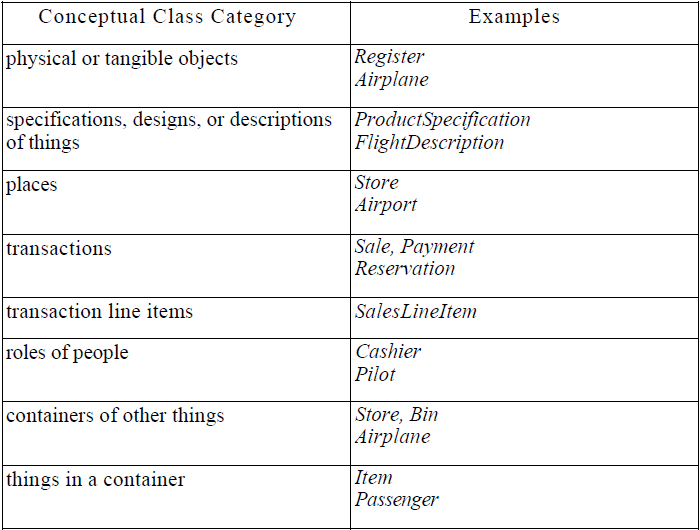
Domæne model (Simon)

En domæne model bliver brugt som en kilde af inspiration for design af diverse software objekter.

En domæne model illustrerer konceptuelle klasser i et problem domæne, og beskriver ikke software komponenter som for eksempel java-klasser. Domæne modellen er en af de mest vigtige artefakter at lave i ens objekt-orienterede analyse.

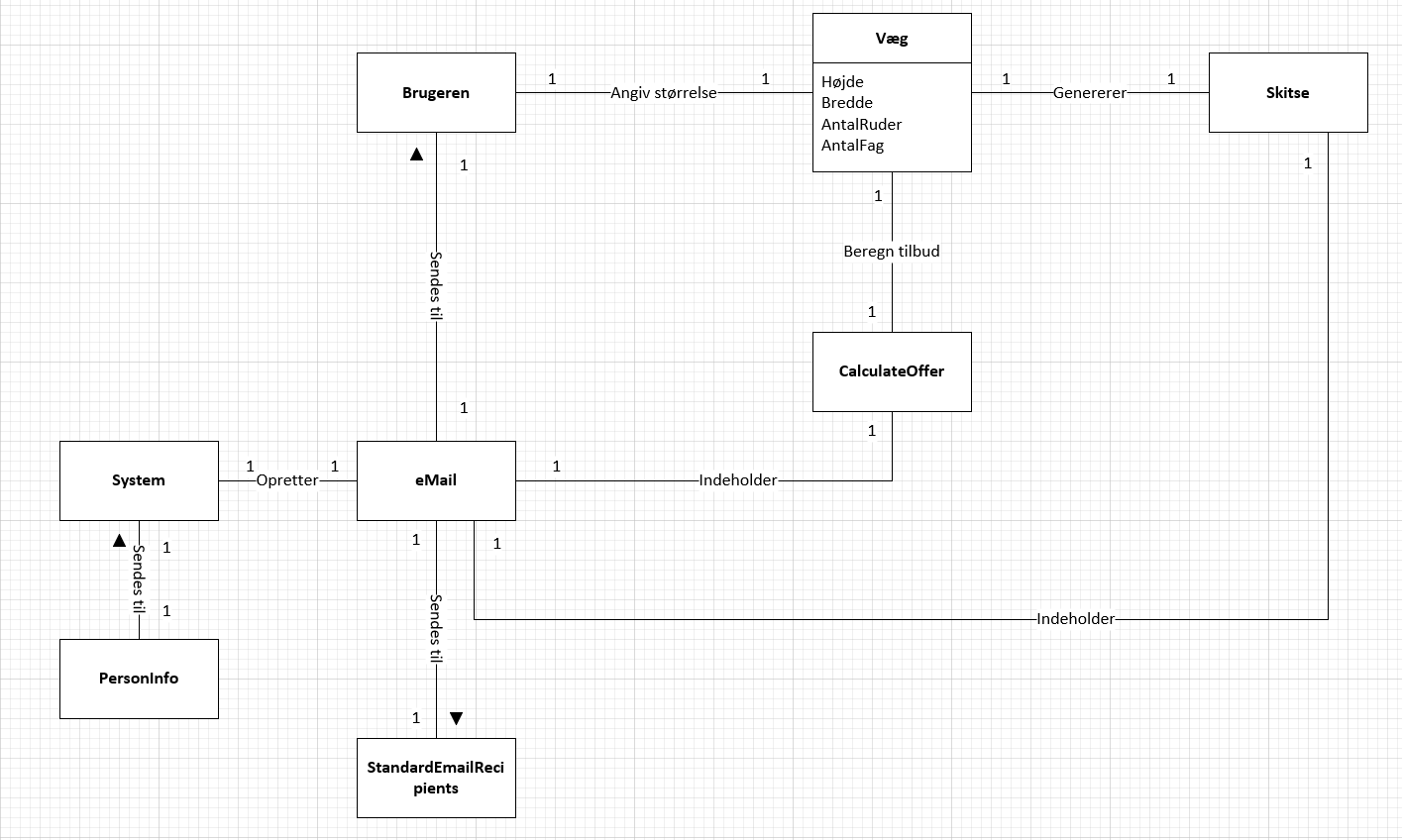
I UML notation er en domæne model illustreret med et set af klasse diagrammer, uden nogen definerede operationer. I stedet viser domæne modellen:

* Domæne objekter eller konceptuelle klasser
* Forholdene imellem de forskellige konceptuelle klasser
* Attributterne de konceptuelle klasser indeholder

Når man laver en domæne model er det en god ide at starte med at lave en liste af konceptuelle klasse kandidater. Nedenunder er der et eksempel på en liste af konceptuelle klasse kandidater taget taget fra store og airline domæne: [[1]](#footnote-1)

Ud fra den liste man har lavet, kan man så begynde at vælge fra kandidaterne man har skrevet op, og starte med at bygge sin domæne model.

I vores første iteration har vi lavet en domæne model, som illustrere de konceptuelle klasser i vores program.

[[2]](#footnote-2)

I System klassen bliver for eksempel personlige informationer, som er sendt fra PersonInfo klassen, gemt.

Brugeren snakker til den konceptuelle klasse Væg, hvor brugeren skal angive størrelsen på de fire attributter: højde, bredde, antal af ruder og antal fag, med en multiplicitet på 1 begge veje da brugeren kun skal indtaste Væg størrelser en gang.

Udover brugeren har Væg klassen også et forhold med Skitse klassen. Efter Brugeren har indtastet sine størrelser, kan Væg klassen fortælle Skitse klassen at den skal generere en skitse af væggen for Brugeren (kunden) og samt forhandler til at se på.

Væg klassen er også forbundet til CalculateOffer klassen, hvor i et tilbud på en pris kan blive beregnet, ud efter hvad Brugeren indtaster af mål i Væg klassen.

eMail klassen er forbundet med flere forskellige klasser. Den er forbundet med: Brugeren, CalculateOffer, Skitse, StandardEmailRecipients og til sidst System. Efter der er blevet generet en skitse og beregnet et tilbud, vil System klassen sende en besked til eMail klassen om at den skal oprette en email, der indeholder informationer fra CalculateOffer klassen og skitsen fra Skitse klassen. eMail klassen vil så sørge for at få sendt denne email til både brugeren, og til StandardEmailRecipients, som er forhandleren og producenten.

Multiplicitet: Multiplicitet (tallene uden for klassen) i en domæne model, bruges til kommunikere hvor mange instanser kan blive associeret med hinanden lige på dette tidspunkt, og ikke over en større længde af tid. For eksempel forholdet i mellem Væg og Skitse klassen, en instans af Væg klassen vil kun generere en instans af Skitse klassen, så derfor står der 1 ud fra de to klasser.

System Sekvens Diagram (SSD) (Simon)

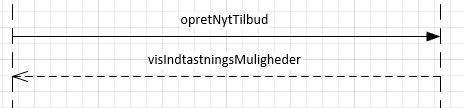
Et system sekvens diagram (forkortet til SSD), er et nemt at lave og overskueligt artefakt der illustrerer de input en bruger laver i et system, og hvilket slags output der sker i følge af input.

Når man laver et program, er det en god ide at kunne forstå, hvordan en bruger af dit program vil interagere med programmet, for at få en bedre forståelse for hvordan programmet er opbygget.

I en SSD går tiden nedad, og SSD’ens events skal følge den samme opstilling som den use-case man skriver sin SSD ud fra.

Når man vil lave sin SSD, starter man med at putte en ’Bruger’ ind, og derefter laver en lifeline der hænger på brugeren. En lifeline repræsenterer den tid brugeren bruger programmet i.

I vores gruppes SSD (som set nedenunder) starter brugeren med at trykke på en knap, for at oprette et nyt tilbud, dette sender en besked til applikationen om at den skal køre metoden opretNytTilbud(). Dette illustreres ved en streg med en pil fra brugerens lifeline, pegende hen imod Newyorker applikationens lifeline. Newyorker applikationen vil så besvare brugerens request, ved at vise de forskellige muligheder for indtastning der findes. Dette svar fra applikationen vises ved, at der fra Newyorker applikationens lifeline, går en stiblet streg med en pil hen imod brugerens lifeline.

[[3]](#footnote-3)

Længere nede af lifelinen er der en boks, hvor i øverst til venstre står der ’alt’, dette ’alt’ står for ’alternative’ og det er til for at vise at alt inde i alternative boksen, er en alternativ mulighed til at oprette et tilbud.

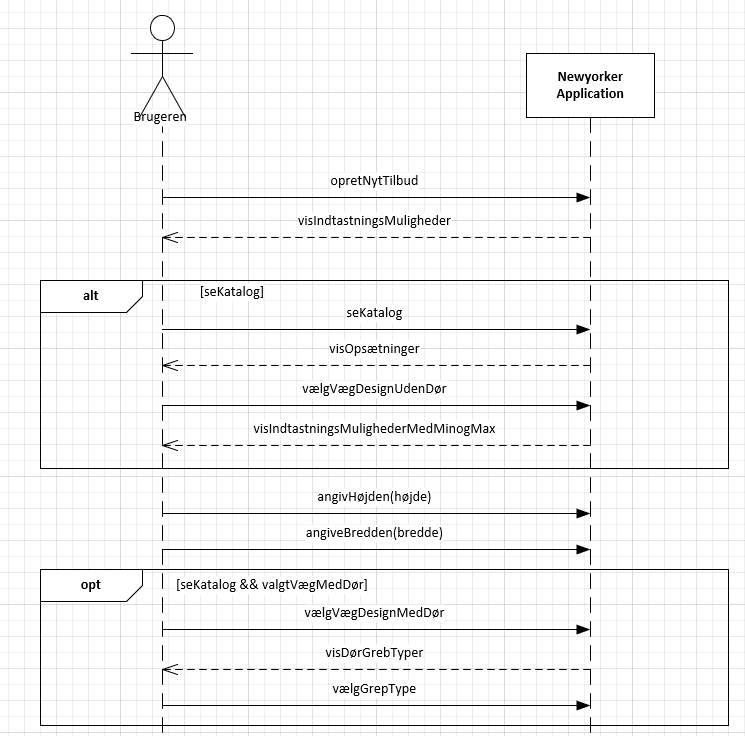
Efter man starter programmet, har man den mulighed at trykke på en knap for at se på kataloget. Hvis brugeren vælger at trykke på knappen for kataloget, vil der sendes en besked til applikationen, som vises med stregen med pilen der pejer mod applikationens lifeline, om at kataloget skal frem på skærmen. Applikationen vil da svare brugeren ved at vise de mulige opsætninger af væg, som vises med den stiblede streg hvor pilen pejer mod brugerens lifeline.

Brugeren kan så vælge det vægdesign som de ønsker, og efter brugeren har valgt et design, vil systemet vise brugeren hvilken højde og bredde der er muligt med det valgte design.

Ned ad lifelinen igen, bliver brugeren bedt om at indtaste hvilke højde og bredde mål de skal bruge på deres væg, og dette bliver sendt til systemet.

Der er endnu boks nu, hvor der øverst til højre står ’opt’, hvilket står for ’optional’. Dvs at alt inde i opt boksen er valgfrit, og ikke noget man behøver at skulle gøre som bruger, for at komme videre i programmet.

’Opt’ boksen beskriver et scenarie, hvor at brugeren vil trykke på knappen for at se kataloget, og derefter vælge et vægdesign med en dør. Det starter med at brugeren vælger at se kataloget lige som i ’alt’ boksen ovenunder, dog i dette scenarie vælger brugeren et vægdesign med dør. Systemet svarer brugeren ved at vise en liste og mulige dørgrebstyper, hvor efter at brugeren skal vælge en af de viste dørgrebstyper.

[[4]](#footnote-4)

Operationskontrakter (OC) (Simon)

En operationskontrakt, også forkortet til OC

1. Eksempel liste fra bogen, Applying UML and Patterns, side 134. [↑](#footnote-ref-1)
2. DM\_LavEtNytTilbud\_Iteration1 [↑](#footnote-ref-2)
3. Udsnit af SSD\_LavEtNytTilbud\_Iteration2 - Øverste pil sender systemet et request, nederste stiblede pil er et svar til brugeren fra systemet. [↑](#footnote-ref-3)
4. Udsnit af SSD\_LavEtNytTilbud\_Iteration2 [↑](#footnote-ref-4)