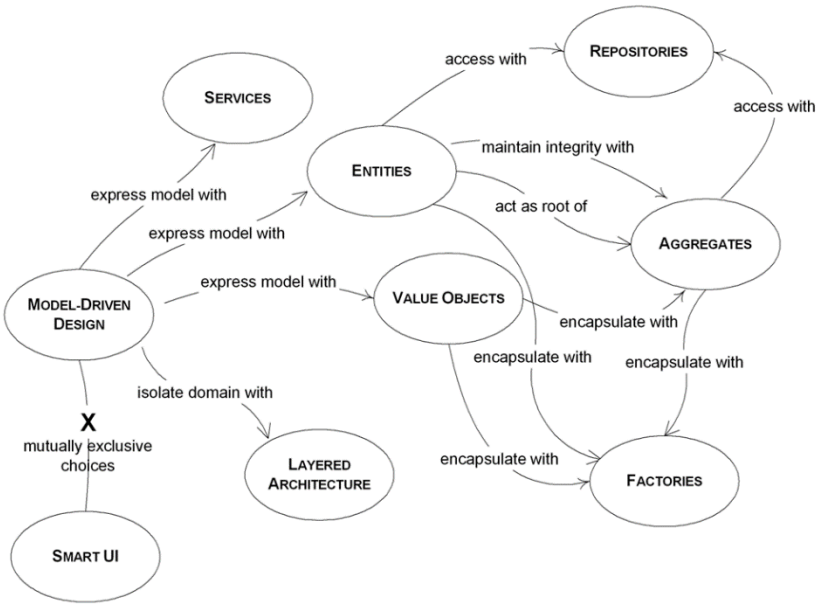
Domain Driven Design – Entity

Introducere în Domain Driven Design(DDD)

Introdus de programatorul Eric Evans în cartea *Domain-Driven Design: Tackling Complexity in the Heart of Software*, conceptul DDD este un mod de abordare a sistemelor software complexe, punându-se accentul pe ideea de *domeniu* sau sferă de activitate în care se implementează o aplicație software. Design-ul unui astfel de sistem se realizează pe baza unui *model* al domeniului, caracterizat prin proprietăți și comportament și care mai departe duce la definirea unui *limbaj universal* ce ajută la scrierea codului într-un mod cât mai ușor de înțeles și care să reflecte realitatea domeniului aplicației.



Modelarea DDD se bazează pe următoarele elemente de bază:

- Entities
- Value Objects
- Aggregations + Roots

Conceptul de Entity

Conform definiției, un *entity* reprezintă un obiect care, spre deosebire de obiectele standard din OOP, nu este definit prin atributele sale, ci mai degrabă printr-un fir de continuitate și prin identitatea sa unică în sistem. Un entity este instanțiat într-o componentă specială a modelului numită *factory* unde e încapsulată logica creerii obiectelor, iar pentru a spori securitatea sistemului toate atributele sale sunt read-only, astfel că modificările se pot face doar prin constructori sau metode. De asemenea, entity-urile sunt stocate și accesate într-o altă componentă numită repository(diferit față de noțiunea de repository din sistemele de versionare), care acționează ca un serviciu în care se pot face query-uri de date. Folosirea unui repository ajută la eliminarea logicii de acces din logica de business a modelului.

Exemplu care ilustrează ideea de entity:

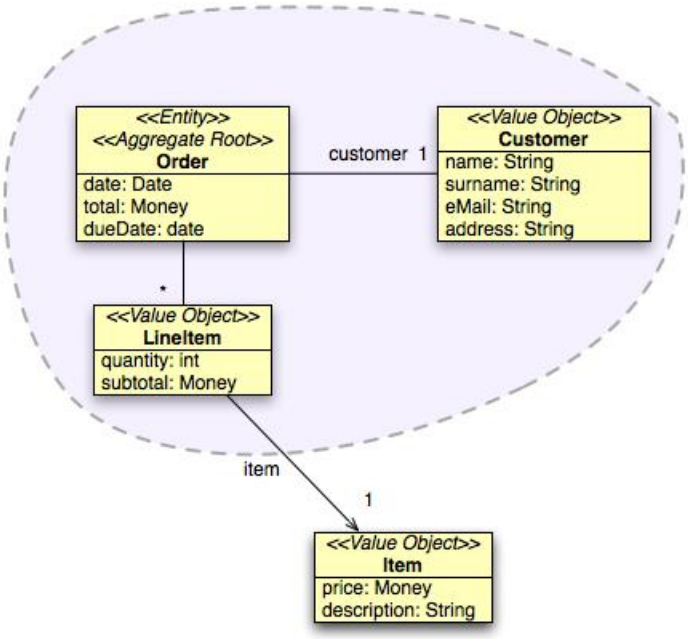
Dacă luăm o abstractizare a unui cumpărător, acesta e definit prin atribute precum nume, adresă, vârstă, dar are și un identificator unic(ex. CNP). Putem avea doi cumpărători cu același nume sau adresă, dar ce e cu adevărat important este identificatorul unic prin care se diferențiază instanțele entității de cumpărător în sistem.

Entities/Value Objects/Aggregates

În modelarea sistemului, este important să diferențiem un entity de un value object, deși ambele sunt în strânsă legătură. Dacă o entitate este definită prin identificatorul unic, un value object e caracterizat prin atributele sale, el neavând o identitate proprie sau posibilitatea de a fi modificat. De altfel, un value object nu este constrâns de ciclul de viață al unei entități. Cu alte cuvinte, un value object poate fi înțeles ca o caracteristică a unui entity.

Exemplu:

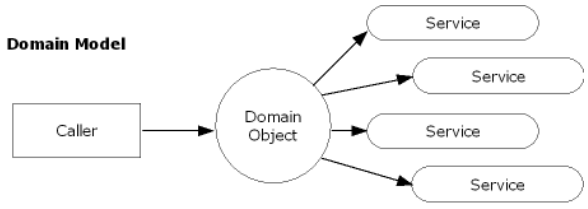
Atunci când căutăm o anumită persoană după nume(care poate fi văzută ca o entitate), numele poate fi văzut ca un value object care nu are o identitate proprie, ci doar caracterizează acea persoană.

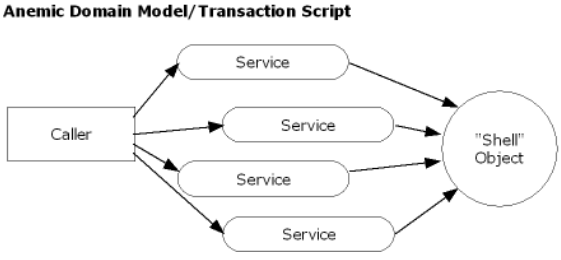


Atunci când avem mai multe entități relaționate unele cu altele iar modelul devine din ce în ce mai complex, putem simplifica comunicarea prin stabilirea de aggregates ce pun la un loc entitățile cu value objects aferente. Fiecare aggregate are un root entity cu identitate globală(fiecare root are asociat câte un factory) care este singurul punct de acces cu exteriorul, modelul fiind astfel mult simplificat.

Anemic Entities

Un domain model anemic este acela caracterizat prin separarea complete a logicii de business(ex. calcule, reguli) de cea de date. În cazul acesta, entitățile definite devin simple containere de date, având implementate doar gettere și settere pentru câmpurile lor, iar întreaga logică de business este implementată în clase separate precum services. Deși bun pentru aplicații simple, acest model încalcă principiile OOP cum ar fi încapsularea datelor.

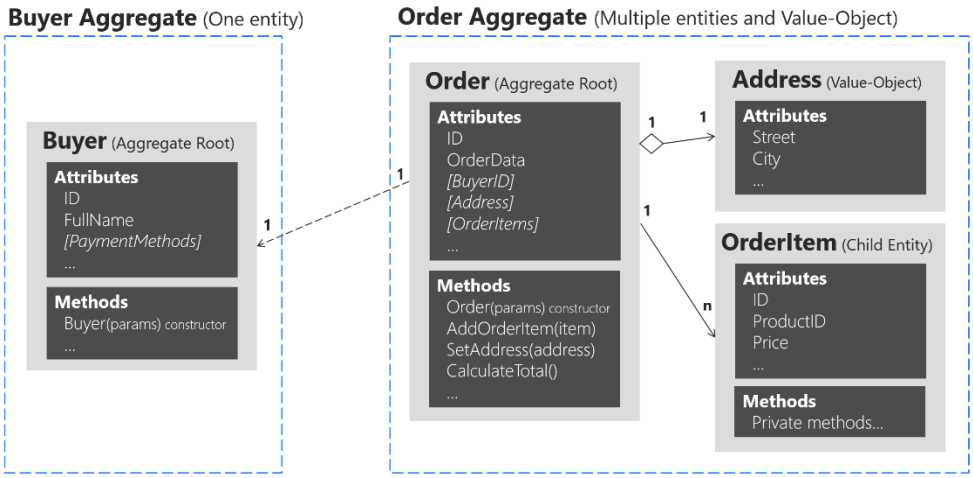




Aplicarea unui model DDD(rich domain model) pentru entități presupune includerea unei părți din logica de business în entități și value objects grupate eventual în aggregates, urmând ca în partea de services să includem numai logica ce operează pe mai multe părți din domeniul aplicației.

Studiu de caz

Pentru a ilustra conceptele prezentate, putem analiza un model DDD simplu al unui magazin online.



Întrucât avem mai multe entități relaționate, au fost create două aggregates care definesc logica de cumpărător și cea de comandă, fiecare cu câte un root care comunică reciproc. Între aceste root-uri se stabilește o relație 1-la-1, în sensul că fiecare comandă este plasată de câte un cumpărător. Atât root-urile, cât și entitatea copil *OrderItem* sunt identificate unic printr-un ID și au constructori/metode care ajută la implementarea logicii aplicației. Obiectul *Address* nu are o identitate proprie, ci acționează ca atribut al entității *Order*, deci este un value object relaționat cu aceasta. Se poate observa că modelul reprezentat nu este unul anemic, întrucât entitățile implementează în metode măcar o parte din logica de business(ex. metoda *CalculateTotal()* a entității Order).

Concluzii

Un aspect cheie al abordării software de DDD este conceptul de entity, întrucât stă, alături de value objects și aggregates, la baza modelării unui domeniu într-un mod cât mai ușor de înțeles.

Un model DDD bine conceput presupune evitarea entităților anemice, punerea cât mai mult în valoare a principiilor OOP precum încapsularea, urmărindu-se în final aplicații cât mai modulare, flexibile și ușor de înțeles fără a folosi aspecte mult prea tehnice.