**Domain Driven Design – Aggregation Root**

Aggregation Root reprezintă punctul de acces într-un cluster de obiecte asociate care pot fi tratate ca o singură unitate. Cu alte cuvinte aggregation root este sigurul punct de comunicare cu exteriorul, fiind global accesibil în aplicatie. În interiorul clusterului vom avea toate obiectele asociate ale acestui agregat.

**Reguli pentru implementare:**

* Entitatea rădăcină are identitate globală;
* Entitățile rădăcinilor au identitate globală. Entitățile din interiorul graniței au identitate locală, unică numai în cadrul clusterului;
* Nimic din afara “graniței” nu poate face referire la nimic din interior decât prin entitatea rădăcină. Entitatea rădăcină poate să face referire la entitățile interne altor obiecte, dar le poate folosi numai temporar (într-o singură metodă sau bloc);
* Numai rădăcinila poate fi interogata dintr-o baza de date;
* Obiectele pot face trimiteri la alte rădăcini;
* Operația de ștergere trebuie să înlăture totul în interiorul “granitei” agregatului;
* Atunci când este efectuata o schimbare a oricărui obiect din granița agregatului, toți invarianții întregului agregat trebuie satisfăcuți;

**Beneficiile utilizării agregatelor**

1. **Interfață simplă**

Primul mare beneficiu al agregatelor este că reduce foarte mult interfața pe care o aveți la un anumit segment al aplicației.

Prin faptul că exista un singur punct de intrare , se poate restricționa accesul la obiectele interne și se poate menține o armonie în interiorul clusterului. Având un singur punct de intrare, de asemenea, este mult mai ușor lucrul cu codul, deoarece nu există nicio ambiguitate a modului în care ar trebui să fie accesate obiectele copil ale unui obiect-parinte.

1. **Menținerea invarianților**

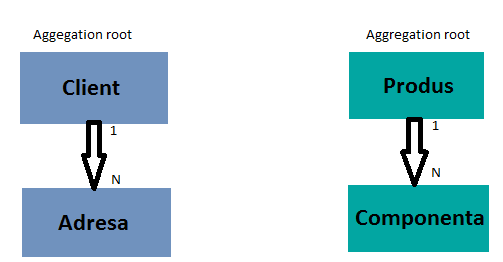
Frumusețea reală a aplicațiilor Domain Driven este capacitatea de a modela regulile de business ale aplicației prin invarianți. Un invariant este o regulă care trebuie să rămână consecventă. Menținerea invarianților unei unități de obiecte ar fi dificilă fără constrângerea limitei.

1. **Eliminarea obiectelor**

O problemă obișnuită atunci când se dezvolta aplicații web este eliminarea în siguranță a obiectelor. Atunci când nu avem constrângeri în jurul obiectelor care pot fi asociate cu alte obiecte, acest lucru poate începe să devină înfricoșător.

Folosirea agregatelor elimină această problemă deoarece toate obiectele aflate în limita agregatului nu pot fi asociate cu nici un obiect în afara limitei.

Agregatele sunt defapt formate din entitati si atribute value objects care sunt tratate ca o singura unitate în vederea efectuarii modificarilor. Trebuie sa luam în considerare consistenta intregului agregat înainte de a efectua modificari. În exemplul prezentat mai jos adresa poate fi accesata doar folosind rădăcina client(care este aggregation root), iar componenta doar prin produs(care este de asemenea un agrregation root).



Este posibil sa avem doar o singură entitate în agregat, iar daca acest lucru se întampla acea entitate va fi aggregation root. În unele cazuri agregatul poate depinde de niste reguli care afecteaza consistenta datelor. Spre exemplu produsul este constituit din mai multe componente, dar pentru a se afla într-o stare valida trebuie sa fie format dintr-un set specific de componente, nu orice fel de componente si nu oricate. Daca lăsăm componentele sa fie modificate independent de rădăcina produs cu care au fost asociate, acest lucru poate determina probleme de consistentă. Orice modificare trebuie să respecte principiile ACID(Atomic, Consistent, Isolated, Durable). De asemenea este responsabilitatea rădăcinii să mentină invariantii, cum ar fi numărul si tipul componentelor ce formeaza produsul. Un invariant este o conditie care trebuie sa fie tot timpul adevarată pentru a nu fi afectata consistenta sistemului.

Pentru a putea decide daca o entitate trebuie sa fie un aggregation root, trebuie sa ne punem problema daca stergerea acestuia ar trebui sa determine stergerea în cascadă a entitatilor in ierarhia agregatului. Daca da, atunci entitatea aceea ar putea fi un aggregation root. Un alt mod pentru a ne da seama daca avem nevoie de un aggregation root este a ne pune intrebarea daca are sens sa avem entitati fara a avea o entitate “parinte”. Spre exemplu putem avea componente standard, dar de asemea avem si componente specifice unui produs care trebuiesc referite folosind entitatea produs.

**Concluzie**

Proiectarea aplicației dumneavoastră ca un set de agregate principale vă poate clarifica procesul de gândire atunci când construiți aplicații complexe. Agregatele captează multe nuanțe ale unei asociații între două obiecte, astfel încât nu se pierd în complexitate.

Desigur, vor exista mereu modalități de a eluda constrângerile pe care un agregat le pune în practică. Dar cred că procesul de identificare și modelare a aplicației în termeni de agregate poate ajuta într-adevăr să distileze problema pe care încercați să o rezolvați.

Utilizarea agregatelor face de asemenea mult mai ușor luarea anumitor decizii. De exemplu, poate fi dificil să se decidă cine are responsabilitatea de a crea și menține relația dintre entități. Când identificați și implementați agregate, nu trebuie să vă gândiți la această decizie, deoarece modelul ar fi trebuit să vă ghideze deja la răspunsul corect.

# <https://www.culttt.com/2014/12/17/aggregates-domain-driven-design/>

<http://www.methodsandtools.com/archive/archive.php?id=97p2>

<https://lostechies.com/jimmybogard/2008/05/21/entities-value-objects-aggregates-and-roots/>