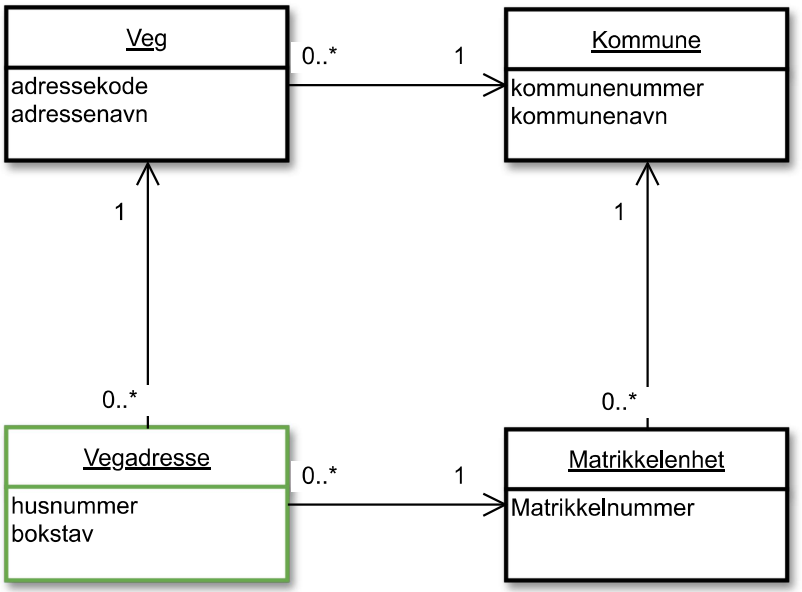
2. Boblemodellen

Hva er bobler?

Matrikkelens domenemodell er bygget opp basert på det logiske domenet modellen representerer. Implementasjonsmodellen er delt inn i selvstendige objekter som refererer til hverandre. Det som gjør denne modellen spesiell er at objektene er løskoblet ved at de refererer til hverandre via koblinger på id-nivå og ikke på objekt-nivå, som i en relasjonsdatabase. Dette fører til at vi kan se på et enkelt objekt alene uten å måtte laste inn andre objekter før vi ønsker å navigere til dem.

Eksempel:

Under følger en forenklet modell for vegadresse og koblingene den har mot andre objekter.



Det er her mange koblinger mellom objekter, men vi ser at alle disse objektene er selvstendige og kan da modelleres som egne bobler. I kode blir dette da følgende

```
class Kommune {
    KommuneId id;
    String kommunenummer;
    String kommunenavn;
}

class Veg {
    VegId id;
    int adressekode;
    String adressenavn;
    KommuneId kommuneId;
}

class Matrikkelnummer {
    KommuneId kommuneId;
    int gardsnummer;
    int bruksnummer;
    int festenummer;
    int seksjonsnummer;
}

class Matrikkelenhet {
    Matrikkelnummer matrikkelnummer;
}

class Vegadresse {
    VegadresseId id;
    int husnummer;
    char bokstav;
    VegId vegId;
    MatrikkelenhetId matrikkelenhetId;
```

```
}
```

Det vi ser her er at alle objekter har sin egen id som er av en type som representerer klassen. Disse id-klassene inneholder igjen et unikt tall som identifiserer objektet. Vi kan så bruke disse id-ene til å referere til de andre objektene i stedet for å vite om objektene i seg selv. En vegadresse vet da om id-en til vegen den er koblet til, men den vet ikke hva adressenavnet er for denne vegen. Det er det kun vegen selv som vet. På samme måte vet vegen hvilken id kommunen den ligger i har, men den vet ikke hva kommunen heter. Det er det kun kommunen som vet.

Bobler og komponenter

I eksemplet over er det en klasse, **Matrikkelnummer**, som ikke finnes som en egen boks i den logiske modellen. Dette er en klasse vi kun lager for å enkelt samle informasjon som henger sammen men som ikke kan sees på for seg som et selvstendig objekt. I dette tilfellet er matrikkelnummeret feltene som logisk identifiserer en matrikkelenhet, men det kan være andre klasser av samme type. Eksempler på dette er en etasje på et bygg, eller et eierforhold for en matrikkelenhet, klasser som har verdi, men som ikke er selvstendige. Disse objektene er en del av boblene de henger på og kan ligge direkte som felter i boblene, eller som del av en liste av komponenter på boblen. Disse objektene følger alltid med boblen de er en del av og kan ikke søkes opp utenom boblene.

Navigering blant bobler

Boblene refererer til hverandre, som vi så i eksempelet over, ved hjelp av id-er. Domenemodellen er ganske tett koblet på den måten at vi ofte vil ønske å sette sammen data fra forskjellige bobler i en visning eller databehandling. Eksempelvis kan vi tenke oss at vi ønsker å vise en adresse for en person. Man er vant til å se adresser med et navn først etterfulgt av et husnummer og bokstav før man til slutt får postnummer og poststed. Denne informasjonen finnes i matrikkelen og i objektene våre, men vi må gå mellom flere objekter og hente data forskjellige steder for å sette sammen disse elementene. For å gjøre dette benytter vi oss av en tjeneste med navn **StoreService**. Denne tjenesten tilbyr metoder for å hente ut et eller flere objekter basert på deres id, og vi kan bruke dette for å hente ut de objektene vi ønsker å navigere til.

Eksempel

```
StoreService storeService; //Oppretter denne slik vi trenger i rammeverket vårt.

//Vi har en vegadresseId fra før. Vi ønsker å hente ut denne og navigere rundt i modellen basert på den.
VegadresseId vegadresseId = new VegadresseId(12345L);
```

```
Vegadresse vegadresse = storeService.getObject(vegadresseId);
Veg veg = storeService.getObject(vegadresse.getVegId());

Kommune kommune = storeService.getObject(veg.getKommuneId());
```

Slik kan vi fortsette for å navigere oss gjennom modellen.

Dette er også godt illustrert i implementasjonsmodellen, se eksempel for adresse her: https://prodtest.matrikkel.no/matrikkelapi/wsapi/v1/dokumentasjon/implementasjonsmodellAPI/index.htm

Det blir fort tydelig at det kan bli mange kall til tjeneren over nettverket hvis man gjør det så enkelt, så vi anbefaler å benytte seg av caching av boblene på klientsiden for lettere bruk. Under følger et enkelt eksempel på en slik cache.

Eksempel 2: Cache

```
class StoreCache{
   Cache<MatrikkelBubbleId, MatrikkelBubbleObject> simpleCache;
   StoreService storeService;

   public MatrikkelBubbleObject get(MatrikkelBubbleId id){
      if(!simpleCace.containsKey(id)){
         MatrikkelBubbleObject o = storeService.getObject(id);
         simpleCache.put(o);
      }
      return simpleCache.get(id);
   }

   public Collection<MatrikkelBubbleObject> get(Collection<MatrikkelBubbleId> ids){
      Set<MatrikkelBubbleId> idsToGet = new HashSet<>();
      for(MatrikkelBubbleId id : ids){
            if(!simpleCache.containsKey(id){
               idsToGet.add(id);
      }
      simpleCache.putAll(storeService.getObjects(idsToGet));
      return simpleCache.getAll(ids);
```

```
    }
}
```

Dette er en veldig enkel implementasjon, og man må nok se på å la objekter forsvinne fra cachen etter en viss tid, eller tømme cachen med jevne mellomrom for å sikre seg at man alltid har de nyeste versjonene av data. Konseptet er uansett at man ikke trenger å hente en boble fra tjeneren hvis man har den fra før! Dette gjør navigasjonen mye raskere, og man kan gå mellom objekter uten anstrengelse.