AARHUS UNIVERSITET

DAB HandIn 1

Gruppe 22

Deltagere	
Studienummer	Navn
201506554	Nina Nguyen
201607649	Stanie Cheung

Indholdsfortegnelse

1 Introduktion

Denne journal består kun af udviklingsgrundlaget til de senere HandIns. Der vil derfor ikke være programmeret.

Opgaven går ud på at bygge et Personkartotek, hvor der sættes fokus på at lave et **DDD**(Domain Driven Design) og et **ERD**(Entity Relationship Design). Der vil være beskrevet om de antagelser der blev foretaget for diagrammerne, samt en forklaring på associationerne der blev anvendt.

Et Personkartotek håndterer en række basale oplysninger for en række personer, som har forbindelse til kartotekets ejer. I mange forskellige sammenhænge, har man brug for at gemme og vedligeholde kontaktoplysninger for en række personer/kontakter. Kontaktoplysningerne i Personkartoteket bruges til at kontakte personer på forskellige måder, samt holde styr på forbindelsen mellem dem.

Oplysninger som data:
Personkartoteket indeholder og sammenholder en række data for hver person:

- En persons fornavn, mellemnavn og efternavn, samt typen af personen, hvor typen er afhængig af kontekst for Personkartoteket.
- Hvis muligt en eller flere emailadresser til personen.
- Et eller flere telefonnumre til personen, samt oplysninger på telefonen.
- adresseoplysninger: vejnavn, husnummer, postnummer og bynavn på personens primære kontaktadresse.
- Hvis et alternativt adresse tilføjes, skal typen af adressen anerkendes.
- Oplysninger om postnummer og by for en adresse kommer fra en liste af muligheder.
- Mangler et postnummer og by, f.eks. "DK 8850 BjerringBro"tilføjes disse oplysninger til listen og tilknyttes den relevante adresse, og ellers vælges der et eksisterende sæt af postnummer og by fra listen, og byen tilknyttes den relevante adresse.
- Noter som er tilføjet i forbindelse til en kontakt kan ændres og slettes efter behov.

2 Domain Driven Design

Domain Driven Design er en tilgang til softwareudvikling til komplekse behov ved at forbinde implementeringen til en udviklende model.

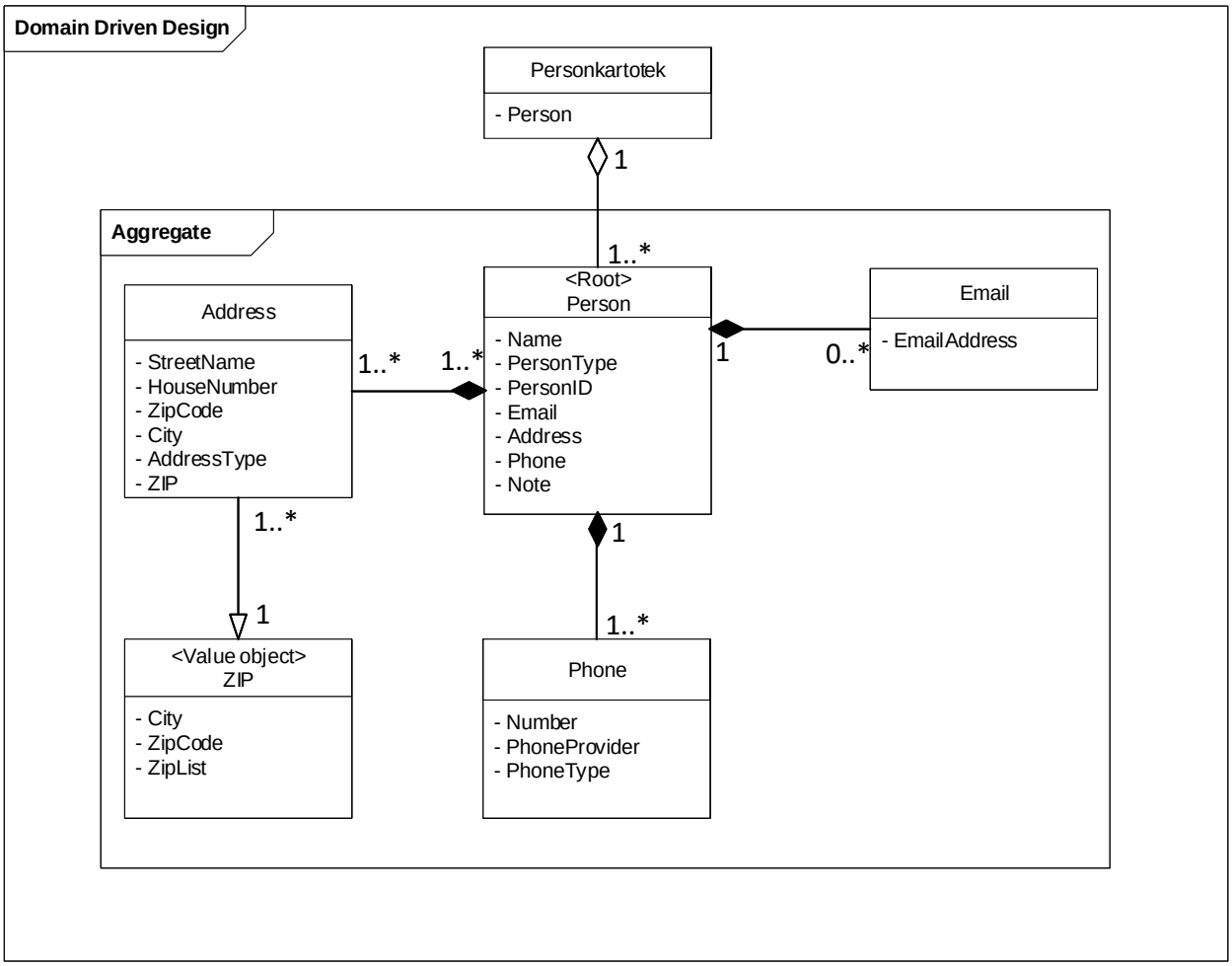
Ud fra data oplysningerne, er der lavet et **DDD**, som udbygger den grundlæggende karakteristik for et personkartotek. Der er blevet anvendt af *entity*, *value object* og *aggregate* i designet.

Aggregate
På figur 3 ses en ramme, *Aggregate*, der adskiller objekter mellem det der er indenfor aggregaten med det der er udenfor aggregaten. Indenfor rammen ses Person, Address, Email, ZIP og Phone. Disse objekter betragtes som én enhed med hensyn til *data ændringer*. I dette tilfælde baserer ændringerne sig om *Personkartotek*. I et aggregate, er det et krav at den har én root. Og en root er en entitet, der kan kommunikere med objekter der er udenfor aggregaten. Med eksempel på figur 3, ses det at Person er markeret root, dvs. at Person er den eneste entitet, der kan kommunikere med Personkartotek.

Entity
Person er med udgangspunkt valgt som en entity, da det er det der oprettes, når en person skal tilføjes til personkartoteket. Det vidnes også at den ses som en root(En root er en entity). Person er en attribute til objektet Personkartotek, som er en karakteristik af *identity*.

Root
Person er valgt som root, da det er det primære egenskab personkartoteket tilføjer i databasen. Dette er også valgt eftersom de andre objekter baserer sig på en persons oplysninger. Derfor er Person valgt som root i dette design.

Value object
ZIP er valgt som et value object, da det er et objekt der indeholder en liste af ZIPs for adresser der tilføjes til personen. Objektet genbruges i flere omgange, hvor det ikke bør være en konsekvens at kopiere sig af den.



Figur 1: Figur af Domain Driven Design

3 Json og XML

I denne sektion ses to modeller: JSON og XML, som er lavet med udgangspunkt fra DDD'et. De to modeller viser et eksempel på hvordan oprettelsen af personer nogenlunde vil se ud, når der skal udvekles data til serveren.

3.1 Json

```
{
  "Name": {
    "FirstName": "Willard",
    "MiddleName": "Carroll",
    "LastName": "Smith"
  },
  "PersonType": "Skuespiller",
  "PersonID": "1",
  "Note": "God skuespiller - virkelig kendt",
  "Email": [
    {
        "EmailAddress": "will.smith@gmail.com"
    }
  ],
  "Address": [
    {
        "StreetName": "Januarvej",
        "HouseNumber": "13",
        "ZipCode": "90210",
        "City": "Los Angeles",
        "AddressType": "Private",
        "Zip": {
            "City": "",
            "ZipCode": "",
            "ZipList": ""
        }
    }
  ],
  "Phone": [
    {
        "Number": "954-565-236",
        "PhoneProvider": "Verizon",
        "PhoneType": "Private"
    },
    {
```

```
        "Number": "958-258-139",
        "PhoneProvider": "Verizon",
        "PhoneType": "Public"
    }
  ]
}
```

3.2 XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<Person>
  <Name>
    <FirstName>Willard</FirstName>
    <MiddleName>Carroll</MiddleName>
    <LastName>Smith</LastName>
  </Name>
  <PersonType>Actor</PersonType>
  <PersonID>1</PersonID>
  <Note>God skuespiller - virkelig kendt</Note>
  <Email>
    <EmailAddress>will.smith@gmail.com</EmailAddress>
  </Email>
  <Address>
    <StreetName>Abbey Road</StreetName>
    <HouseNumber>13</HouseNumber>
    <ZipCode>90210</ZipCode>
    <City>Los Angeles</City>
    <AddressType>Private</AddressType>
    <Zip>
      <City></City>
      <ZipCode></ZipCode>
      <ZipList></ZipList>
    </Zip>
  </Address>
  <Phone>
    <Number>954-565-236</Number>
    <PhoneProvider>Verizon</PhoneProvider>
    <PhoneType>Private</PhoneType>
  </Phone>
</Person>
```

4 Entity Relationship Diagram

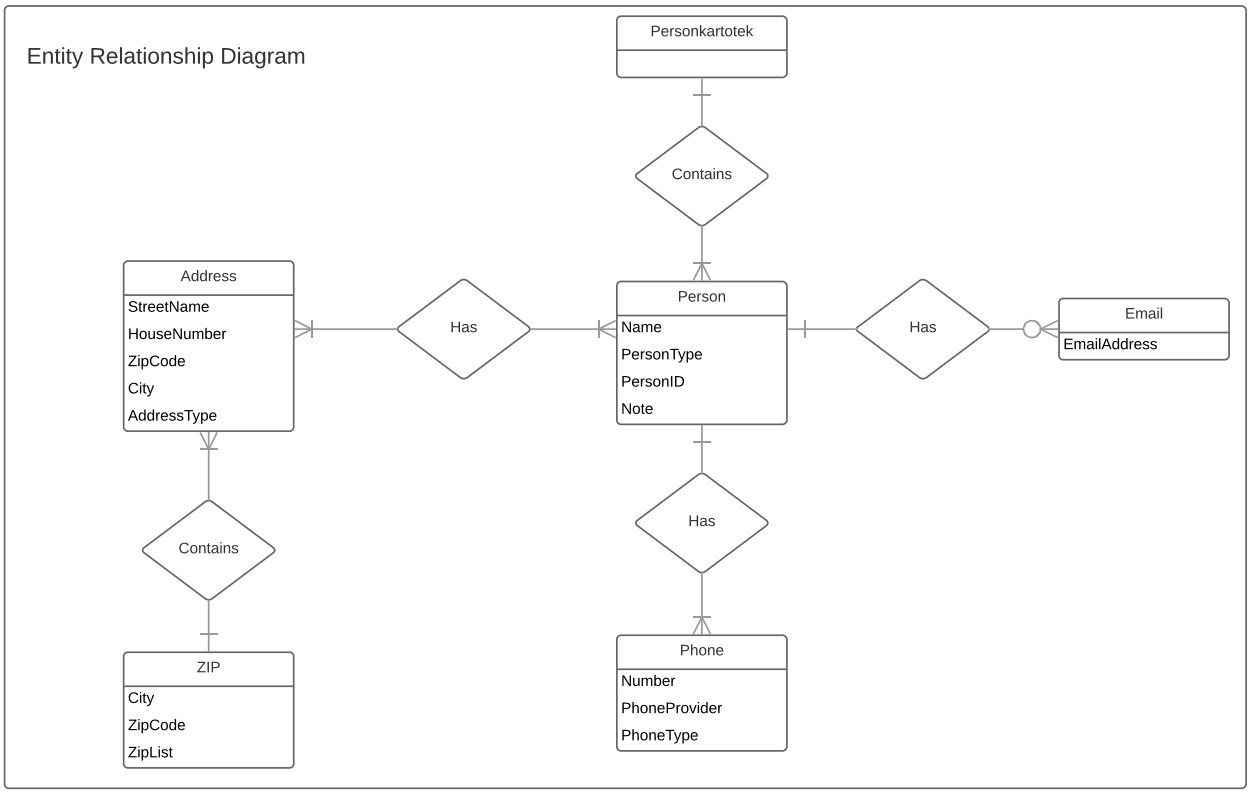
På figur 2 nedenfor er der udarbejdet et ERD ud fra DDD'et. ERD bruges til at beskrive entities, attributter og de forskellige forhold der findes i systemet. Den basale ER-model består af 3 klasser: entities, relationship og attributes.

Entities
Et entitet repræsenterer et objekt fra DDD'et. På figur 2 ses 5 entities; Personkartotek, Person, Address, ZIP, Phone og Email. I et ERD er et entitet et objekt, der indeholder data. Disse entiteter er opbygget som rektangler med et initial.

Relationships
På figur 2 ses det flere steder et diamantformet-blok tilknyttet til to entiteter. Disse blokke kaldes for relationships. Et relationship har til formål at beskrive hvilken type af forhold entiteterne har mellem hinanden. Der findes tre forskellige typer for et relationship, som er følgende: *Degree*, *Connectivity* og *Existence*. Degree beskriver antallet af entiteter tilknyttet til en relationship, connectivity beskriver kardinaliteten mellem entiteterne, dvs. *one-to-one*, *one-to-many* og *many-to-many* aspekter. Existence beskriver om forekomsten for entiterne er *mandatory* og *optional*. Med et eksempel fra figur 2, ses det at entiteten Person *has* en Address. I dette tilfælde er det en degree type af *binary*, dvs. et relationship mellem to entiteter. Det ses også at dens connectivity type er en many-to-many, og existence typen er en mandatory.

Hele ERD'et består primært af **binary** degree typer.

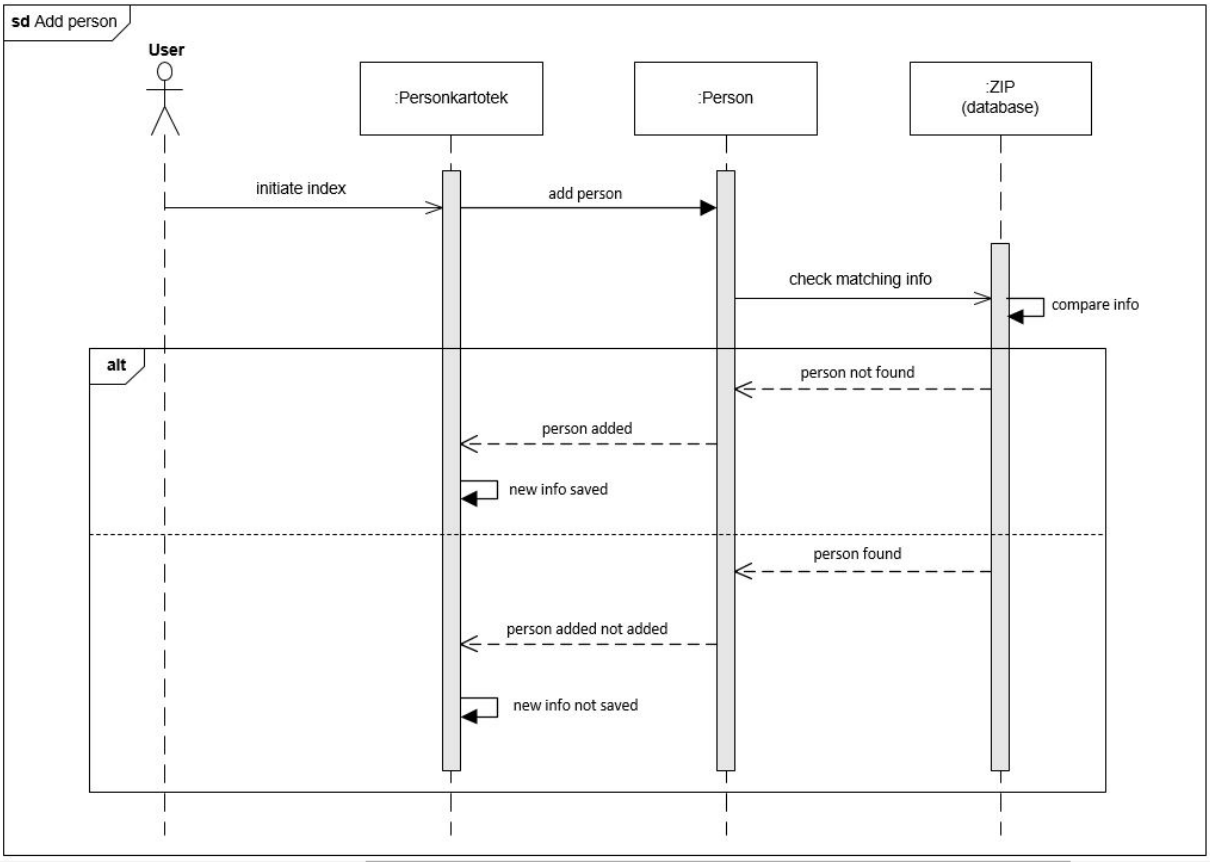
Attributes
Attributes er objeketer der beskriver om entitetens fundament. Der findes overordnet to slags attributes, en identifier (key), og en descriptor. Identifier anvendes til at angive om entiteten har et primary-key eller foreign-key. En descriptor specificere en unik karakteristik af et entitet, som kan være et navn el. lign.



Figur 2: Figur af Entity Relationship Diagram

5 Applikations software arkitektur

I dette afsnit er der lavet et sekvensdiagram over tilføjelse af en ny person til personkartoteket. Sekvensdiagrammet giver et scenarie af, hvordan en person bliver tilføjet hvis denne ikke allerede eksisterer i databasen. Hvis personen derimod allerede eksisterer i databasen, viser sekvensdiagrammet det alternative scenarie, hvor den allerede eksisterende person så ikke vil blive tilføjet til databasen igen, da dette ellers vil overlappe det allerede eksisterende data i databasen.
Det skal lige tilføjes at der ikke er tilskrevet rigtige funktionsnavne til funktionerne endnu, men at disse kan tilføjes senere når der skal programmes, og at sekvensdiagrammet herefter kan rettes til.



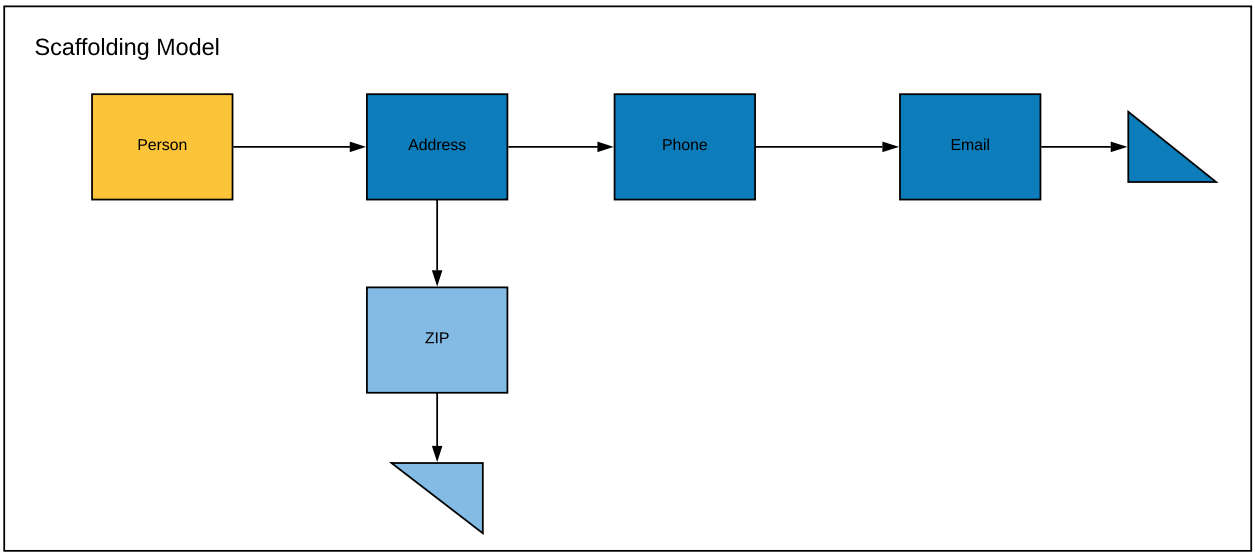
Figur 3: Figur af sekvensdiagram

6 Scaffolding og Graph

Scaffolding- og Graphmodellen er lavet ud fra DDD'et, hvor der er taget udgangspunkt i aggregate. Her indgår Person, Address, Phone, Email og ZIP med en farve-identifikation. Person har farven gul, som indikere at den er root-objektet. Address, Phone og Email er mørkeblå, som indikere at de er objekter der tilhører root-objektet. I de to eksempler ses det at de alle tilhører root-objektet. ZIP har farven lyseblå, som indikere at det er et objekt der tilhører et mørkeblåt objekt. Det betyder at den er et under-objekt, tilhørende til et objekt, som tilhører root-objektet.

6.1 Scaffolding

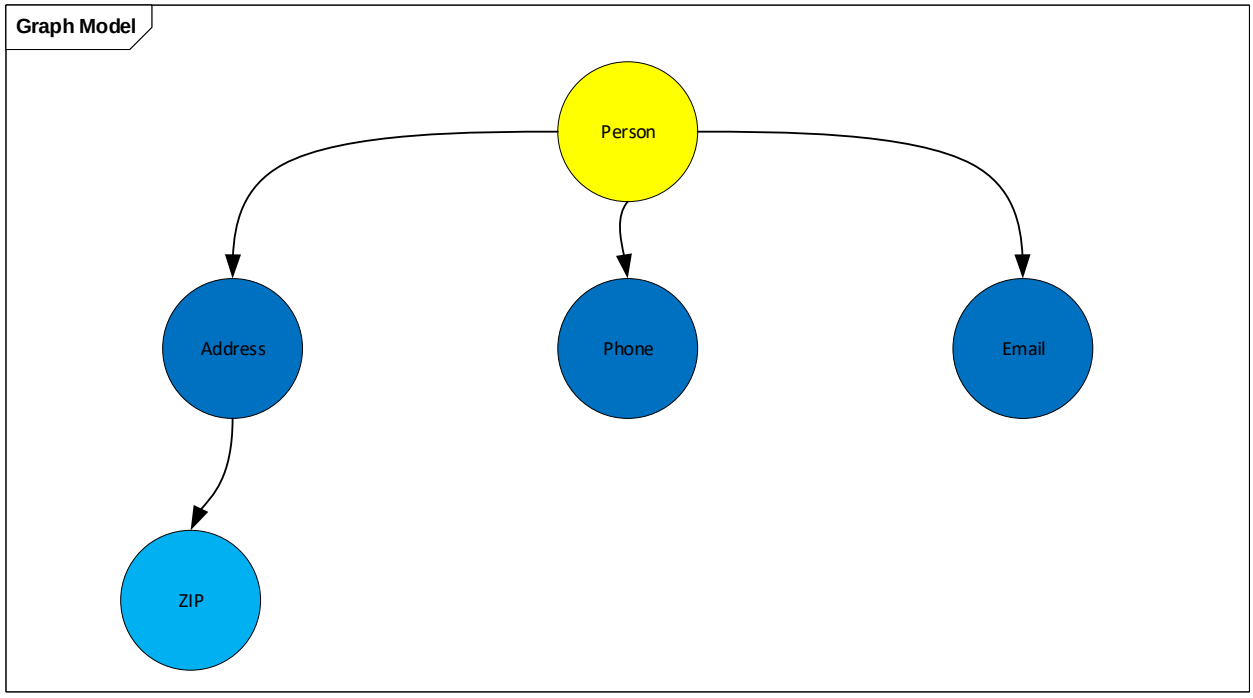
Figur 4 nedenfor er scaffolding modellen til Personkartotek. Scaffolding anvendes til at understøtte **MVC** (Model-view-controller) rammen, hvor programmøren kan angive hvordan applikationen i forhold til databasen kan anvendes. På den måde kan der anvendes af **CRUD** konceptet: CREATE, READ, UPDATE, DELETE. Som grundlæggende gør det til et scaffolding model. Det vil løbende forbedre applikationen.



Figur 4: Figur af Scaffolding modellen

6.2 Graph

Figur 5 nedenfor er graph modellen til Personkartotek. På samme måde, som scaffolding, er modellen bygget efter database behov. Modellen kan sammenlignes med datastukturer. Den fortæller hvordan man søger i strukturen på tværs af teknikker, dvs. kode og database. Med et eksempel, starter man altid fra root, når man søger i en datastruktur. Med andre ord kræves det, at man altid starter fra root-objektet, før der kan interageres med de andre objekter.



Figur 5: Figur af Graph modellen