Razvoj vođen domenom (Doman-Driven Design) na primjeru upravljanja stambenim zgradama

Željko Tepšić

Mentor: Prof.dr.sc. Nikola Bogunović

Sadržaj

- Uvod
- Razvoj vođen domenom (DDD)
  - Sveprisutni jezik
  - Građevni blokovi DDD-a
  - Kontinuirano refaktoriranje
  - Omeđen kontekst
  - TDD
- Domena upravljanja stambenim zgradama
- Implementacija
- Zaključak

Uvod (1)

- U informacijskim sustavima, stvarni svijet predstavljamo objektima koji su imenovani i oblikovani prema konceptima s kojima se susrećemo svaki dan.
- Razvoj programske potpore često je povezan sa automatizacijom procesa iz stvarnog svijeta ili sa pružanjem rješenja za poslovne probleme.
- Ta automatizacija procesa i poslovni problemi čine domenu programske potpore.

Uvod (2)

- Programska potpora potječe iz domene te je time i usko povezna s domenom.
- Često puta se previše vremena troši samo na tehnologiju i implementaciju.
- Da bi smo razvili dobru programsku potporu potrebno je znati radi čega se razvija programska potpora.
- Npr. nije moguće napraviti informacijski sustav banke ukoliko ne razumijemo što je uopće banka i bankarstvo. Dakle, potrebno je razumjeti bankarsku domenu.

Razvoj vođen domenom (Domain-Driven Design) (1)

- Razvoj vođen domenom, odnosno DDD, je pristup za razvoj kompleksne programske potpore koji duboko povezuje implementaciju sa razvijajućim modelom jezgre poslovnih koncepata.
- **Glavni temelji DDD-a su:**
  - Glavni fokus projekta postavlja se na jezgru domene i domensku logiku
  - Složeni dizajn temelji se na modelu
  - Uspostavljanje kreativne kolaboracije između eksperata domene i razvojnika radi što bližeg približavanja konceptualnoj srži problema

Razvoj vođen domenom (Domain-Driven Design) (2)

- DDD nije tehnologija niti metodologija.
- DDD pruža strukturu prakse i terminologije za donošenje odluka u dizajnu koje fokusiraju i ubrzavaju projekte programske potpore koji se bave složenim domenama.
- DDD se vodi po načelu “Persistence Ignorance”
  - Model-driven, a ne data-driven
  - Fokus je na domeni, a ne na podatkovnoj strukturi.

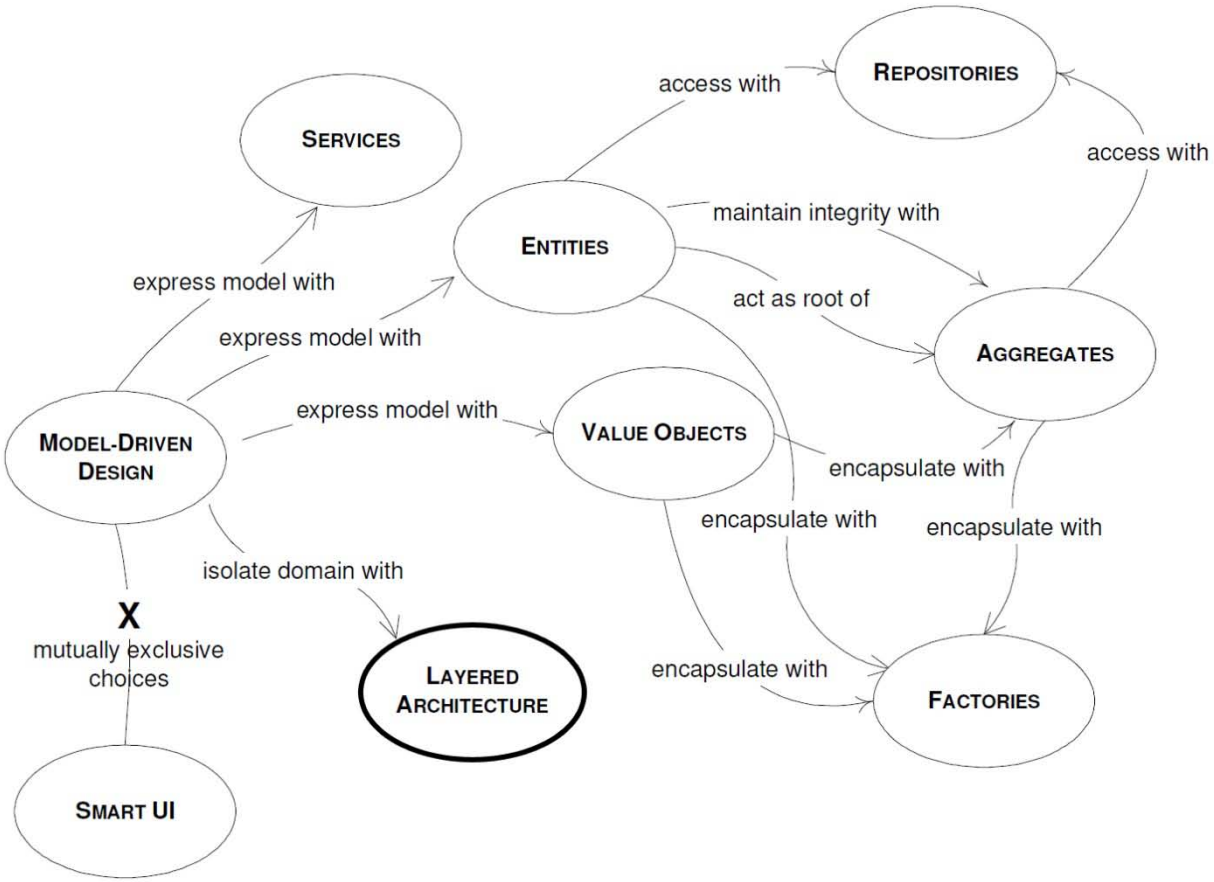
Sveprisutni jezik (1)

- Razvoj modela domene uz suradnju razvojnika i eksperata domene
  - **Problem:** komunikacijska barijera
- Razvojnici misle samo na razrede, metode, algoritme i obrasce
  - Pokušavaju upariti koncepte iz stvarnog svijeta sa programskim konceptima kao što su prog. knjižnice, razvojni okviri, perzistencija i sl.
- Eksperti domene ne znaju ništa o onome što znaju razvojnici
- Eksperti domene znaju samo koncepte iz domene i o tome govore svojim žargonom

Sveprisutni jezik (2)

- Komunikacija je važna za uspjeh projekta.
- Jedan od osnovnih principa DDD-a je korištenje jezika baziranog na modelu.
- Model se mora moći izreći jezikom domene.
- Svi suradnici na projektu koriste taj jezik u svim oblicima komunikacije.
- Iz tog razloga jezik se naziva sveprisutni jezik.

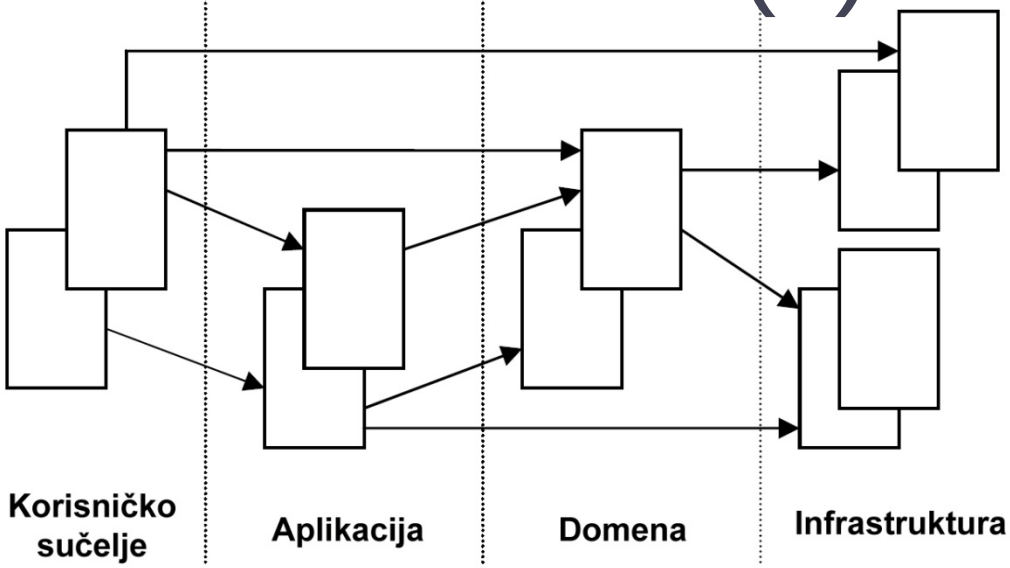
Građevni blokovi DDD-a (1)



Slojevita arhitektura (1)

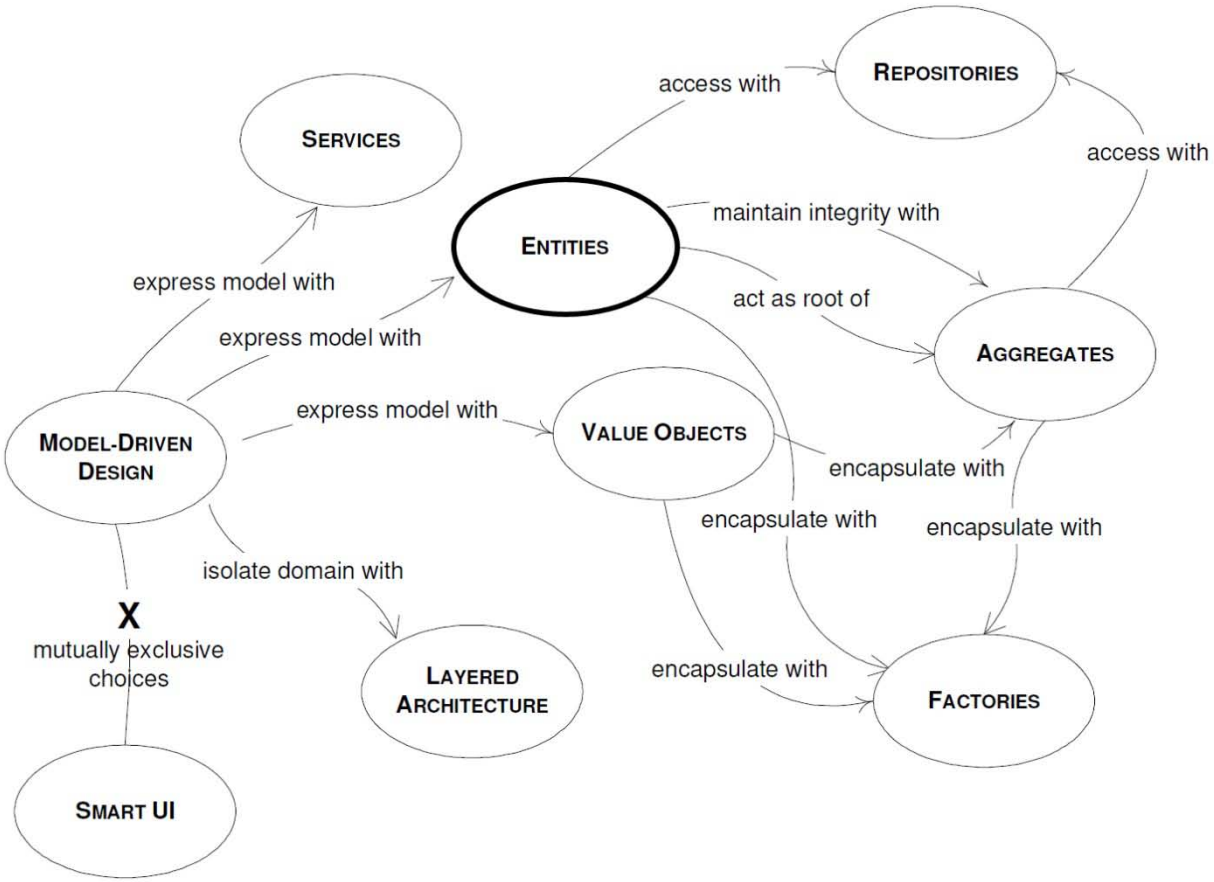
- Često puta se poslovna logika smješta u korisnička sučelja i kod za pristup bazi podataka.
- Kada je domenski orijentiran kod pomiješan sa ostalim slojevima, postaje ekstremno teško razumjeti i razmišljati o domeni koju modeliramo.
- Iz tih se razloga složeni programski sustavi particioniraju u slojeve.
- Slojeve je potrebno razvijati tako da je svaki pojedini sloj kohezivan i da ovisi samo o sloju ispod njega.

Slojevita arhitektura (2)



- **Korisničko sučelje** – prezentacija informacija korisniku
- **Aplikacijski sloj** – tanki sloj koji koordinira aplikacijske aktivnosti. Ne sadrži poslovnu logiku.
- **Domenski sloj** – sadrži informacije o domeni. Jezgra poslovnih informacijskih sustava.
- **Intrastrukturni sloj** – podupiruća knjižnica za sve ostale slojeve

Građevni blokovi DDD-a (2)



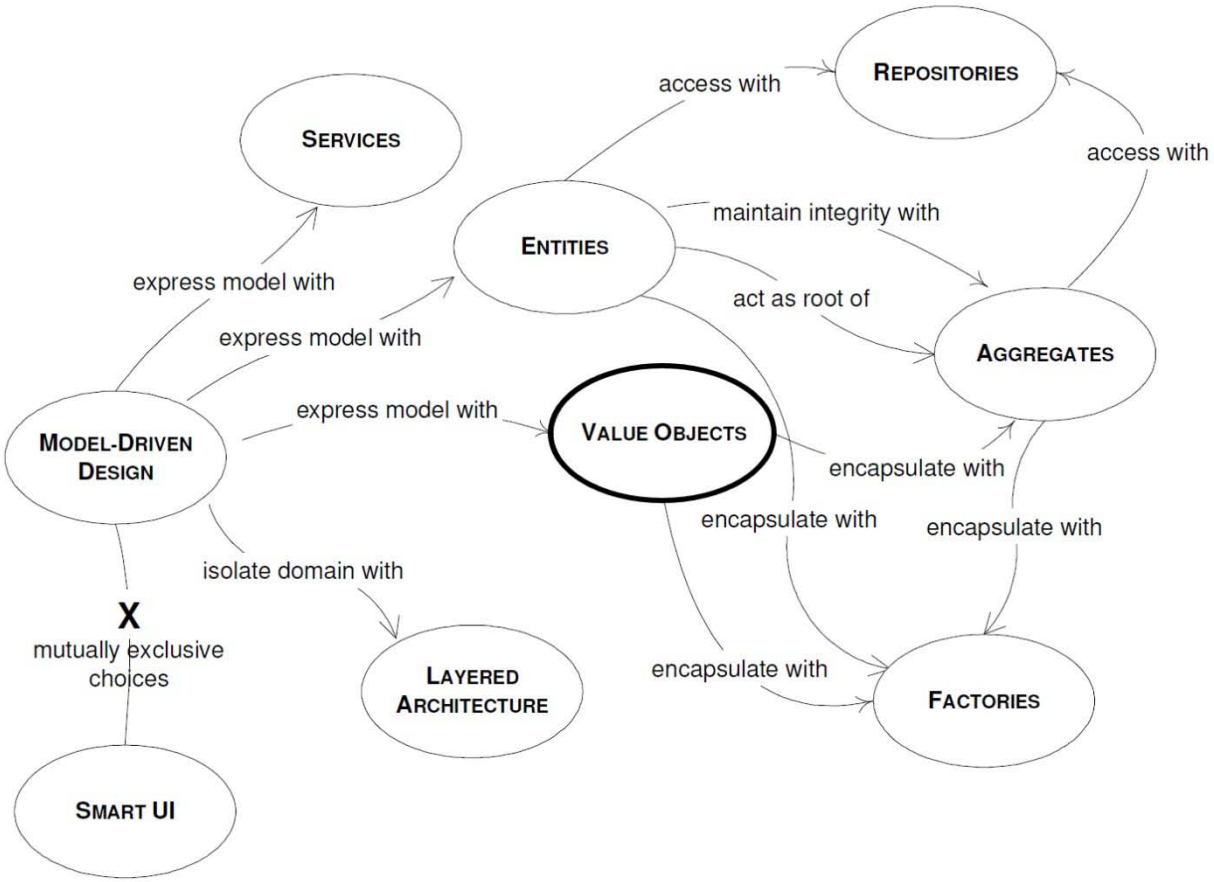
Entiteti(1)

- Objekt koji je primarno određen svojim identitetom naziva se entitet.
- Životni ciklus entiteta može radikalno promijeniti formu i sadržaj objekta, ali identitet i kontinuitet objekta mora biti očuvan.
- Primjer: Osoba tijekom svojeg života mijenja svoj izgled, financijsko stanje, ali njezin identitet ostaje uvijek isti.

Entiteti(2) – Modeliranje entiteta

- Definicija identiteta proizlazi iz modela, odnosno definiranje identiteta zahtjeva poznavanje domene.
- Za osiguranje identiteta potrebno je definirati jedinstveni identifikator i pridodati ga objektu kao atribut.
- Identifikator može biti generiran automatski u sustavu ili može biti definiran vanjskim faktorom (npr. JMBG ili OIB).
- Jednom određen identitet/identifikator se ne nikada ne može promijeniti.

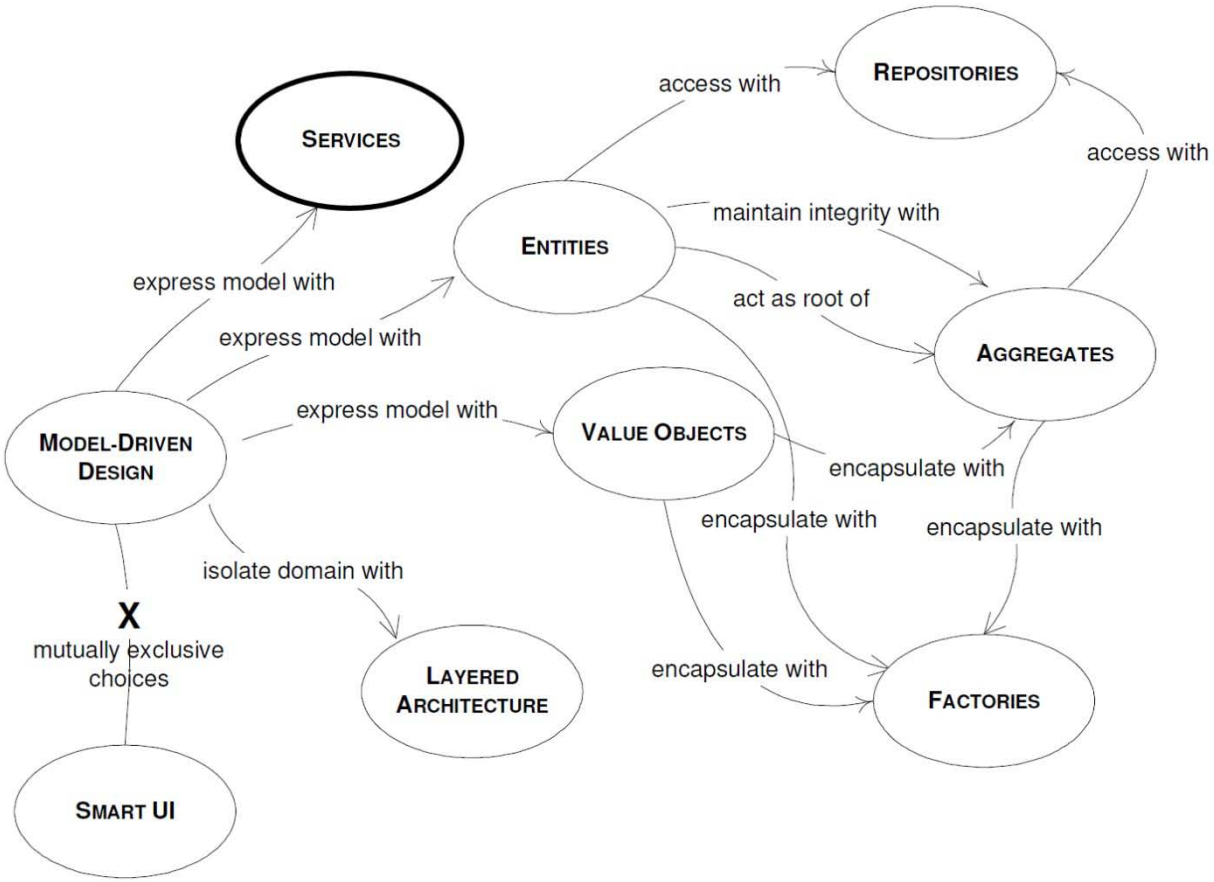
Građevni blokovi DDD-a (3)



Vrijednosni objekti

- Objekt koji predstavlja opisni aspekt domene bez konceptualnog identiteta zove se vrijednosni objekt.
- Mogu se sastojati od drugih objekata.
- Tranzijentne su prirode, odnosno stvaraju se radi neke operacije i nakon toga se mogu odbaciti.
- Vrijednosni objekti moraju biti nepromjenjivi (engl. immutable).

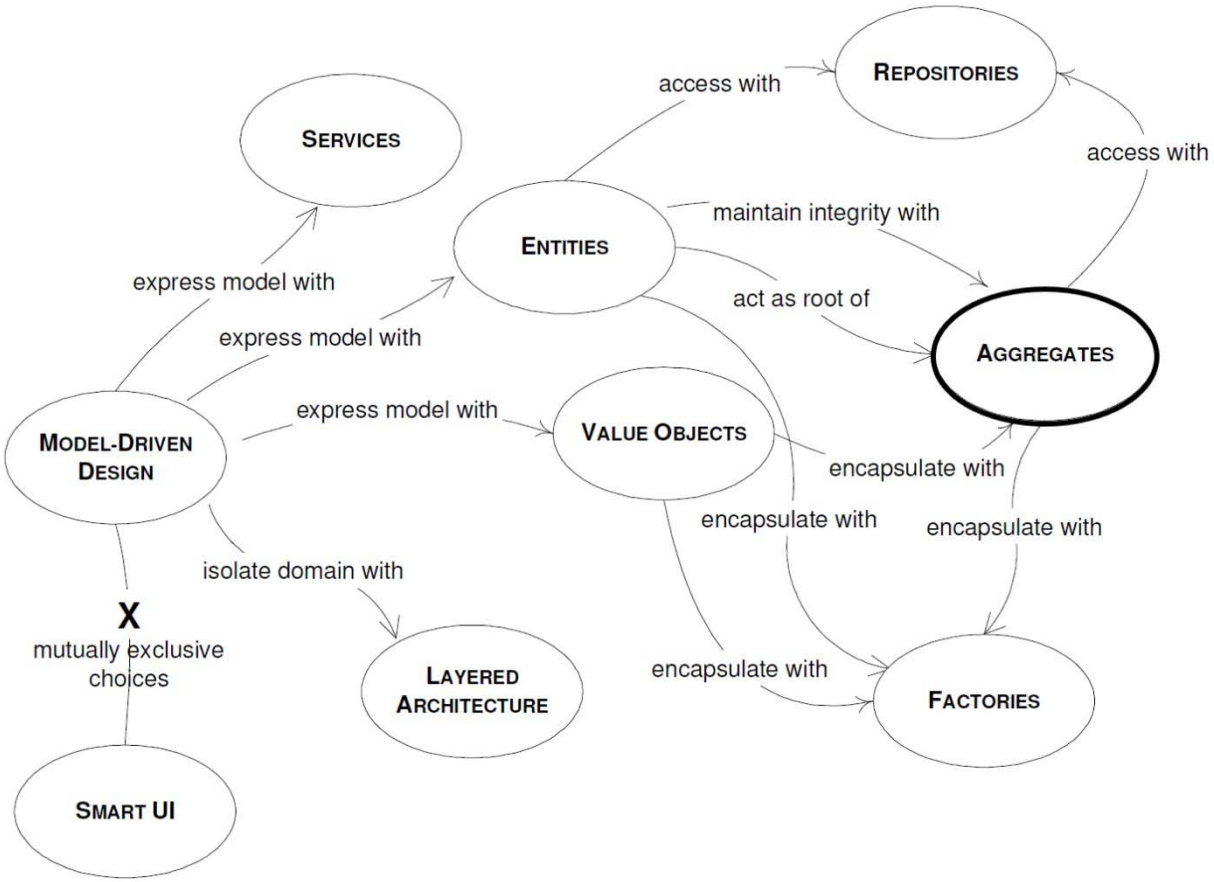
Građevni blokovi DDD-a (4)



Servisi

- Servis je operacija ponuđena preko sučelja koje postoji samostalno u modelu, bez enkapsulacije stanja.
- Dobar servis ima tri karakteristike:
  - Operacija se odnosi na koncept iz domene koji nije prirodni dio entiteta ili vrijednosnog objekta
  - Sučelje je definirano u okvirima drugih elemenata u domeni.
  - Operacije koje servis sadrži moraju biti bez stanja.

Građevni blokovi DDD-a (5)



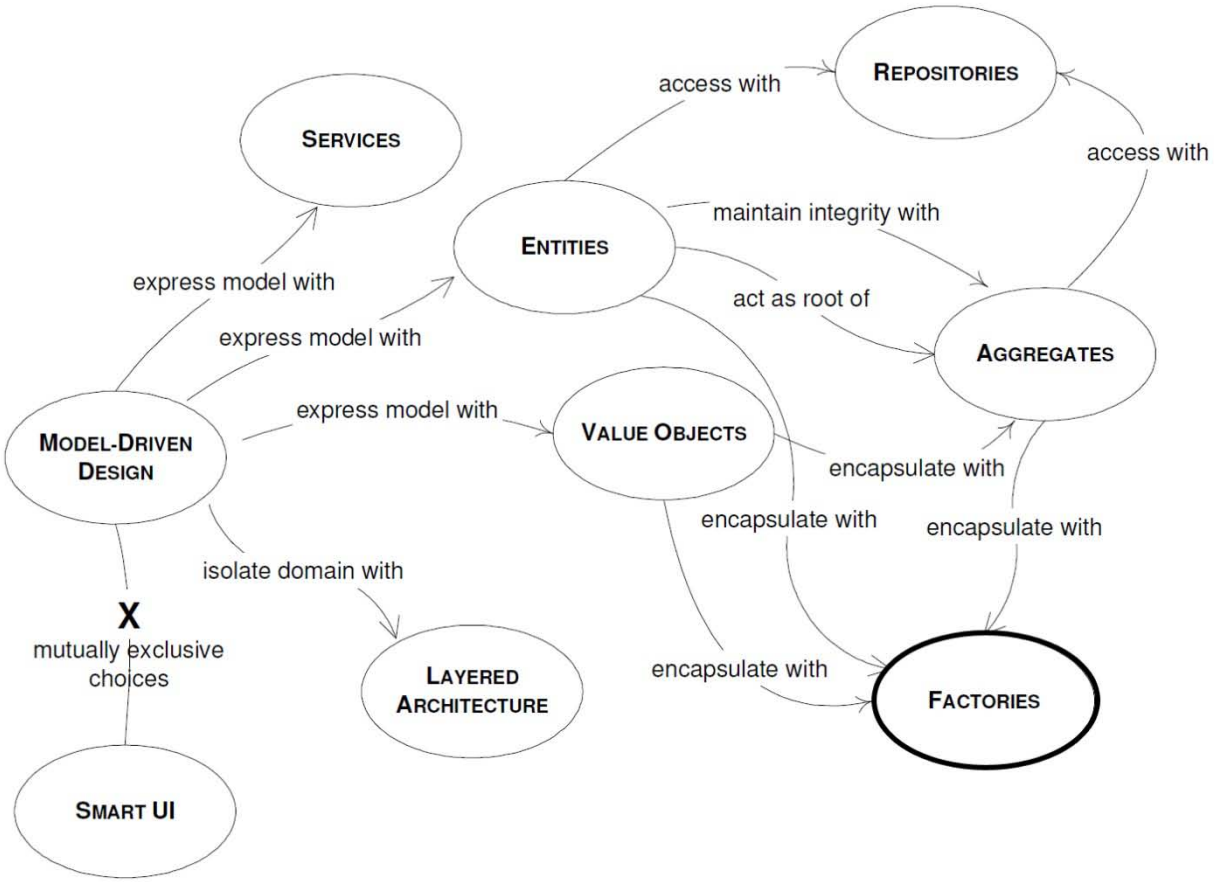
Agregati (1)

- Agregat je skup povezanih objekata koje tretiramo kao jedinku prilikom promjene podataka.
- Korijen je jedan, specifičan entitet sadržan unutar agregata.

Agregati (2)

- Pravila agregata:
  - Korijenski entitet ima globalni identitet i odgovoran je za provjeravanje invarijanti
  - Entiteti unutar granice imaju samo lokalni identitet, jedinstven jedino unutar agregata.
  - Ništa izvan agregata ne može držati referencu na objekt unutar granice. Korijen može predati referencu na unutarnje entiteta na privremeno korištenje.
  - Iz baze je moguće dobiti samo korijenske entitete.
  - Objekti unutar agregata mogu držati referencu na korijene drugih agregata.
  - Prilikom brisanja agregata, operacija brisanja mora odjednom obrisati sve elemente agregata.

Građevni blokovi DDD-a (6)



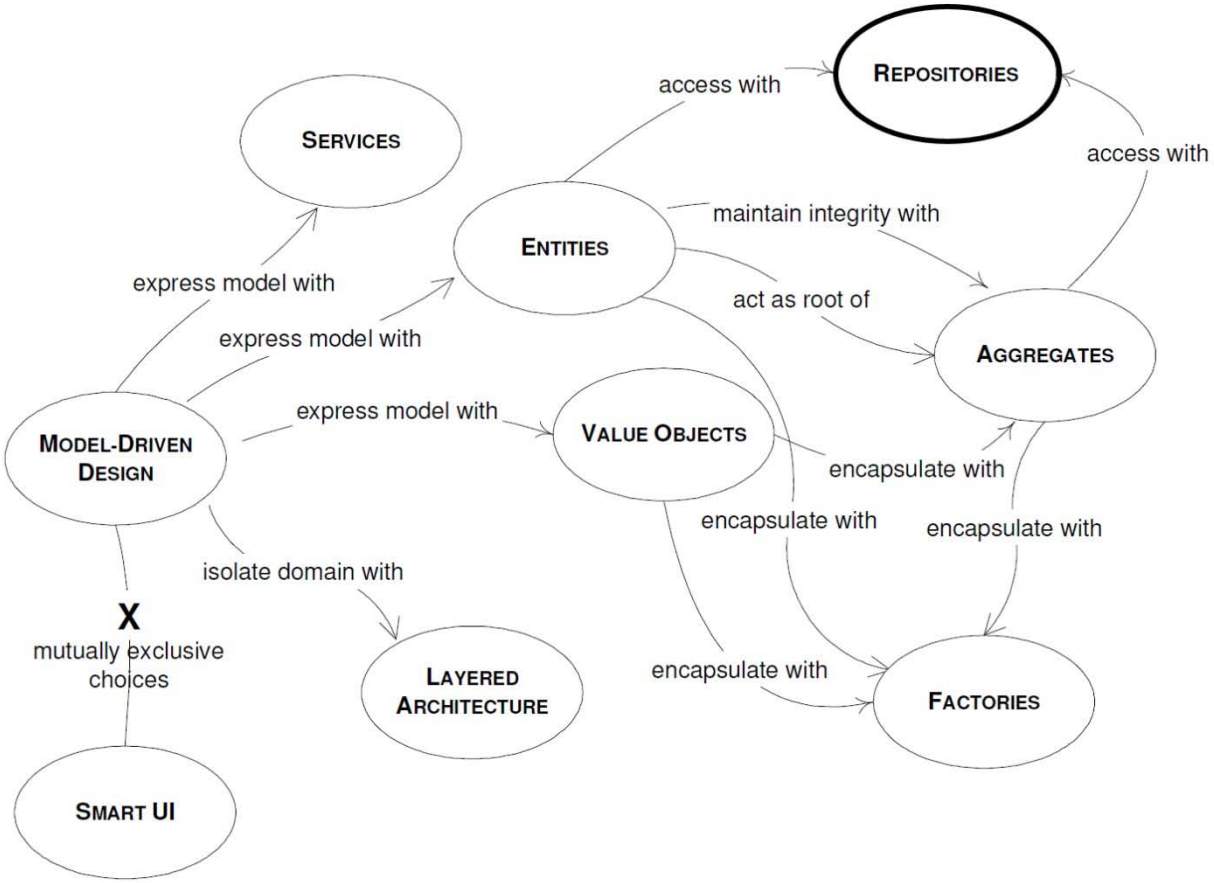
Tvornice (1)

- Stvaranje objekata može biti složena operacija.
- Stvaranje kompleksnih objekata odgovornost je sloja domene, no međutim taj zadatak ne pripada objektima koji izražavaju model.
- Programski element čija je odgovornost stvaranje drugih objekata naziva se tvornica.
- Tvornica enkapsulira znanje potrebno za stvaranje kompleksnih objekata ili agregata.

Tvornice (2)

- Dva osnovna zahtjeva za implementaciju tvornice:
  - Svaka operacija stvaranja je atomarna i provodi sve invarijante kreiranog objekta ili agregata.
  - Tvornica treba biti apstrahirana prema željenom tipu, a ne prema konkretnom razredu.
- Ponekad je dovoljan samo konstruktor.

Građevni blokovi DDD-a (7)



Repozitoriji

- Repozitorij je mehanizam za enkapsulaciju ponašanja spremanja, dohvaćanja i pretraživanja koji emulira kolekciju objekata.
- Repozitorij je potrebno ostvariti kroz dobro poznato globalno sučelje.
- Repozitoriji se definiraju samo za korijenske agregate koji u stvarnosti trebaju direktan pristup.

Kontinuirano refaktoriranje

- Refaktoriranje je proces redizajniranja programskog koda radi poboljšanja modela bez bilo kakvog utjecaja na ponašanje aplikacije.
- Tradicionalno refaktoriranje je tehnički motivirano.
- DDD refaktoriranje je motivirano uvidom u domenu radi odgovarajućeg poboljšanja modela ili njegove reprezentacije u kodu.

Omeđen kontekst

- Veliki projekti obuhvaćaju više modela.
- Kombinacija programskih kodova sa različitim modelima uzrokuje teško održavanje i ne razumijevanje modela.
- Komunikacija postaje ne shvatljiva (sveprisutni jezik).
- Rješenje omeđen kotekst (engl. bounded context) – potrebno je eksplicitno definirati kontekst unutar kojeg se primjenjuje model.

Test-Driven Development (TDD) (1)

- Razvoj vođen testiranjem, ili TDD, je pristup koji pomaže timu u ranoj identifikaciji dizajnerskih problema u projektu kao i verifikaciji da je programski kod u skladu sa modelom domene.
- Vrlo je važno testirati stanja i ponašanje modela domene, a što manje se fokusirati na implementacijske detalje pristupa podacima ili perzistanciju.

Test-Driven Development (TDD) (2)

- Pravila (Red/Green):
  - Napisati programski kod koji ne zadovoljava test
  - Napisati programski kod koji će proći unit test
  - Napraviti preoblikovanje koda (Refaktoriranje)

Upravljanje stambenim zgradama (1)

- Sudionici upravljanja su suvlasnici i upravitelj.
- Vlasnici se brinu i odgovaraju za svoje vlasništvo, a izvršne poslove oko toga povjeravaju upravitelju.
- Upravitelj upravlja zgradom, održava ju, prikuplja pričuvu za zgradu te obavlja i sve druge poslove koje mu povjere suvlasnici.
- Izvor prihoda kojim se osigurava i ostvaruje briga za stambenu zgradu jest pričuva.

Upravljanje stambenim zgradama (2)

- Propisano je da se mora plaćati minimalno 1,53 kn/m^2 mjesečno u zajedničku pričuvu zgrade.
- Potpuno uređen suvlasnički odnos u nekoj nekretnini postoji kada se točno utvrdi tko je vlasnik kojega posebnog dijela zgrade, a što je zajedničko vlasništvo.
- Kako bi se to postiglo potrebno je etažirati zgradu.

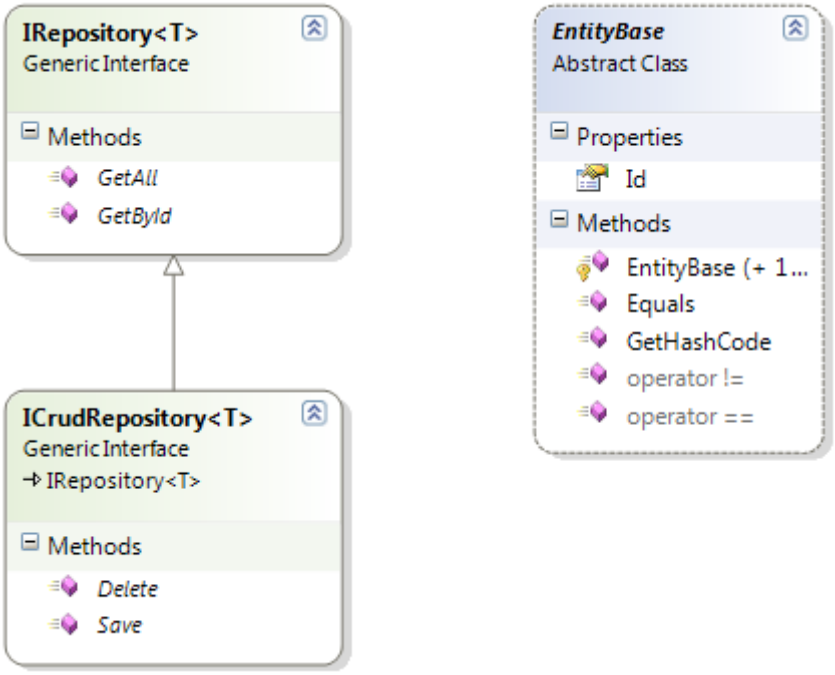
Upravljanje stambenim zgradama (3)

- Svi poslovi koje suvlasnici poduzimaju na zgradi imaju karakter redovne i izvanredne uprave.
- O redovnoj upravi suvlasnici odlučuju većinom glasova.
  - Odluka se smatra donesenom kada se za nju izjasne suvlasnici koji zajedno imaju većinu suvlasničkih dijelova.
- Dok je za izvanrednu upravu potrebna suglasnost svih suvlasnika.

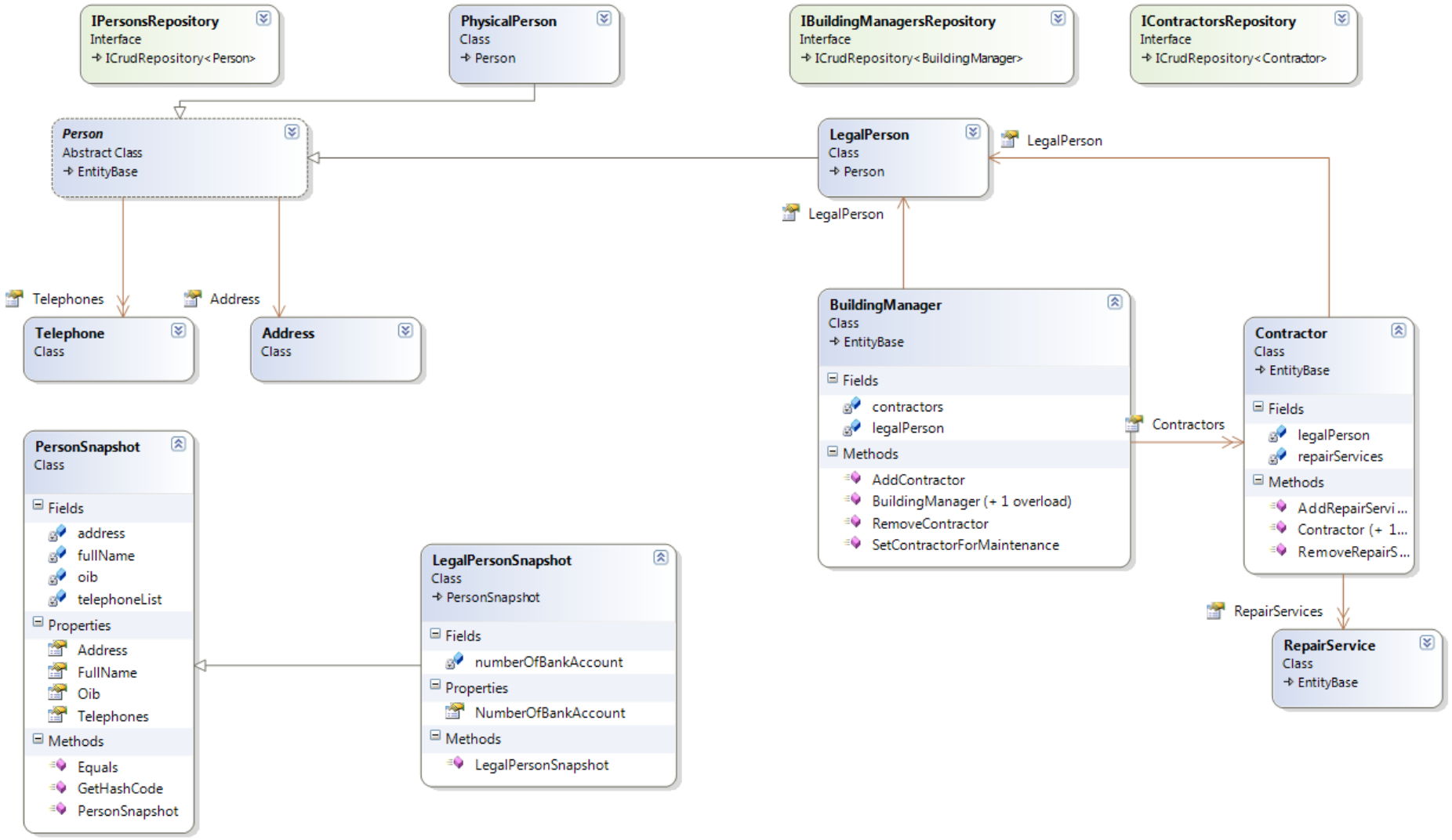
Upravljanje stambenim zgradama (4)

- Suvlasnici mogu prijaviti kvarove upravitelju.
- Upravitelj za prijavljene kvarove angažira svoje kooperante za sanaciju kvarova.
- Predstavnik suvlasnika mora potvrditi da je posao sanacije kvara obavljen.
- Predstavnik suvlasnika mora odobriti svako plaćanje iz pričuve upravitelju.

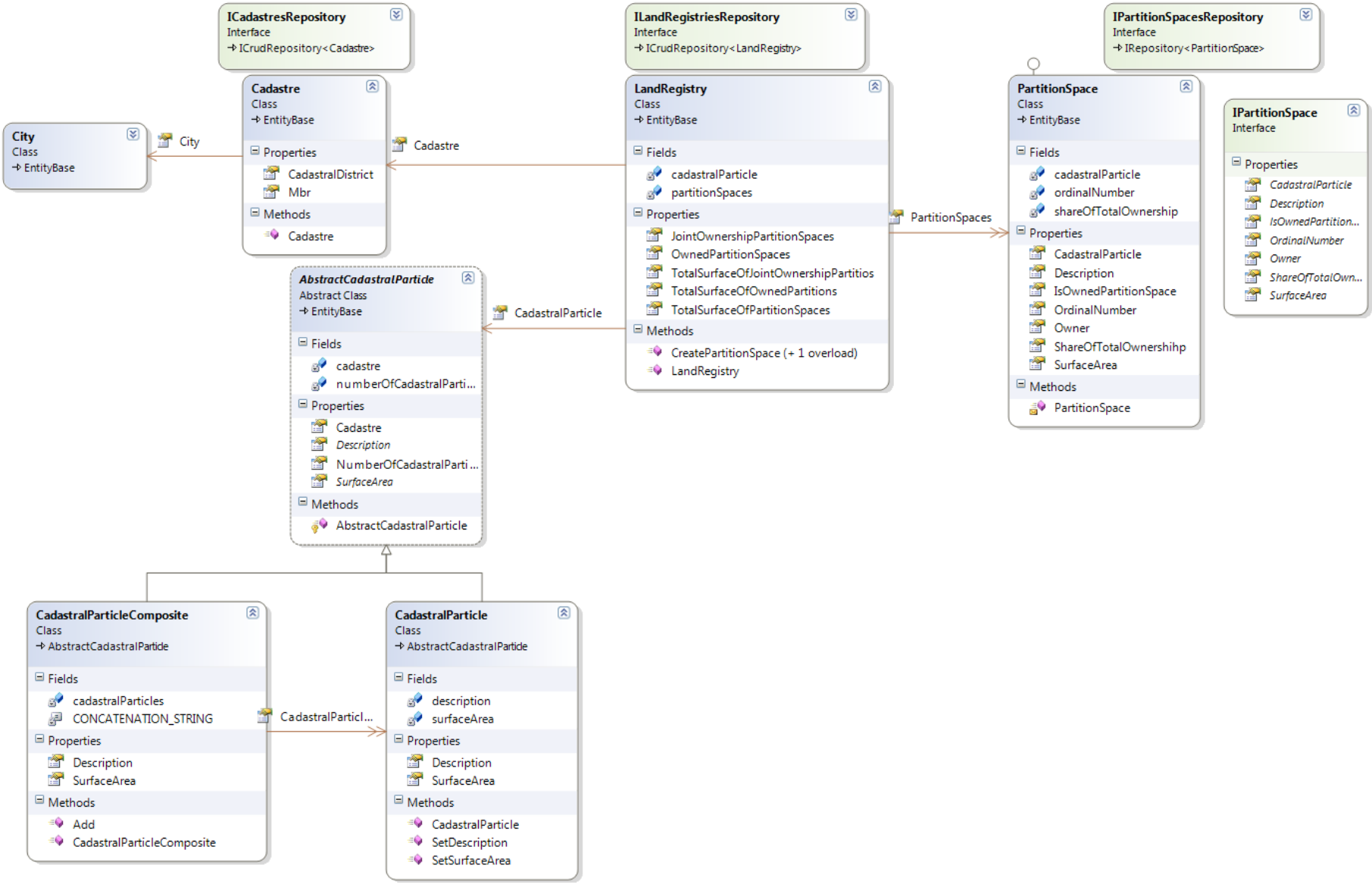
Implementacija - Apstrakcije



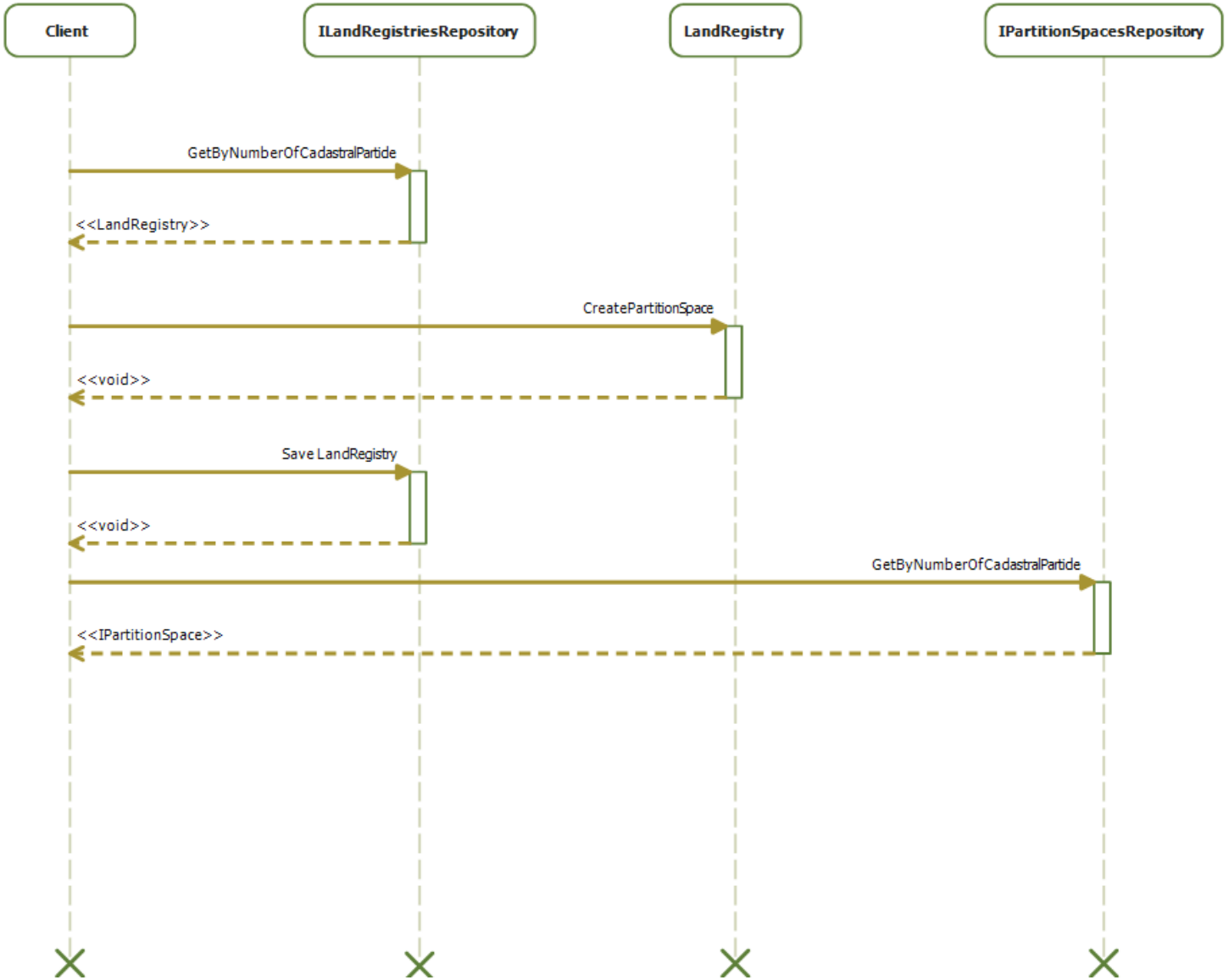
Implementacija – Osobe i uloge



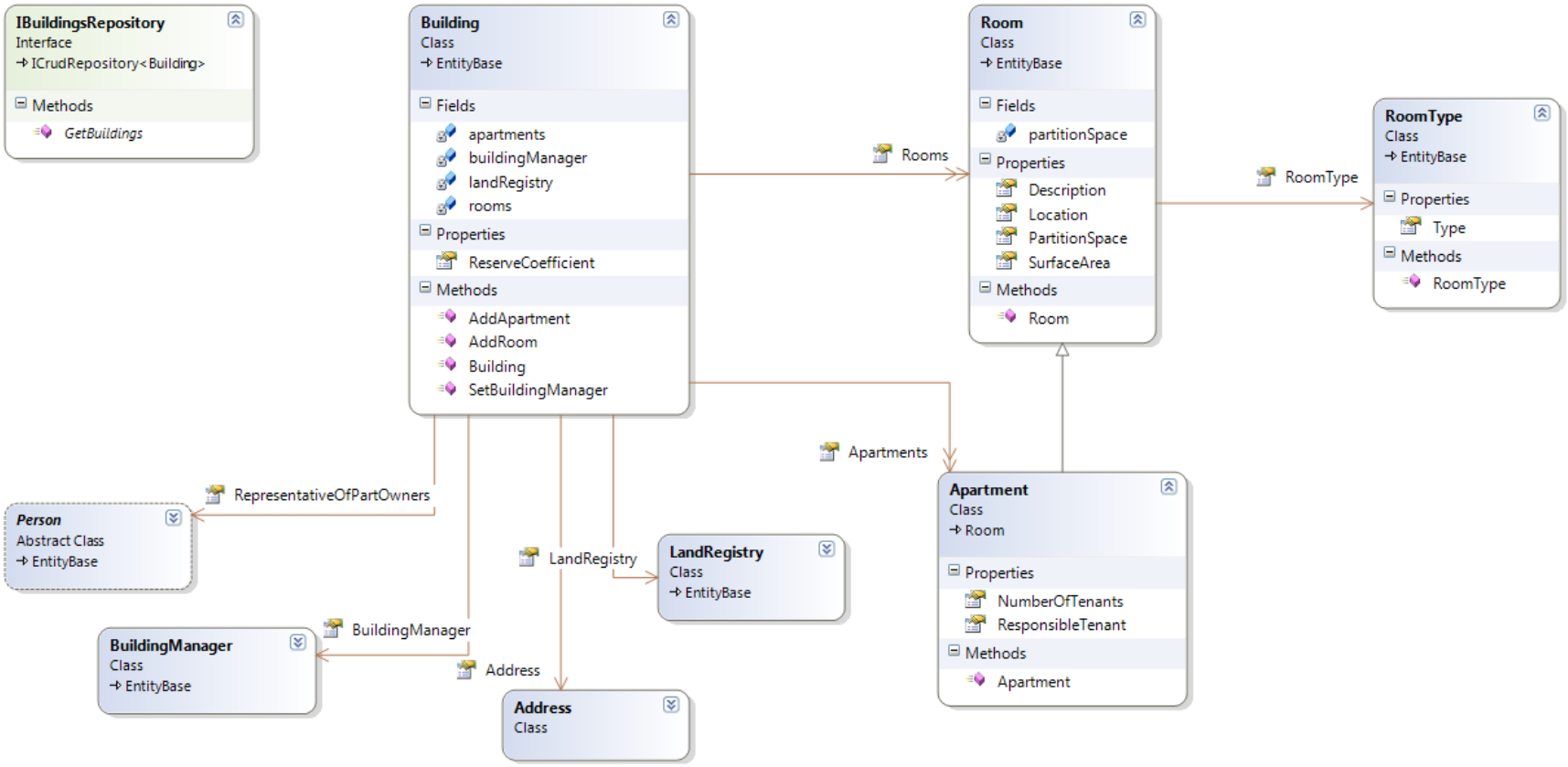
Implementacija - Zakonodavstvo



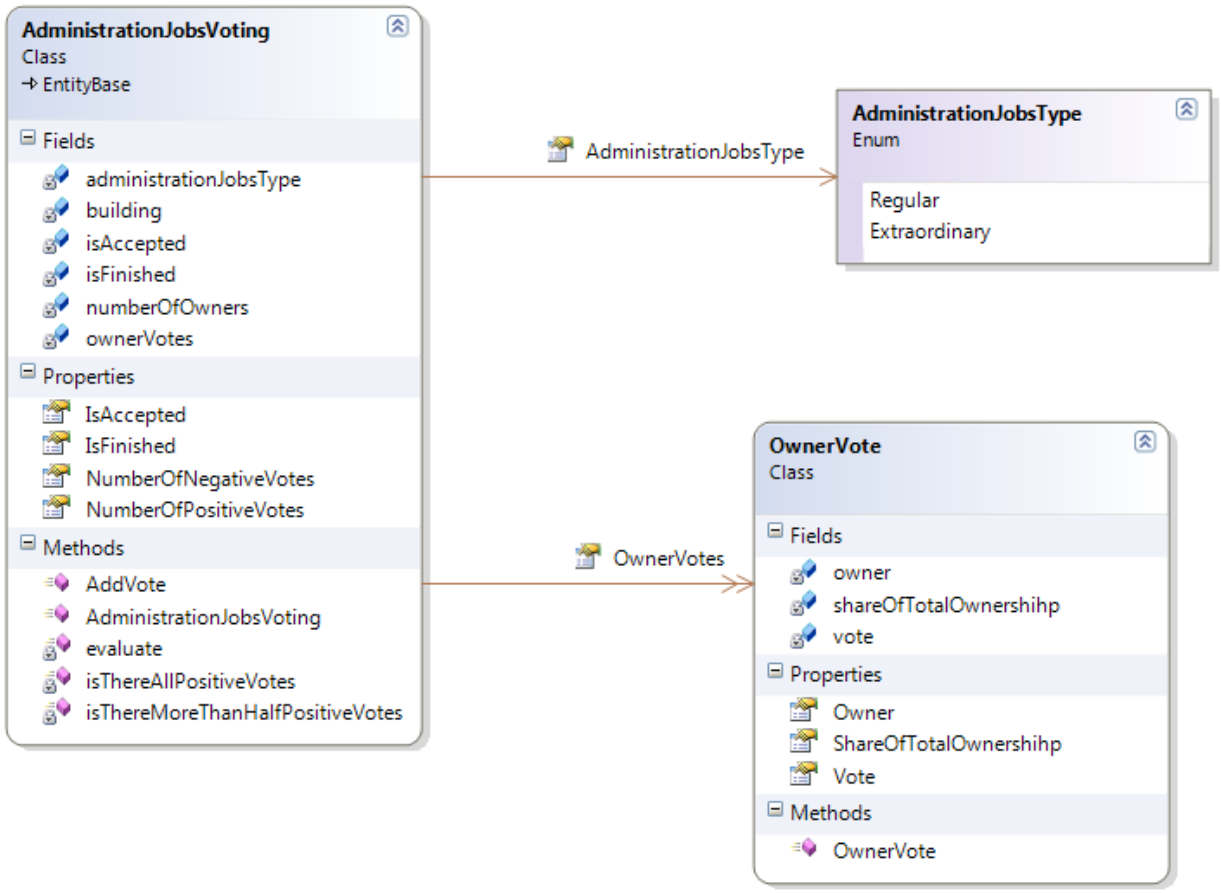
Implementacija - Zakonodavstvo



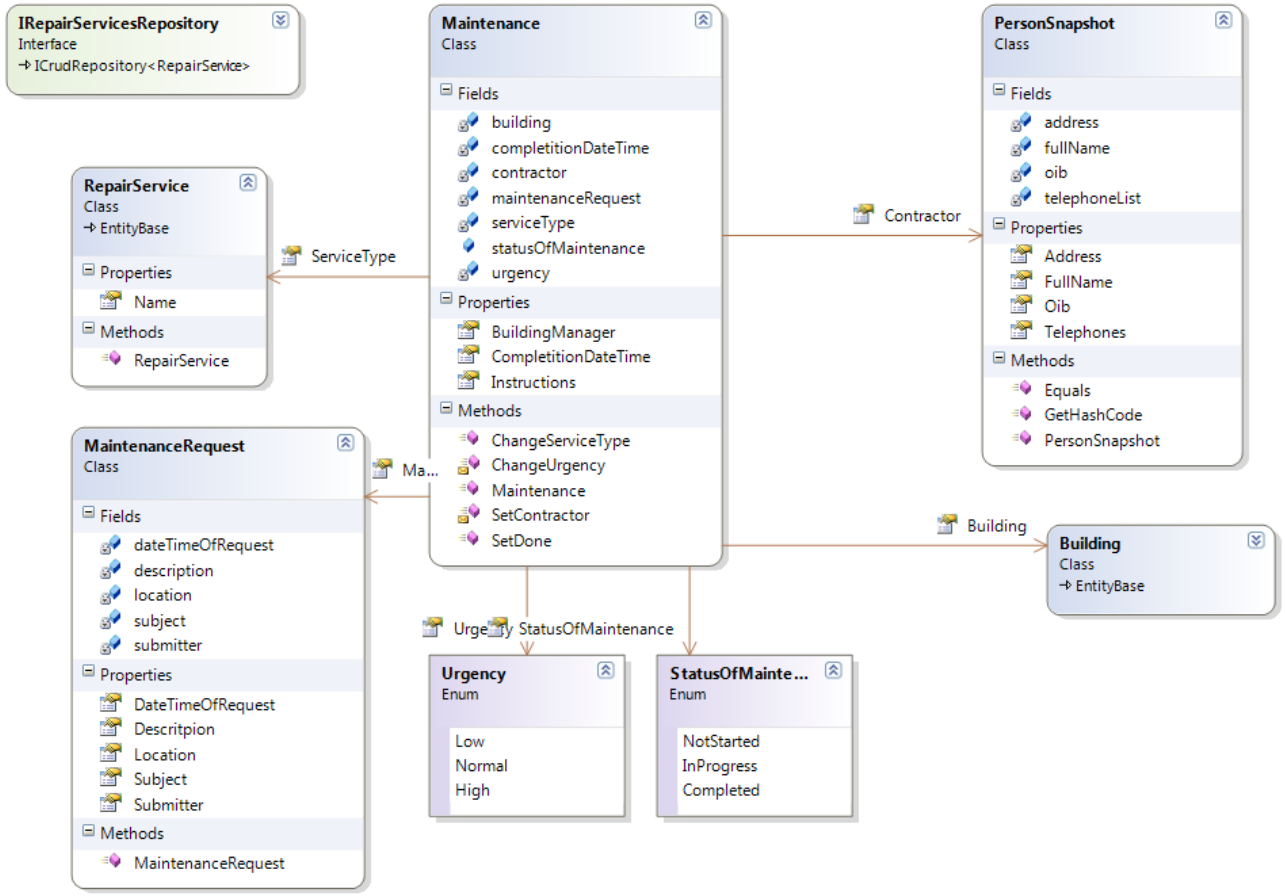
Implementacija – Upravljanje zgradom



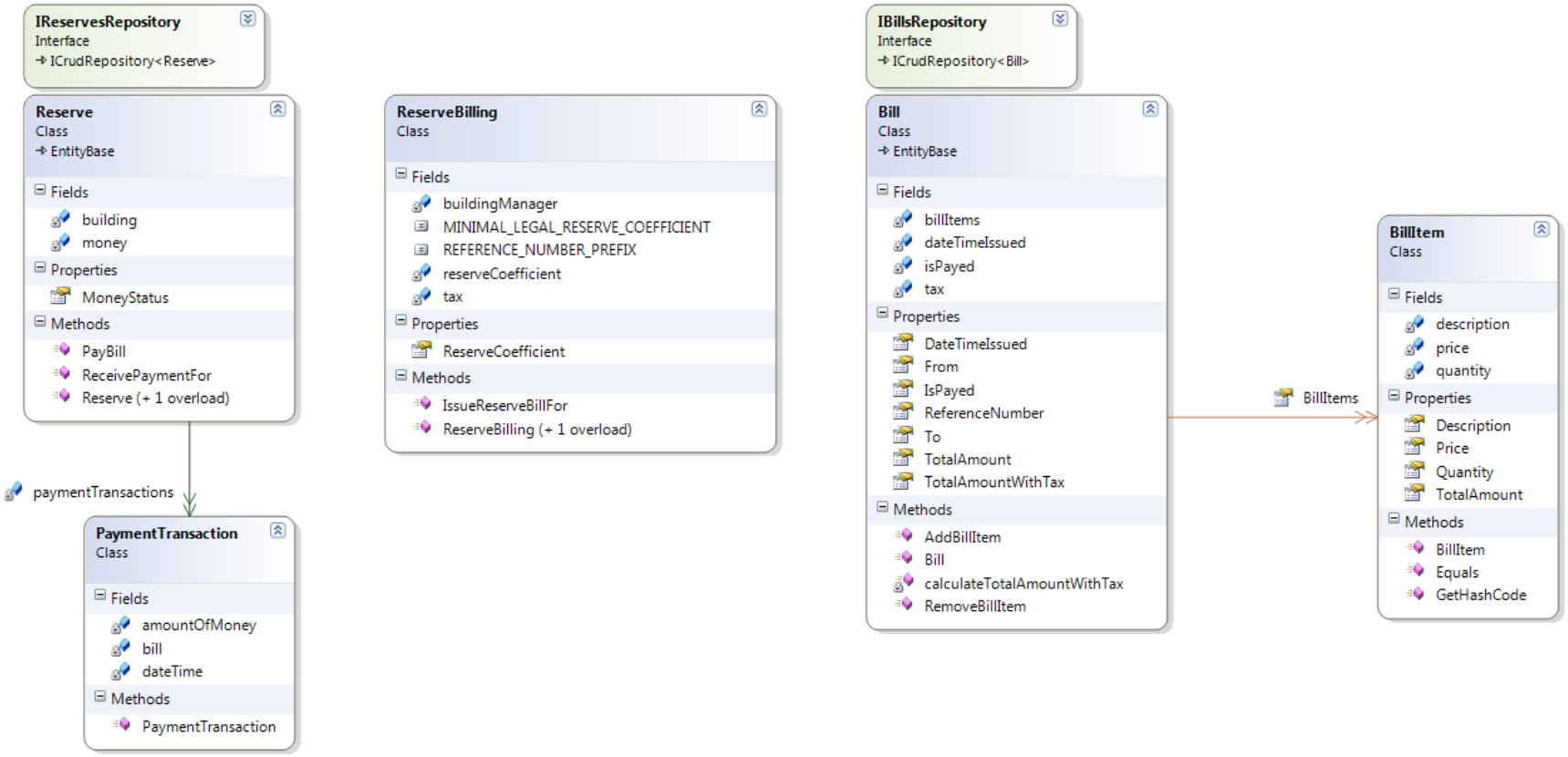
Implementacija – Upravljanje zgradom



Implementacija – Prijava kvara



Implementacija – Financije



Zaključak

- DDD preporučljiv za projekte sa kompleksnom domenom.
  - Popularan i kod manjih projekata.
- Zahtjeva veliku disciplinu prilikom razvijanja od strane razvojnika.
- Na početku vrlo spori rast, kasnije se to nadoknađuje u održavanju i proširivanju.
- Premda je DDD zastupa “Persistence ignorance” potrebno je razmišljati i o implementacijskim detaljima.
  - Implementacijski detalj ponekad određuje što će biti korijenski agregat

Diplomski rad

- Izrada kompletnog informacijskog sustava gdje je jezgra model dizajniran primjenom DDD-a.
- Implementacija infrastrukturnog sloja, odnosno repozitorija, pomoću O/R mapera NHibernate
- Prezantacijski i aplikacijski sloj ostvaren pomoću MVC obrasca, ASP.NET MVC 2
  - DDD model, odnosno sloj domene savršeno se nadopunjuje sa MVC-om

Literatura

- *Eric Evans: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison Wesley, 2003.*
- Srini Penchikala: InfoQ: Domain Driven Design and Development In Practice, 2008.
  URL: http://www.infoq.com/articles/ddd-in-practice
- Abel Avram & Floyd Marinescu: Domain-Driven Design Quickly, 2006.
  URL: http://www.infoq.com/minibooks/domain-driven-design-quickly
- Domain-Driven Design Community
  URL: http://domaindrivendesign.org/
- Domain Driven Design
  URL: http://en.wikipedia.org/wiki/Domain-driven_design