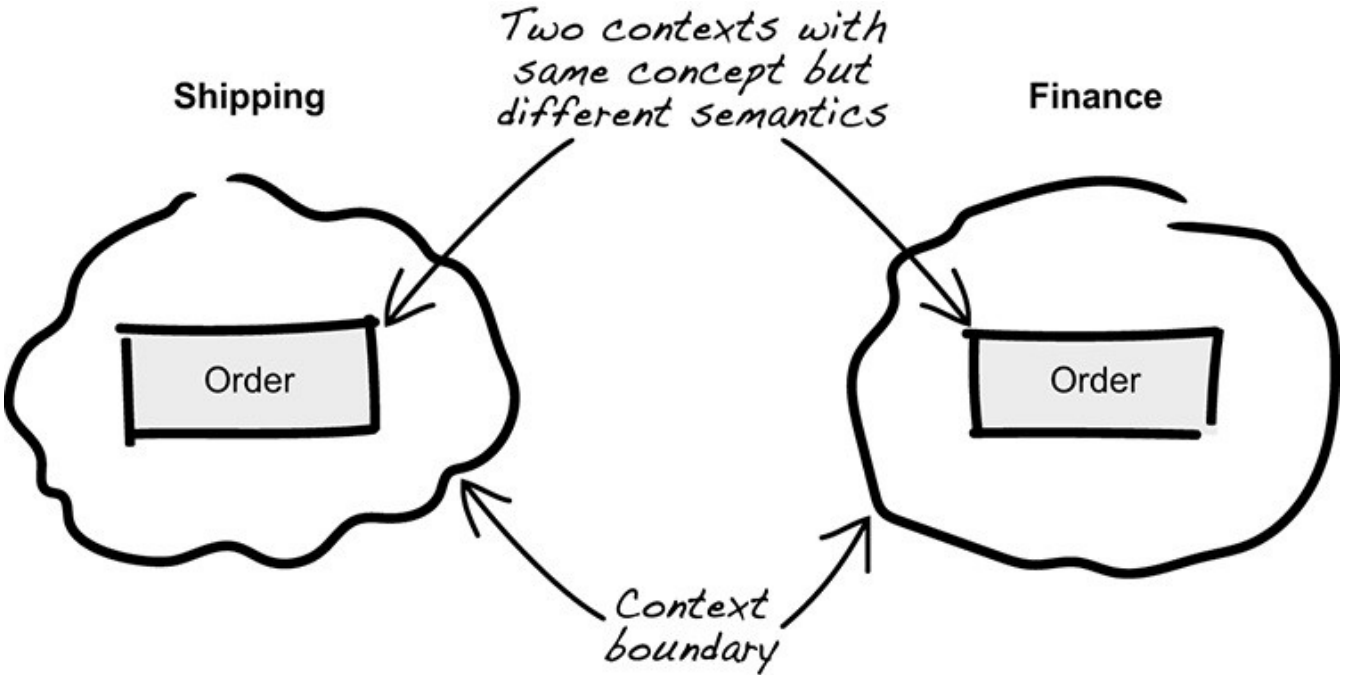
Developer: “Really? Please explain.”

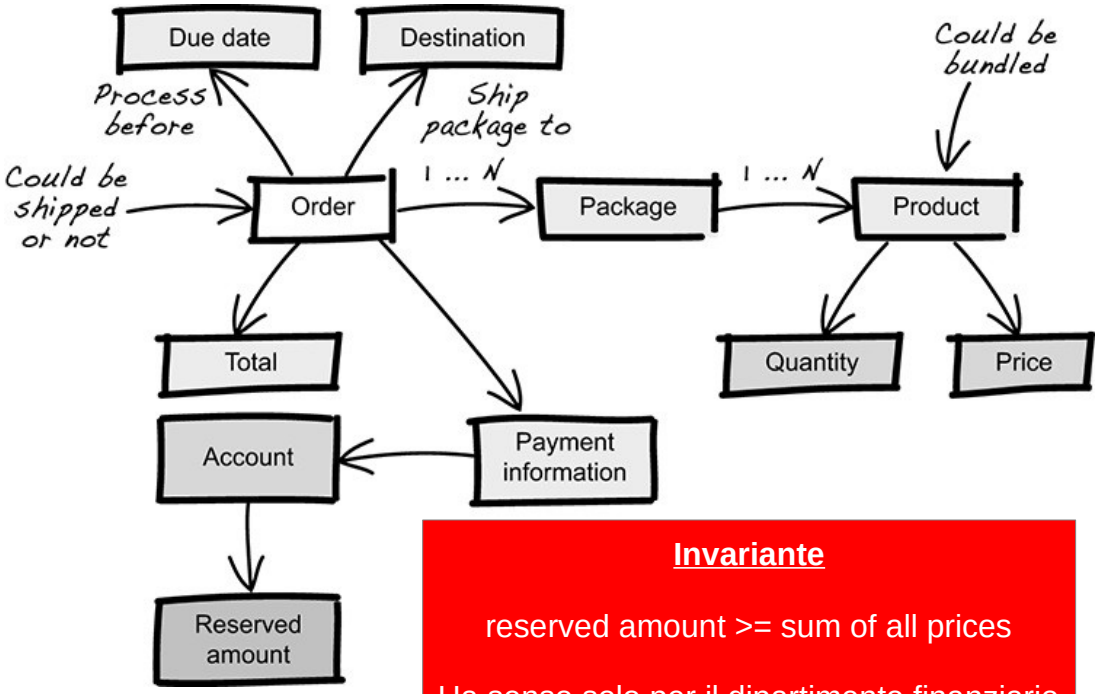
Finance Expert: “The payment information and due date are missing. Also, the reserved amount doesn’t seem to be represented.”

Developer: “Aha. We seem to have a different definition of an order. Thanks for your time.”

Individuato il confine dei contesti, bisogna capire se è necessaria qualche forma di comunicazione

Bisogna prestare molta attenzione ai dati che attraversano i confini





Invariante

reserved amount >= sum of all prices

Ha senso solo per il dipartimento finanziario

ATTENZIONE

Non insistere su un modello unificato: alcuni concetti saranno spesso non usati; ci sarà difficoltà con nuovi requisiti

In questo caso, per le spedizioni è importante che il prezzo dei prodotti rifletta il loro valore (per la dogana)

Il dipartimento finanziario vuole invece sapere il prezzo di vendita

Expert: "To simplify customs declarations for international shipments, we need to list the actual value of a package."

Developer: "OK. So how should we treat bundled products?"

Expert: "Well, previously we made the product free by faking the price by setting it to zero, but that's no longer OK."

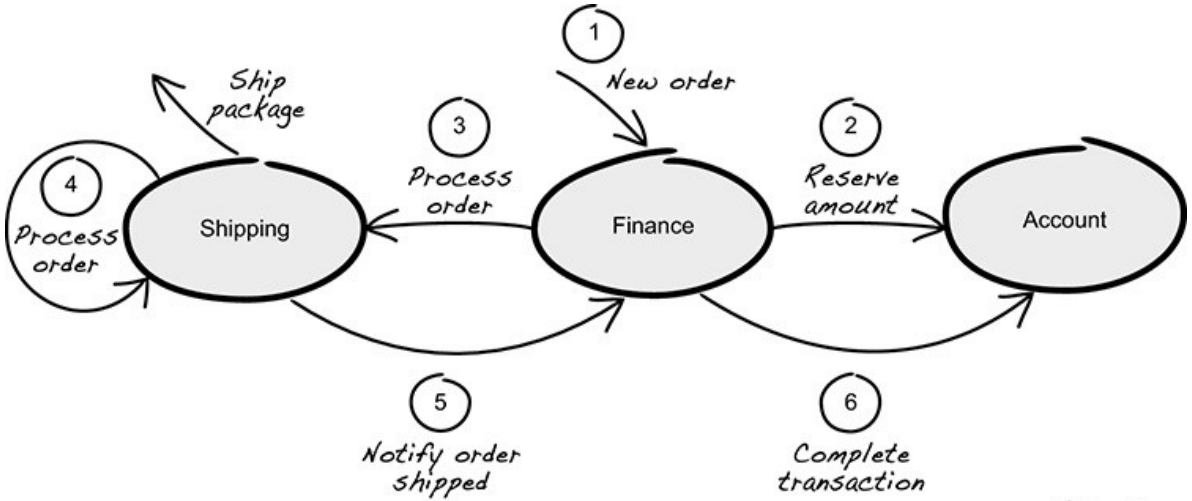
Developer: "Right, so is it OK to just remove the faked price?"

Expert: "Yes, the sum of all prices is the actual value of the package, so that should work."

Developer: "And then we deduct the bundle prices from the total, right?"

Expert: "No, the reserved amount is what's charged by Finance, so we don't need to deduct anything."

Developer: "OK, that makes sense. I'll only remove the faked price then."



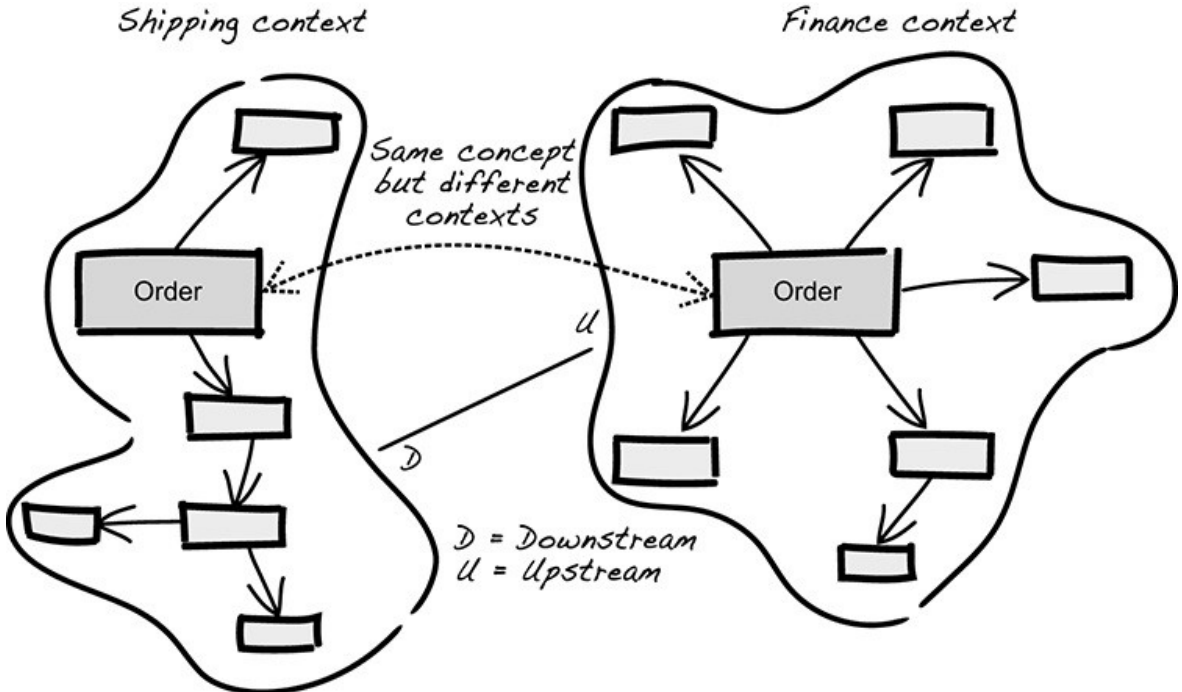
Cerca di capire come come è strutturata la comunicazione fra i vari dipartimenti

Rifletti questa comunicazione nei contesti del modello

Il contesto del dipartimento spedizioni è a valle del contesto del dipartimento finanziario

L'ordine del dipartimento finanziario viene convertito in un ordine del dipartimento spedizioni

La mappa contestuale sulla destra chiarisce questo flusso



Google Forms and Reading

- Rispondete ai moduli online su DDD

  https://forms.gle/eyMP4MzPxg8GE7yp8

  https://forms.gle/RzJmuMSLTaWgx1sc8
- Reading sui diagrammi snapshot (istantanee)

  http://web.mit.edu/6.031/www/fa18/classes/02-basic-java/

  (sezione **Snapshot diagrams**)

Fine della lezione