Sebastian HIRTIE
Anul IV, AC-IS

Tema 2 – PSSC

DDD – Aggregation root

Initial introdus de catre programatorul Eric Evans in cartea sa din 2004 Domain Driven Design:Combaterea complexitatii in inima software-ului, DDD este un indrumator si o solutie prescrisa pentru dezvoltarea sistemelor complexe care interfereaza cu activitati, task-uri si date dintr-un anumit domeniu tehnic. El incearca sa faca din software un model al unui sistem sau process din lumea reala.

DDD este o abordare a design-ului OO, care are ca scop obținerea unui grafic al obiectelor de domeniu în stratul de afaceri al sistemului. - obiectele de domeniu sunt responsabile pentru satisfacerea cerințelor de afaceri la nivel înalt și în mod ideal ar trebui să se poată baza pe volumul de date pentru lucruri cum ar fi performanța și integritatea magazinului de date persistent.

Important aici este sa se inteleaga domeniul problemei pentru a crea un model abstract care poate di implementat intr-un set de tehnologii. DDD ofera sfaturi cum se face acest model sa devina un sistem capabil sa raspunda nevoilor fiind robust in fata schimbarii domeniului.

Lipsa unei intelegeri commune a domeniului intre persoanele tehnice si cele care dezvolta software-ul este una din problemele abordate de aceasta metodologie. De aceea ea implica colaborarea indeaproape cu un expert pentru a intelege bine domeniul si pentru a pastra un limbaj tehnic si cu aceeasi semnificatie astfel încât nevoile reale din domeniul problemei să poată fi descoperite și să fie creată o soluție adecvată pentru a răspunde acestor nevoi.

DDD ajuta la descoperirea arhitecturii de nivel superior si la informarea despre mecanica si dinamica domeniului pe care software-ul il replica. In mod concret, aceasta inseamna ca o analiza bine facuta minimizeaza neintelegerile si reduce numarul de schimbari costisitoare. Prin impartirea complexitatii domeniului in context mai mici se evita proiectarea obiectelor umflate unde se pierde mult timp in elaborarea detaliilor de implementare.

Sunt introduce astfel mai multe concept tehnice :

- Entitati
- Obiect valoare
- Depozit
- Fabrica
- Serviciu
- Radacina de agregare

Un model poate contine un numar mare de obiecte din domeniu. Chiar daca ne straduim la proiectare, se intampla sa existe multe obiecte associate intre ele creand o retea complexa de relatii. Exista mai multe tipuri de relatii: unu-la-unu, unu-la-mai-multi si mai-multi-la-mai-multi care pot fi si bidirectionale. Ele cresc complexitatea foarte mult si administrarea unui astfel de sistem devine dificila. De asemenea trebuie respectate acele norme ori de cate ori datele se schimba. Este dificil garantarea coerentei la modificari la obiecte dintru-un model cu asociatii complexe, de aceea se recomanda folosirea agregatului.

Agregatul este un design pattern esential in DDD, este un grup de obiecte associate ca fiind o singura unitate in ceea ce priveste schimbarile de date. Agregatul este marcat de o granita care separa obiectele din interior fata de exterior. Fiecare agragat are o radacina, ea este o entitate

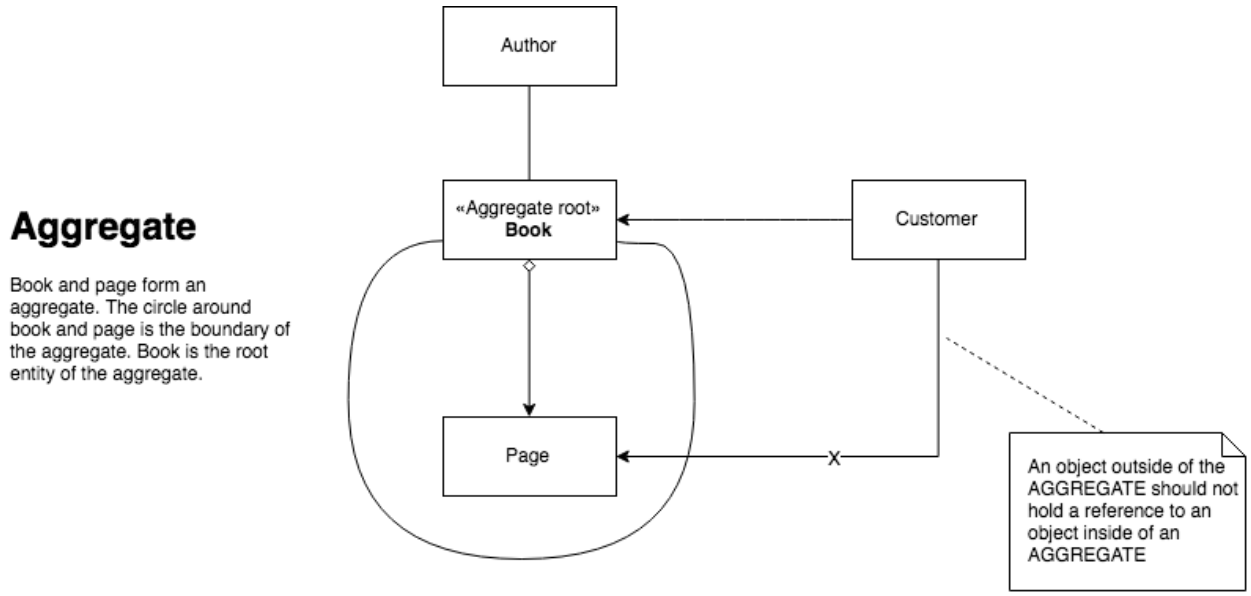
si este singurul care este accesibil din exterior. Radacina este responsabila pentru controlul accesului la membri agragatului. Ea are referinte la oricare din obiectele agregate, obiectele interioare pot avea referinta unele la altele, insa un obiect din exterior poate avea referinta doar la radacina. Daca agregatul este stocat intr-o baza de date, numai radacina trebuie sa fie returnata in urma unei interogari.

Din moment ce o entitate exterioara poate referential doar radacina aceasta inseamna ca entitatea nu poate schimba in mod direct celelalte obiecte din agregat, indeplinind integritatea datelor si respectarea invariantilor. Radacina este accesibila si prin urmare poate fi modificata, schimbarea obiectelor din agragat se face prin radacina insa doar atat cat aceasta permite.

Daca radacina agragatului este stearsa atunci tot restul agregatului este de asemenea sters pentru ca nu va exista nici o alta referinta la el. Este posibil ca radacina sa ofere referinte tranzitorii ale obiectele interne celor din exterior cu conditia ca aceste referinte sa nu fie pastrate dupa incheierea operatiei.

Beneficiile utilizarii DDD si a radacinii agragatului sunt: reducerea substantiala a interfetelor din aplicatie, consecventa regulilor (invariantilor), eliberarea memoriei nefolosite prin stergerea in cascada a obiectelor associate cu o radacina stearsa, simplificarea modelului perceput ca o cutie neagra al carei intrari este radacina.

Studiu de caz: Avem de realiza o aplicatie pentru o biblioteca pentru a administra cartile si imprumuturile lor catre clienti. Pentru aceasta am modelat sistemul ca in figura, ca si radacina se poate alege autorul, clientul sau cartea. Orice carte este formata din mai multe pagini care nu au sens sa existe in afara ei, ele vor fi referentiate prin intermediul unei carti. Fiecare carte trebuie sa fie identificata in mod unic, insa paginile incep sa fie numerotate la fel deci nu sunt unice in sistem, prin urmare ele formeaza un agregat avand ca radacina cartea. Daca o carte va fi eliminata sau stearsa, toate paginile aferente ei vor fi sterse. Fiecare carte are un autor si poate sau nu fii atribuita unui client. Clientul poate citi o pagina doar accesand radacina adica cartea.



Sebastian HIRTIE
Anul IV, AC-IS

In concluzie, perceperea sistemului ca un set de radacini agragate poate clarifica proiectarea si intelegerea sa. Agregatul asocieaza obiecte astfel incat sa nu se piarda in complexitate si inlesneste responsabilitatea si relationarea obiectelor.

Bibliografie:
https://deviq.com/aggregate-pattern/
https://www.culttt.com/2014/12/17/aggregates-domain-driven-design/
https://airbrake.io/blog/software-design/domain-driven-design
https://comment-it.co.uk/ddd-part-4-life-cycle-patterns/