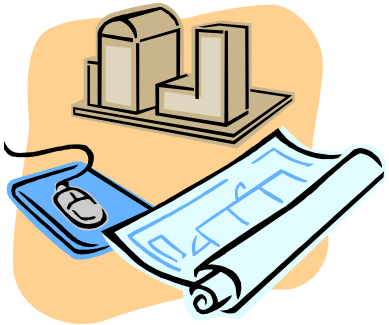
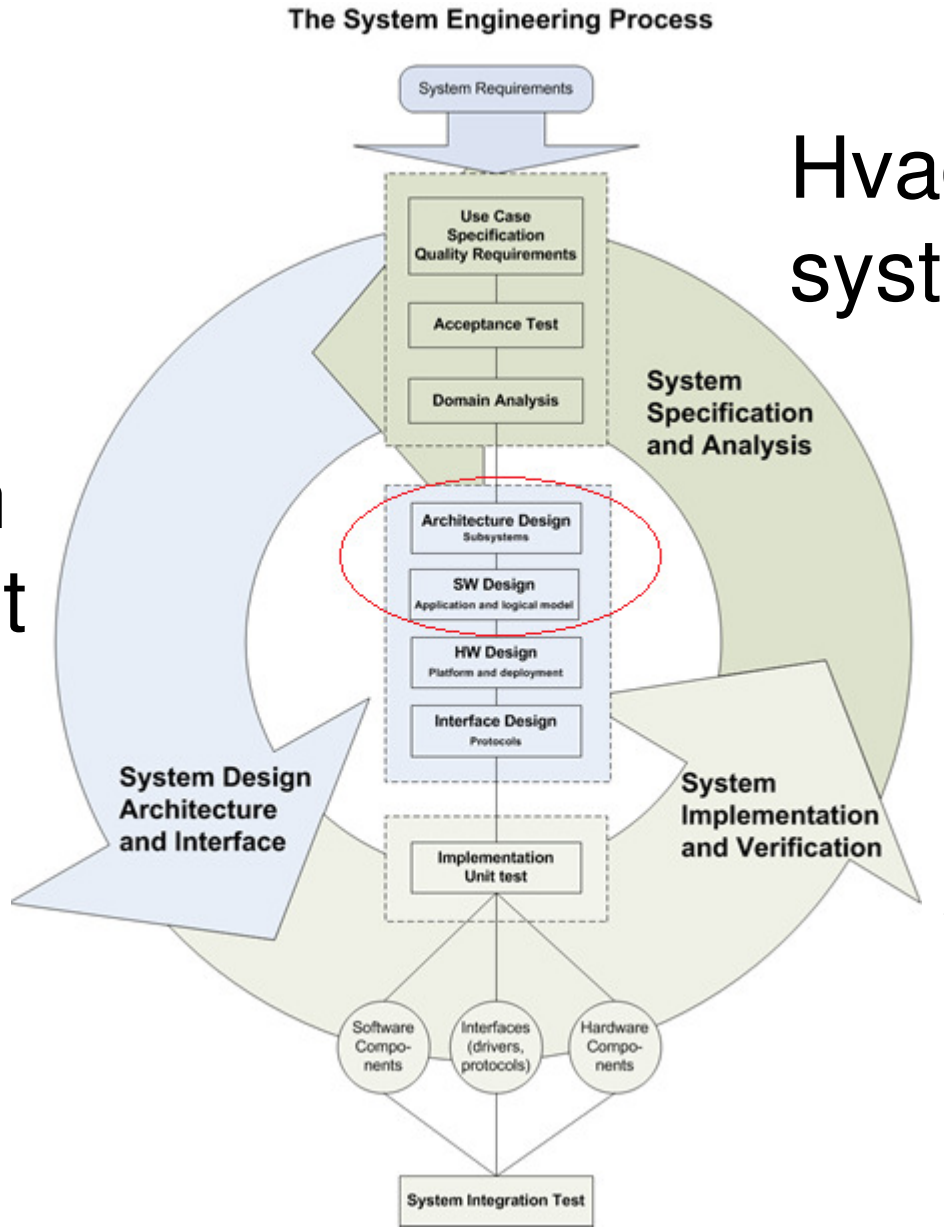
ISE

Applikationsmodel



Hvordan systemet gør det?



Hvad skal systemet gøre?

Virker systemet?

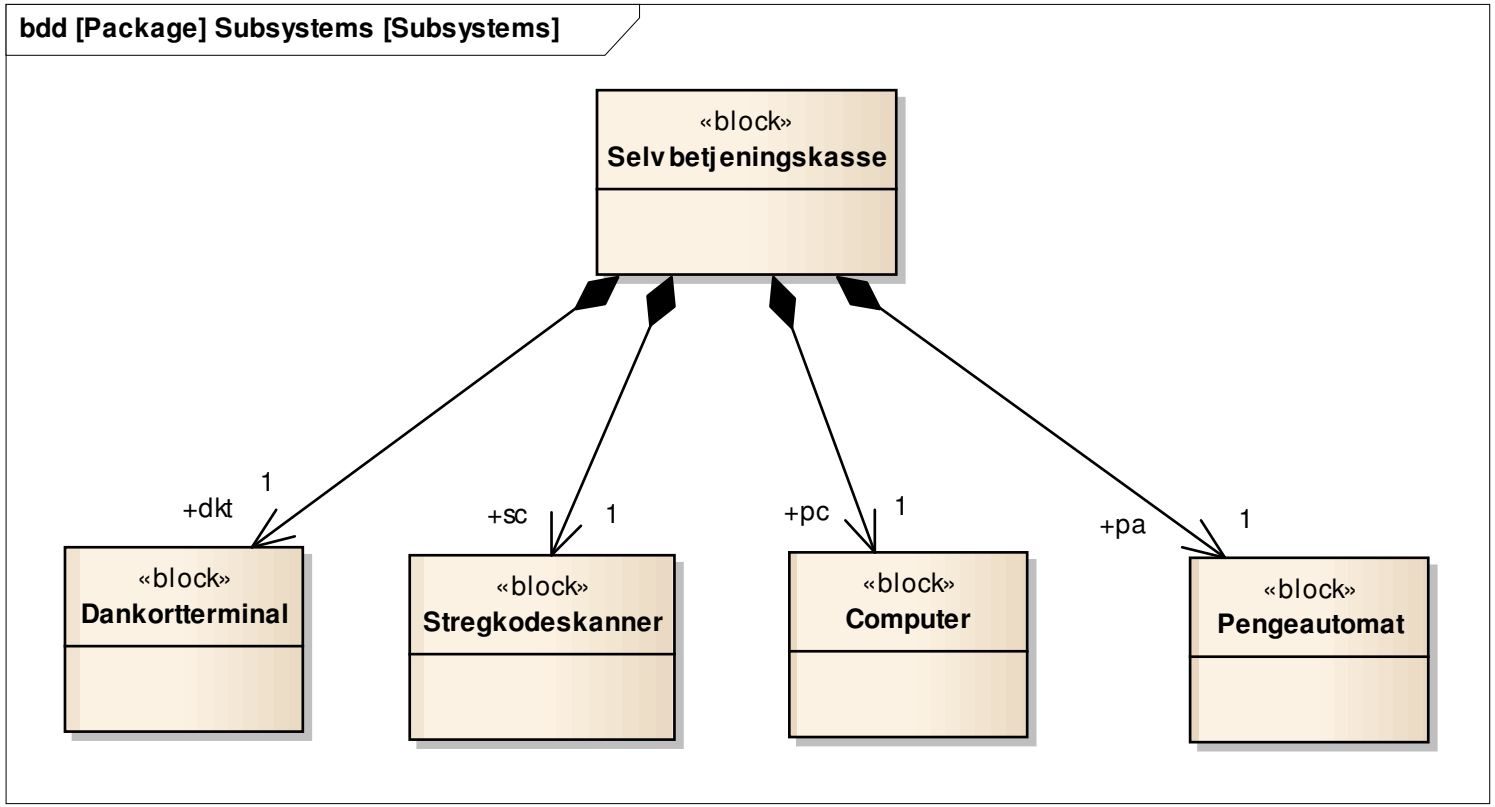
Fra domain til applikationsmodel (Software)

System Arkitektur - Subsystemer

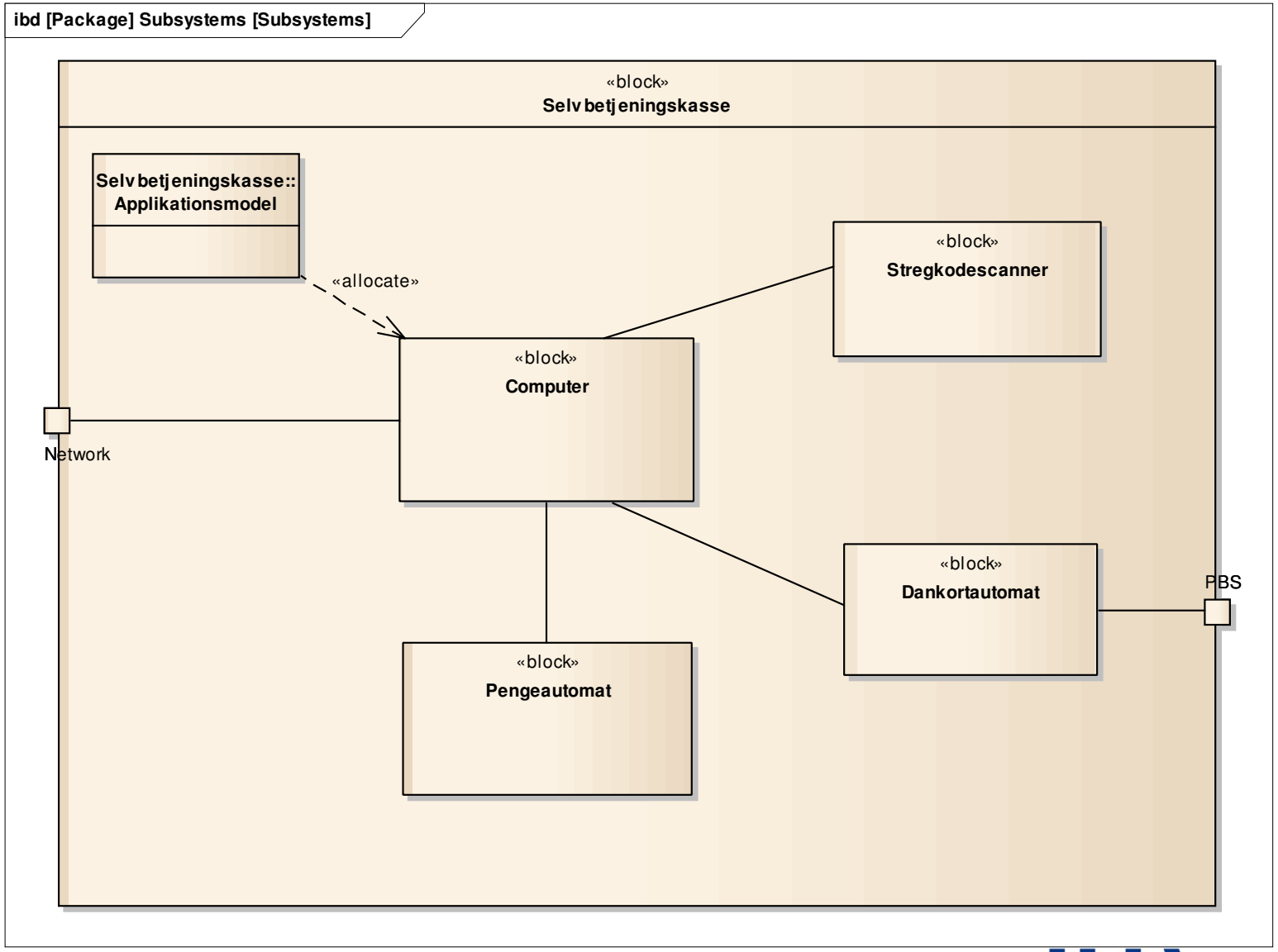
- Systemet opdeles i delsystemer
- Fastlæggelse af hvilke computere der indgår i systemet
- Fastlæggelse af hardware arkitektur som er grundlaget for applikationsmodellen
- Arkitektur består af processering og kommunikation
- Der fastlægges en applikationsmodel for hvert delsystem (computere)

Selvbetjeningskasse

Subsystemer

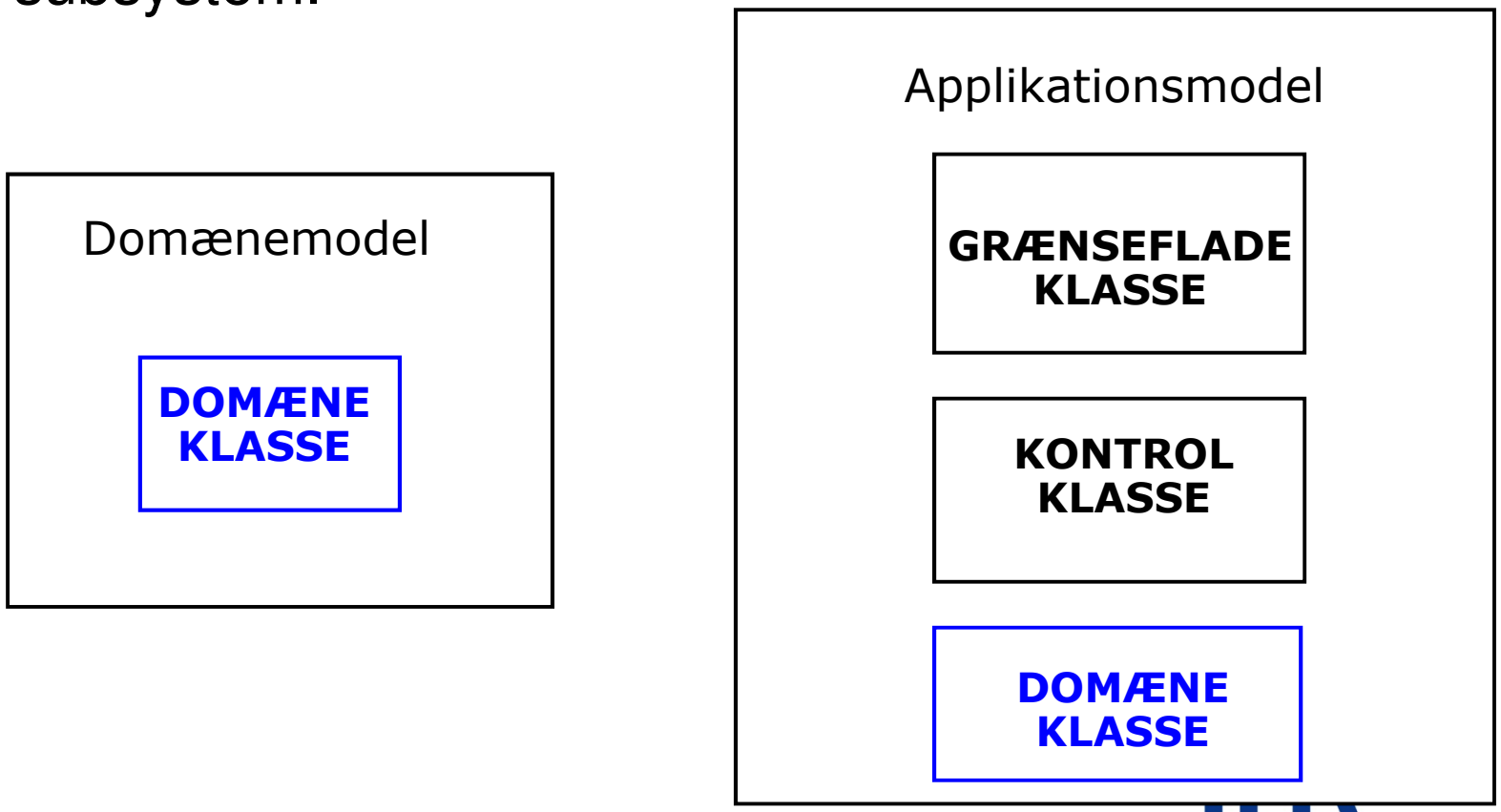


Arkitektur for selvbetjeningskasse



Applikationsmodel

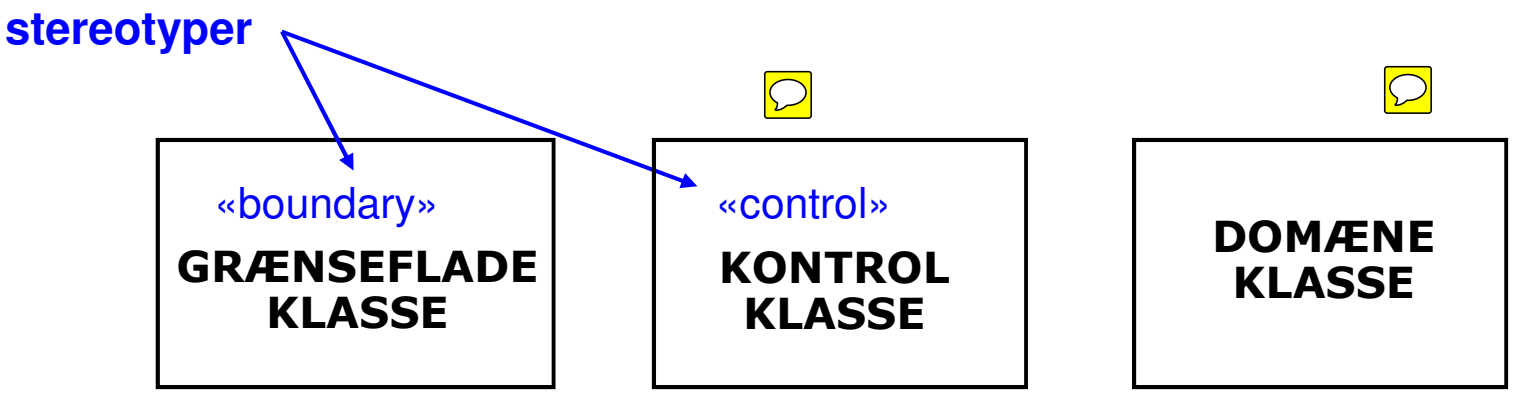
- Domænemodellen beskriver problemdomænet, men ikke selve applikationen.
- Applikationsmodellen beskriver applikationen for et givent subsystem.



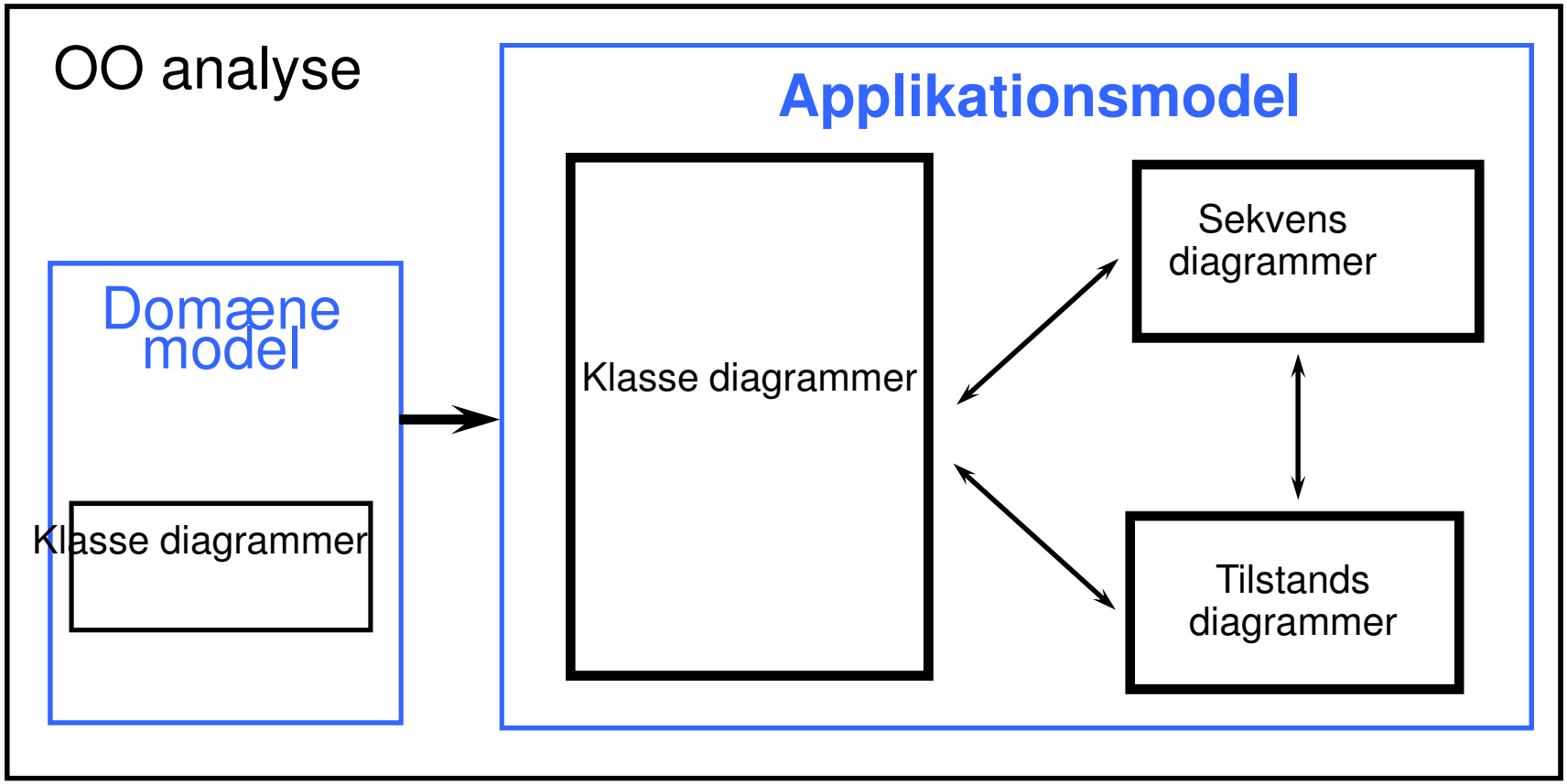
Applikationsmodellens 3 klassetyper

1. Grænsefladeklasser («boundary »)
   - repræsenterer grænsefladen til aktørerne på et abstrakt niveau eller tager udgangspunkt i konceptuelle domain klasser
2. Kontrolklasser («control »)
   - repræsenterer funktionalitet/forretningslogik (Use Cases)
3. Domæneklasser

Typen indikeres på et klassediagram vha. UML's stereotype



Oversigt over modellerne



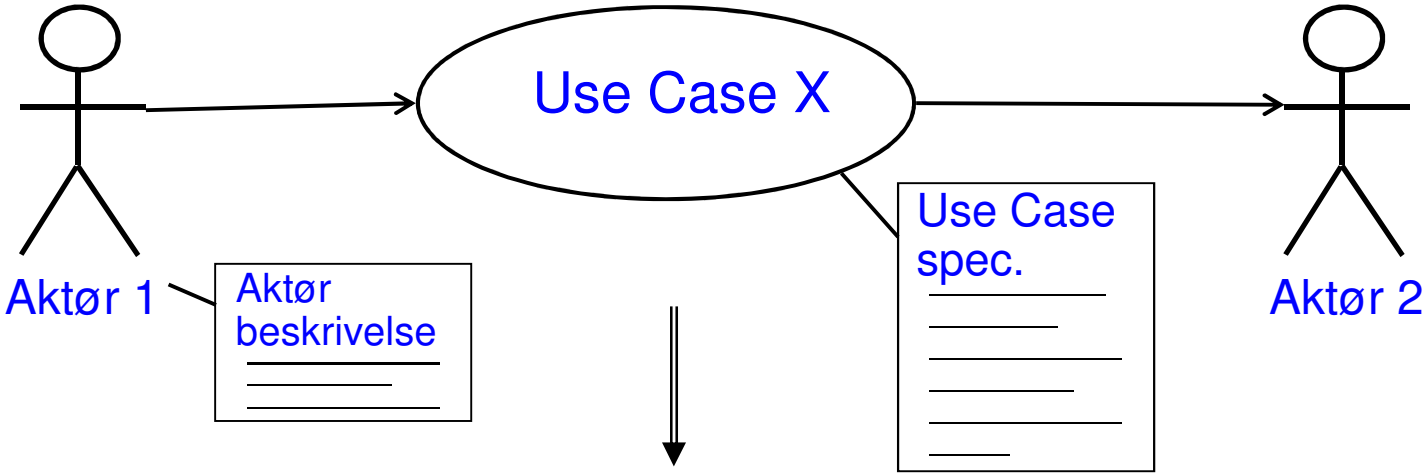
Applikationsmodellen og diagrammerne

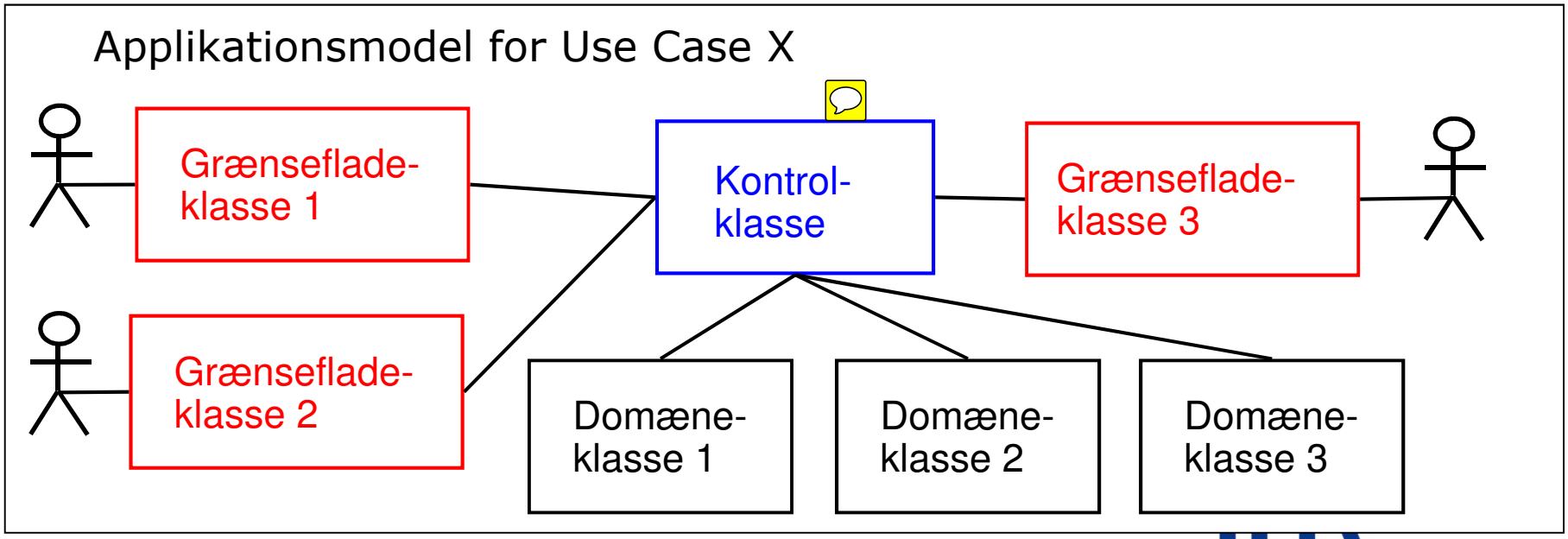
Applikationsmodellen ud over klassediagrammer også kan indeholde interaktionsdiagrammer og tilstandsdiagrammer.

Disse to diagrammeringsformer anvendes til at finde de operationer, der skal allokeres til klasserne i applikationsmodellen.

Hver klasse, der har en tilstandsafhængig opførsel, kan beskrives vha. et tilstandsdiagram (UML Statechart).

Udvælg en Use Case til modellering



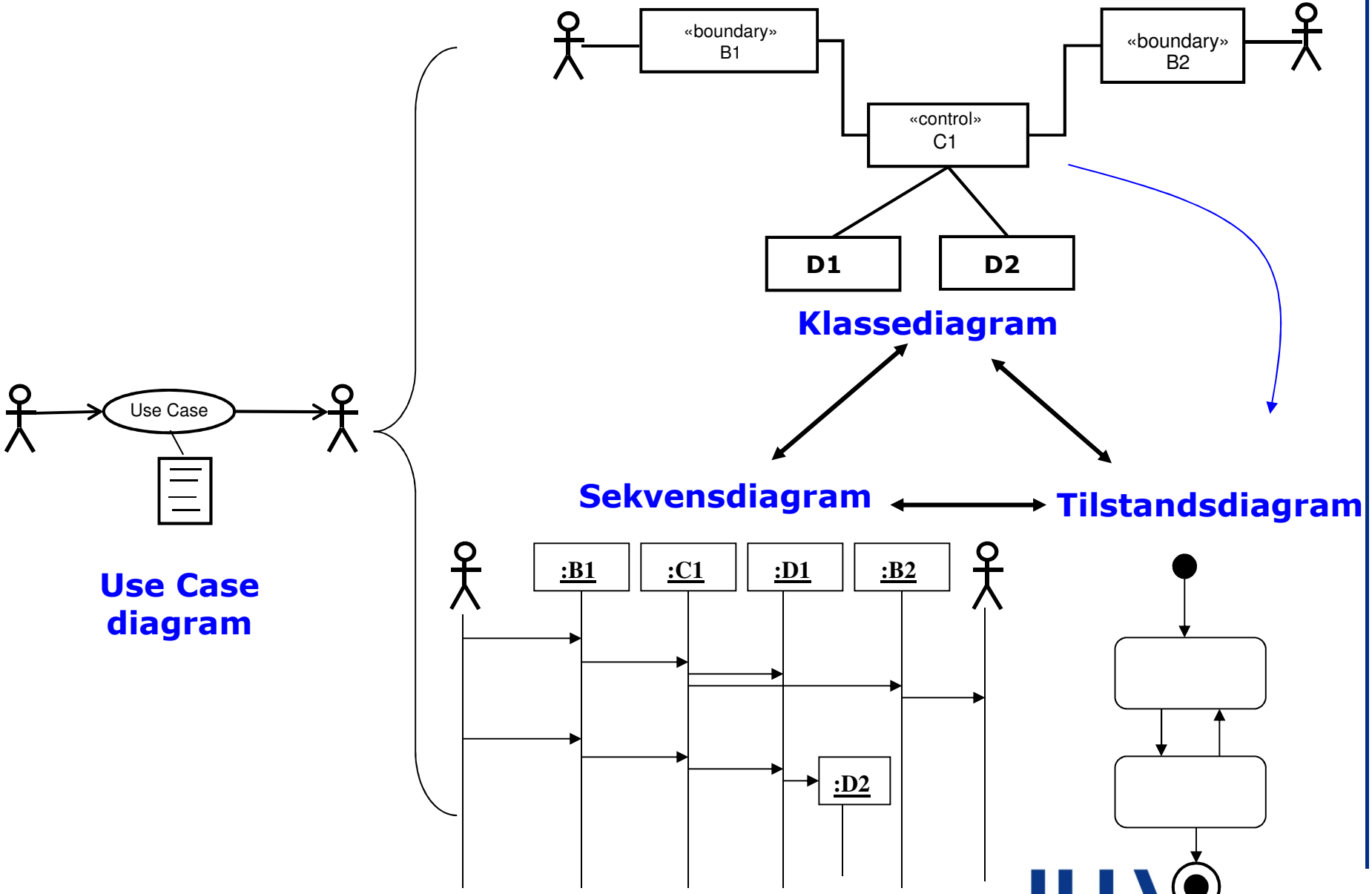


Iterationer og Use Cases

Først vælger man en af de Use Cases, der indgår i den planlagte iteration. Er Use Casene prioriteret, så vælger man den med den højeste prioritet. Der er normalt kun de selvstændige Use Cases, der udvælges, da evt. include og extend Use Cases modelleres sammen med en tilhørende selvstændig Use Case.

Hvis Use Casen *ikke* er specificeret på nuværende tidspunkt - så skal Use Case specifikationen først udfyldes, før man kan gå videre med de efterfølgende metode trin.

Fra Use Case til applikationsmodel

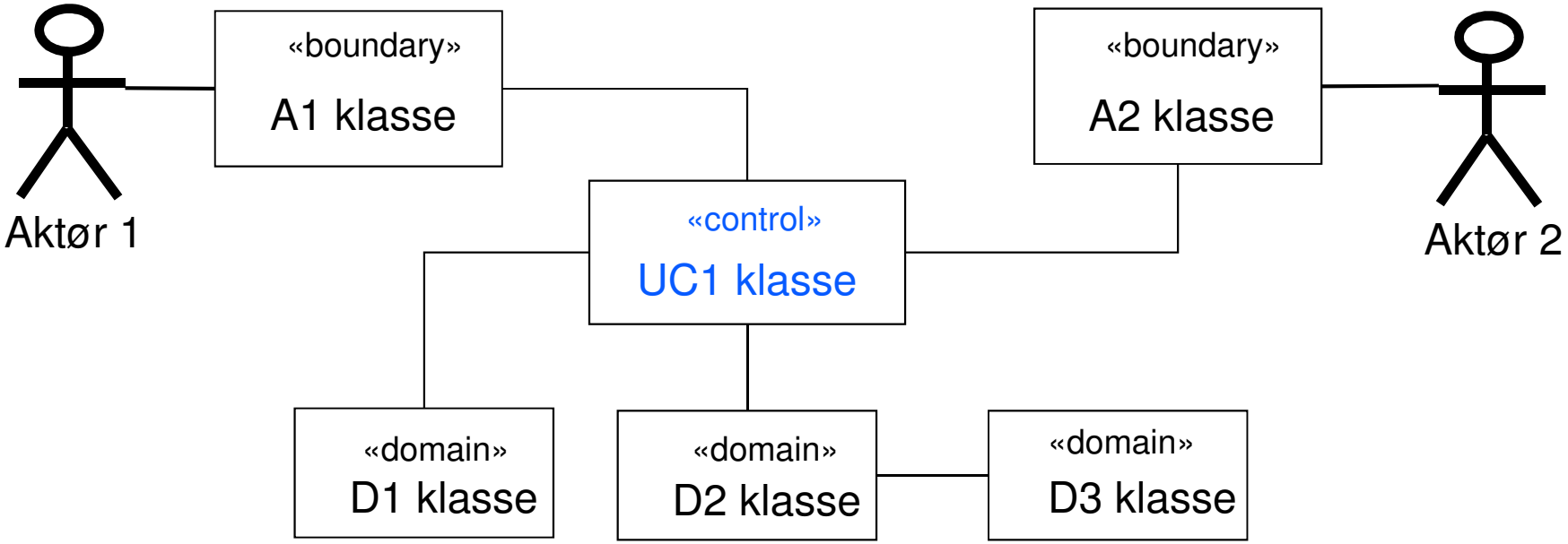


Fra Use Case til OO-model (1)

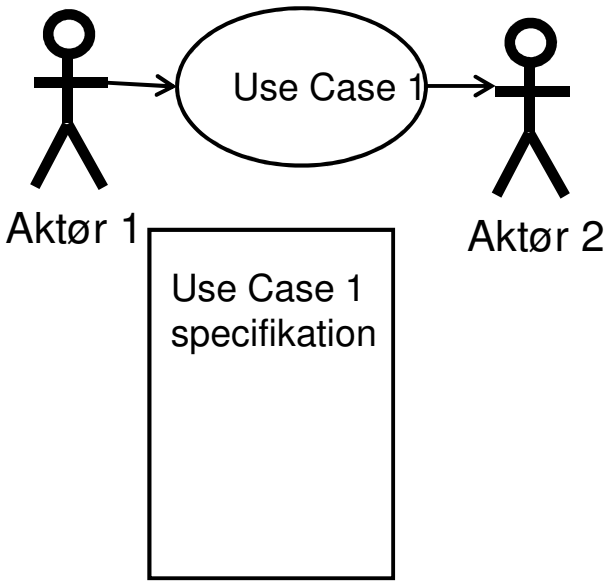
1. Udvælg en Use Case, der har betydning for arkitekturen
2. Tag udgangspunkt i OO analysemodellerne for denne (Specifikation og relevante begreber i domæne model)
3. Overvej og skitser et eller flere løsningsalternativer
4. Overvej anvendelsen af arkitekturmønstre, lagdelt…
5. Indfør de nye designklasser og opdater OO modellerne (diagrammer + diagramelement information)
6. Udarbejd et system arkitekturdokument (~design dokument)
7. Er der flere arkitektursignifikante Use Cases i iterationen – så fortsættes ved pkt. 1.

Domæneklasser: ”Konceptuelle begreber I problemområdet”

Fra Use Case til OO-model (2)

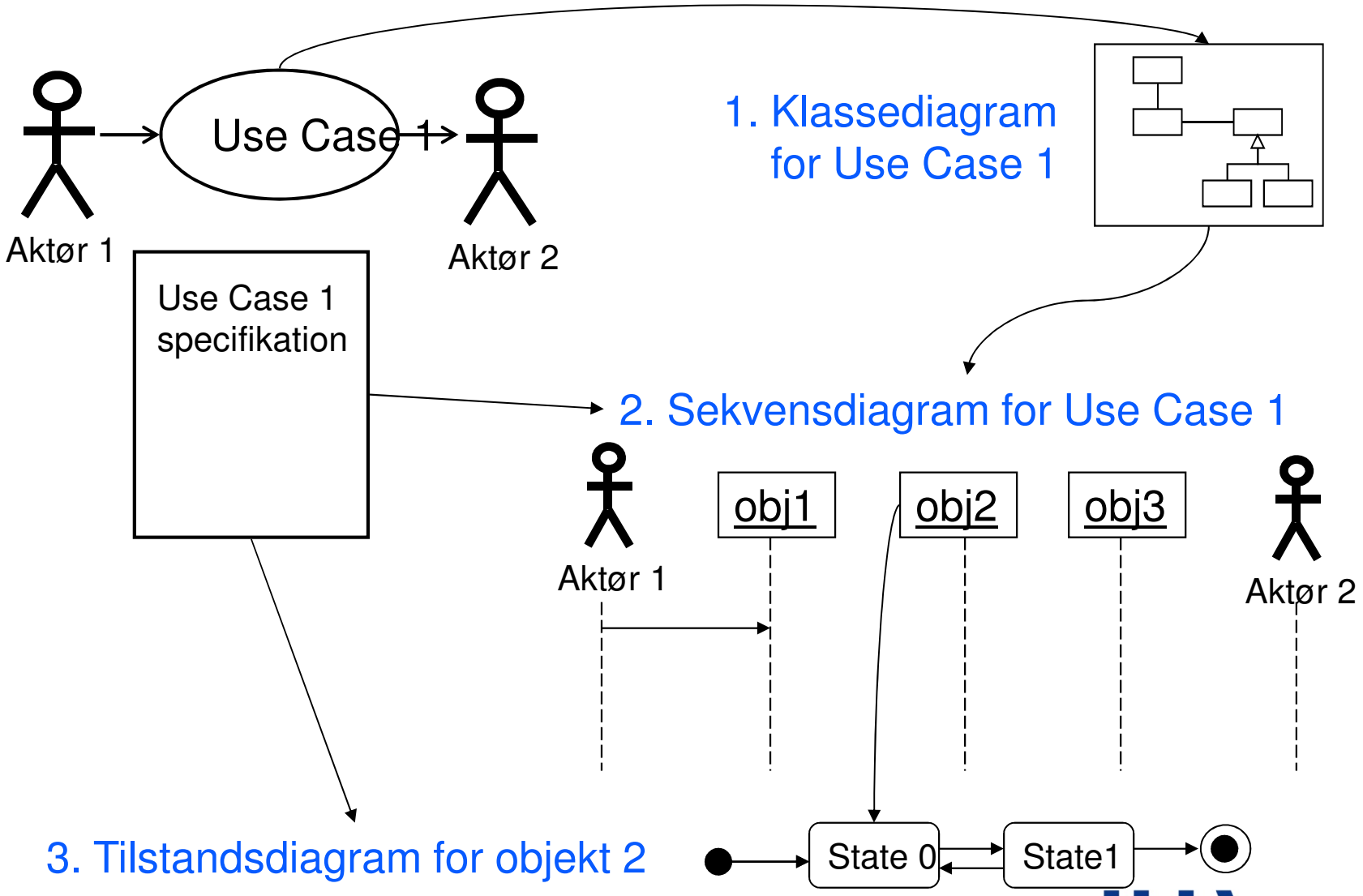


Fra Use Case til OO-model (3)



1. Find hændelser mellem aktører og Use Case
2. Find tidsafhængige hændelser (systeminitierede Use Cases)
3. Start altid med Use Case normal-forløbet
4. Tegn et sekvensdiagram for normalforløbet.
5. Tegn tilstandsdiagrammer
6. Verificer diagrammerne indbyrdes og mod Use Casen
7. Forsæt med undtagelsesscenarier

Fra Use Case til OO-model (4)



Fra Use Case til OO-model (5)

Vigtigt:

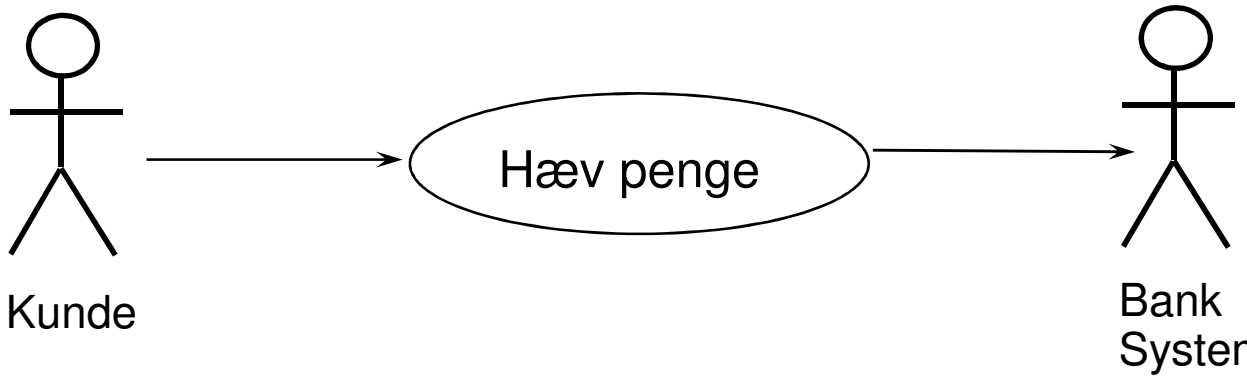
- udgangspunktet er en Use Case specifikation.
- at man parallelt og iterativt arbejder på forfining af klassediagram, sekvensdiagram og tilstandsdiagram.
- arbejdet med sekvensdiagram og tilstandsdiagram giver kandidater til klassens metoder og attributter, der påføres klasserne.
- sekvensdiagrammer er vigtige for at forstå funktionaliteten.

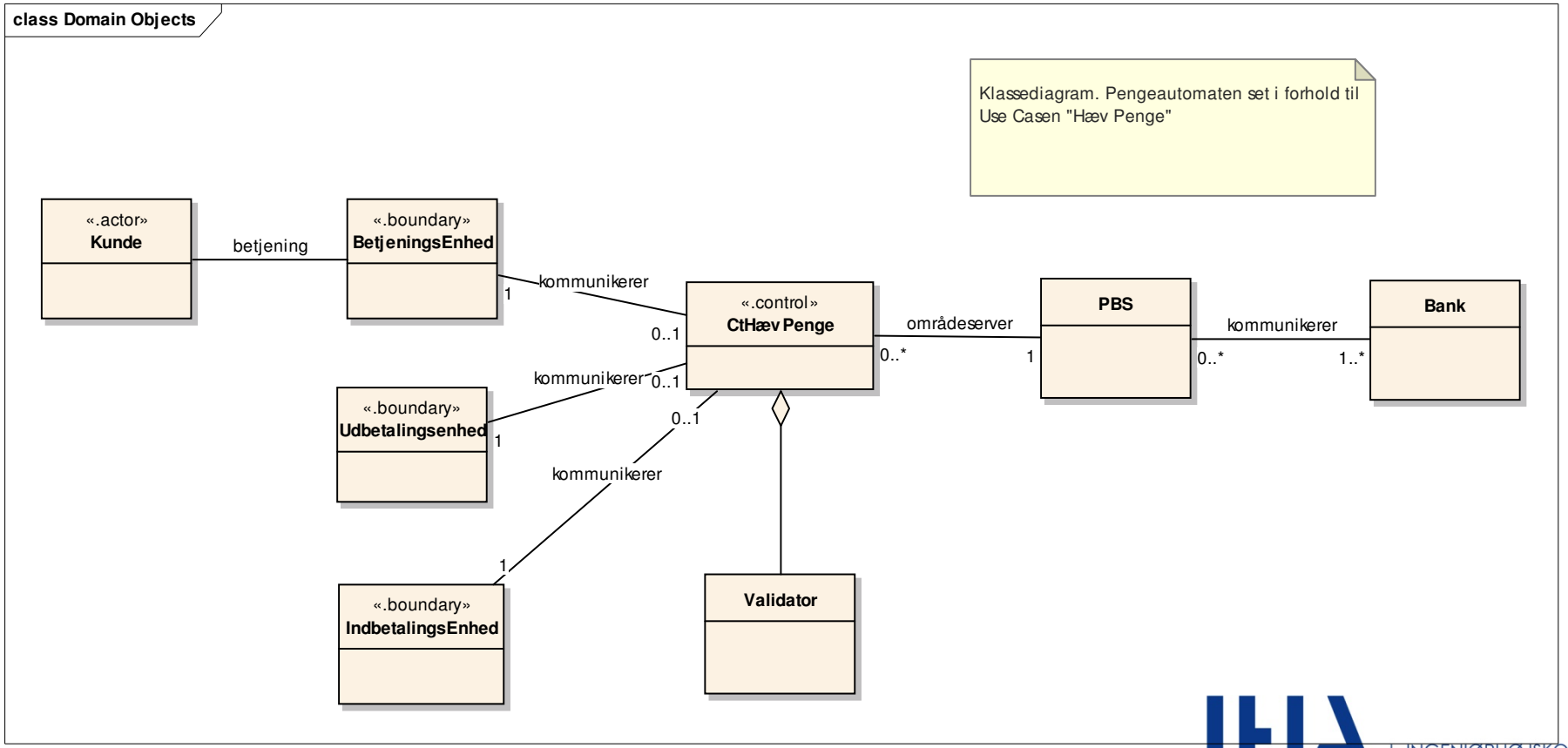
Eksempler

Hæveautomat

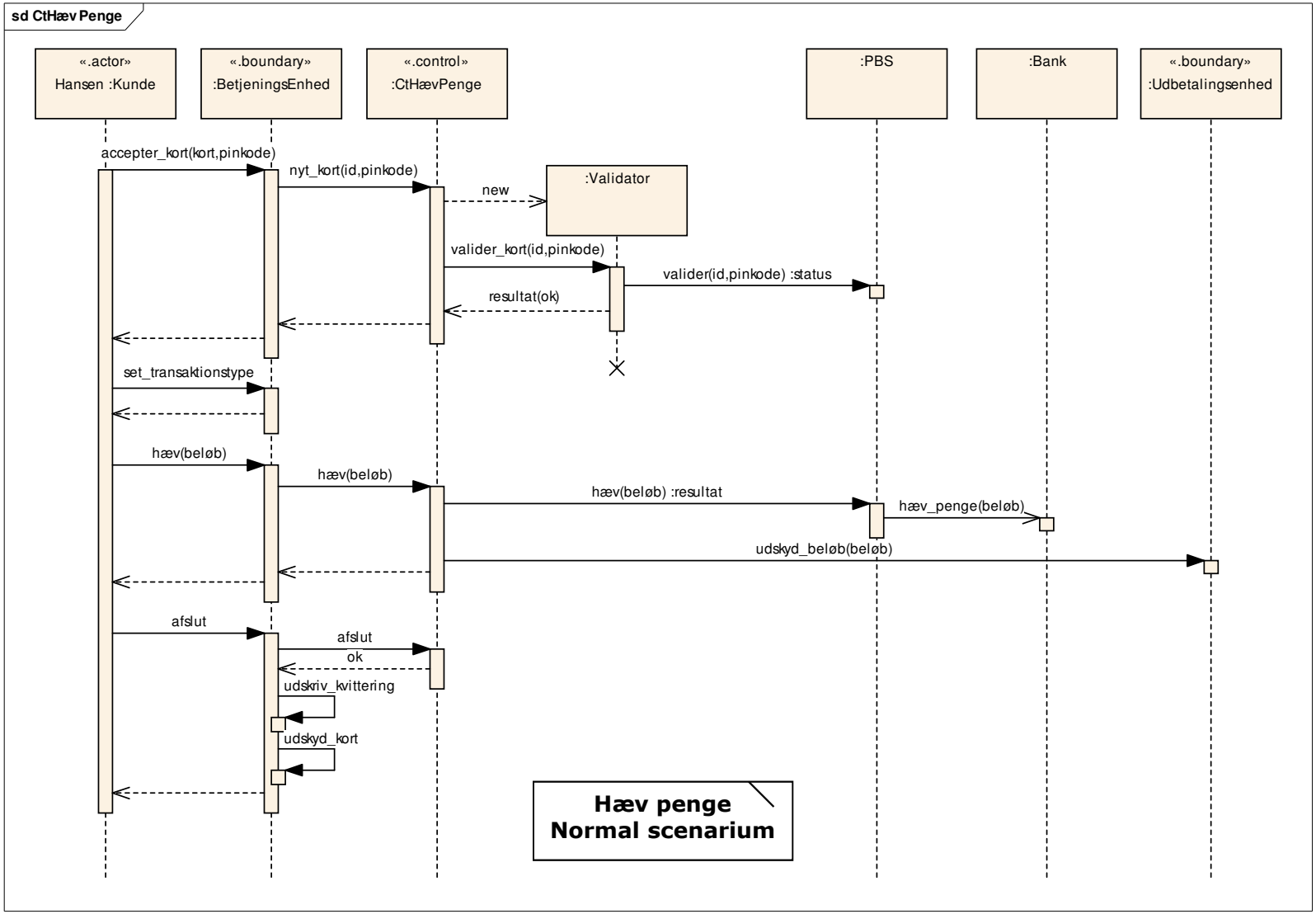
Selvbetjeningskasse

Hæveautomat (1. eksempel)

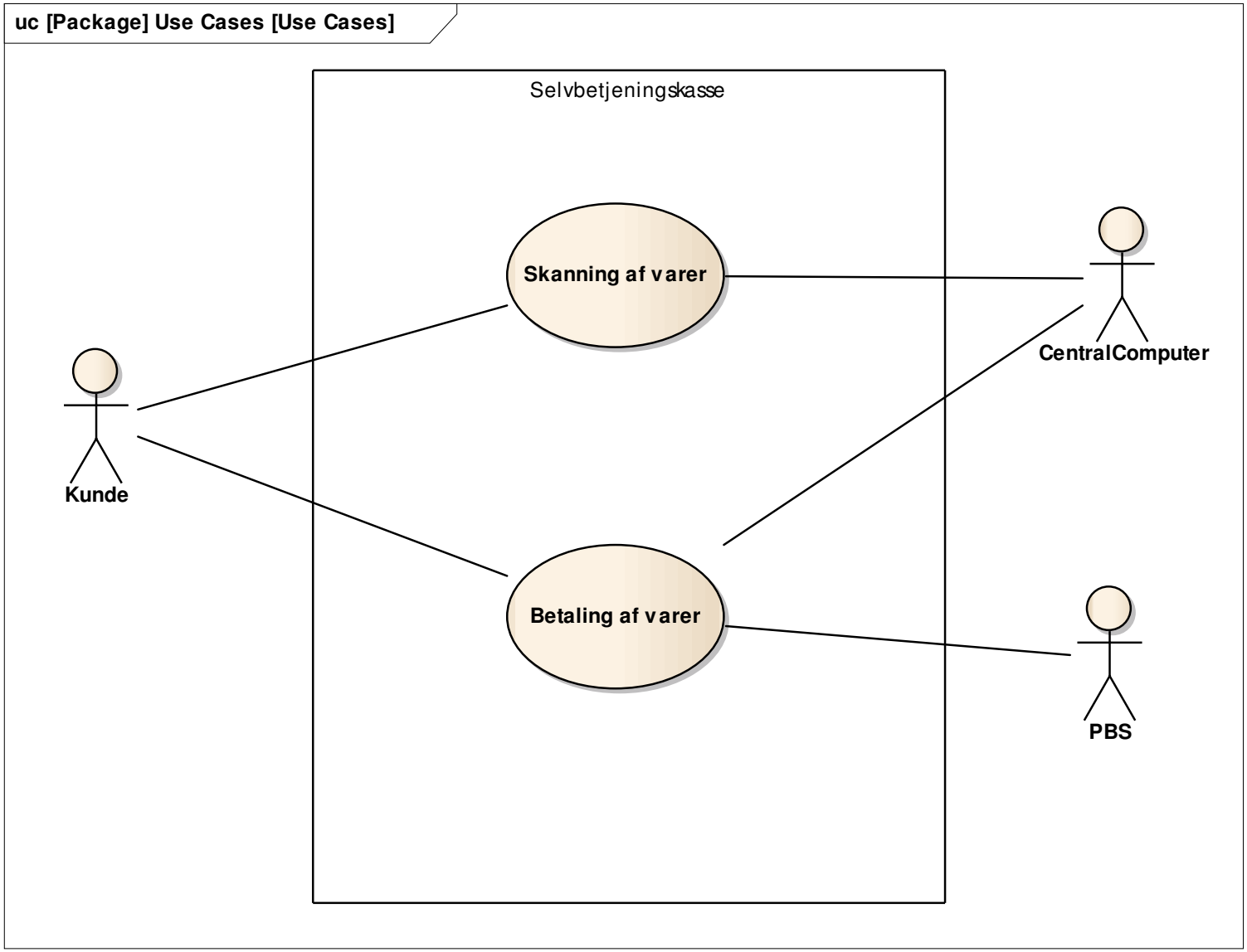




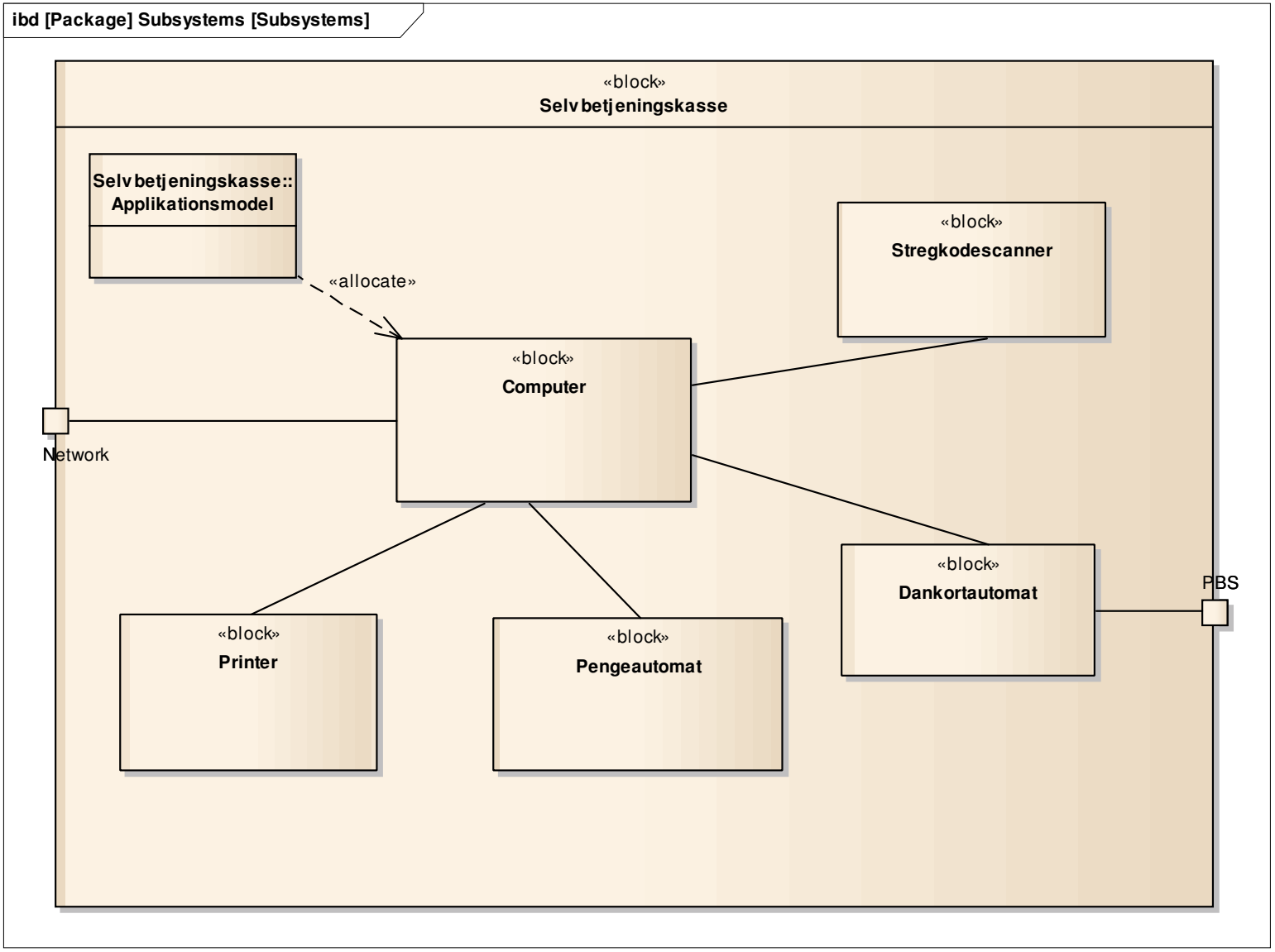
Sekvensdiagram (normal scenarium)



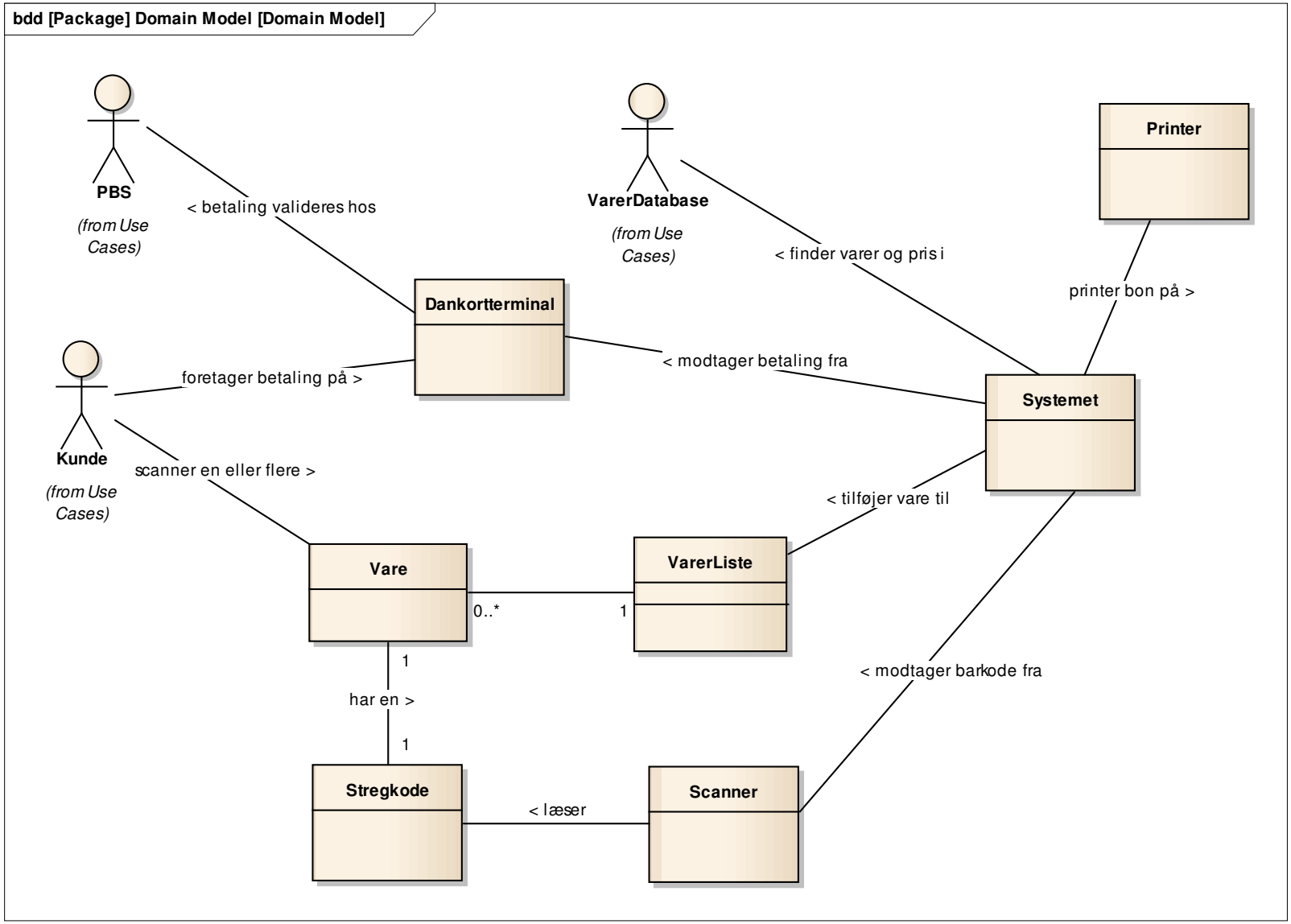
Selvbetjeningskasse (2. eksempel)



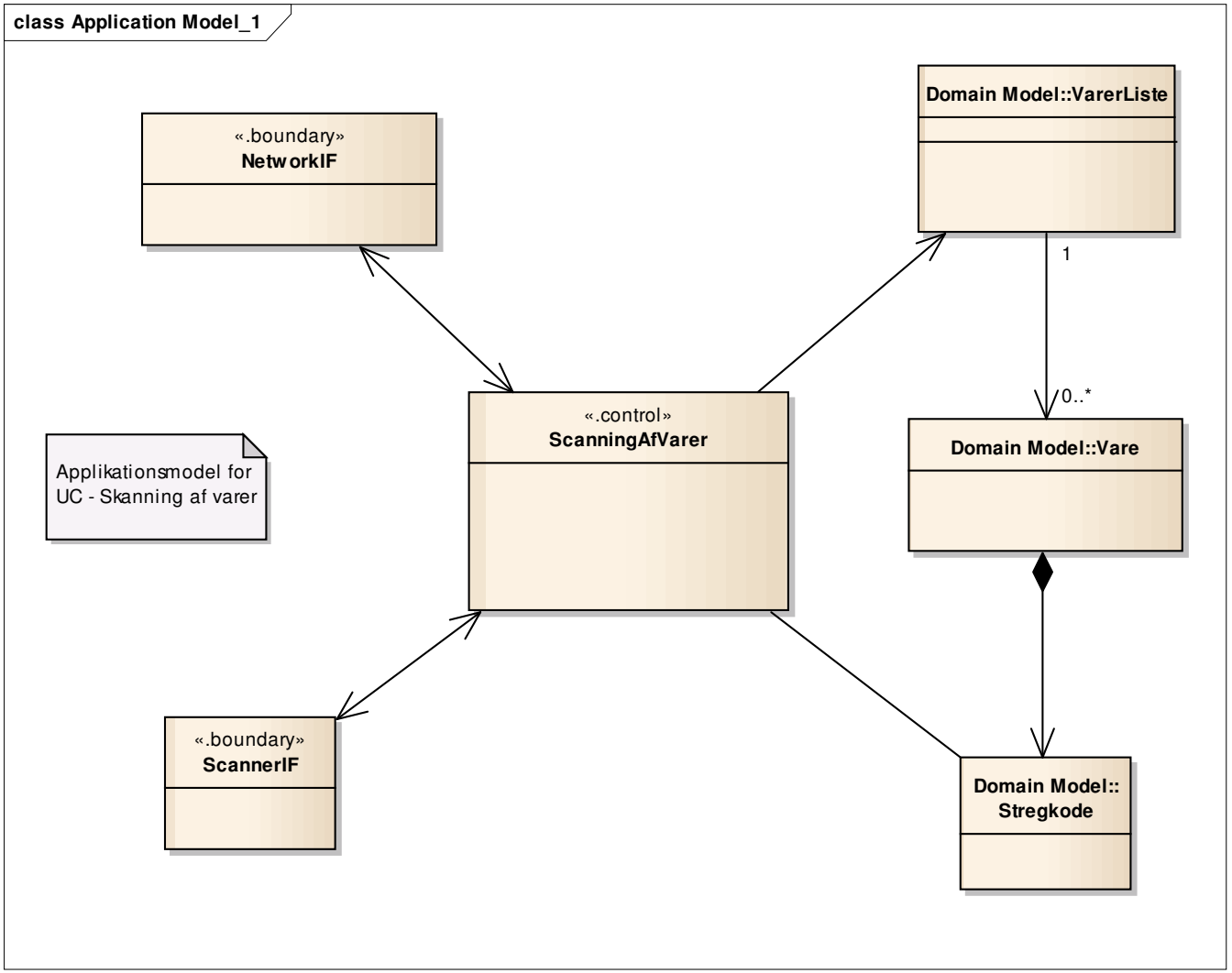
Subsystemer



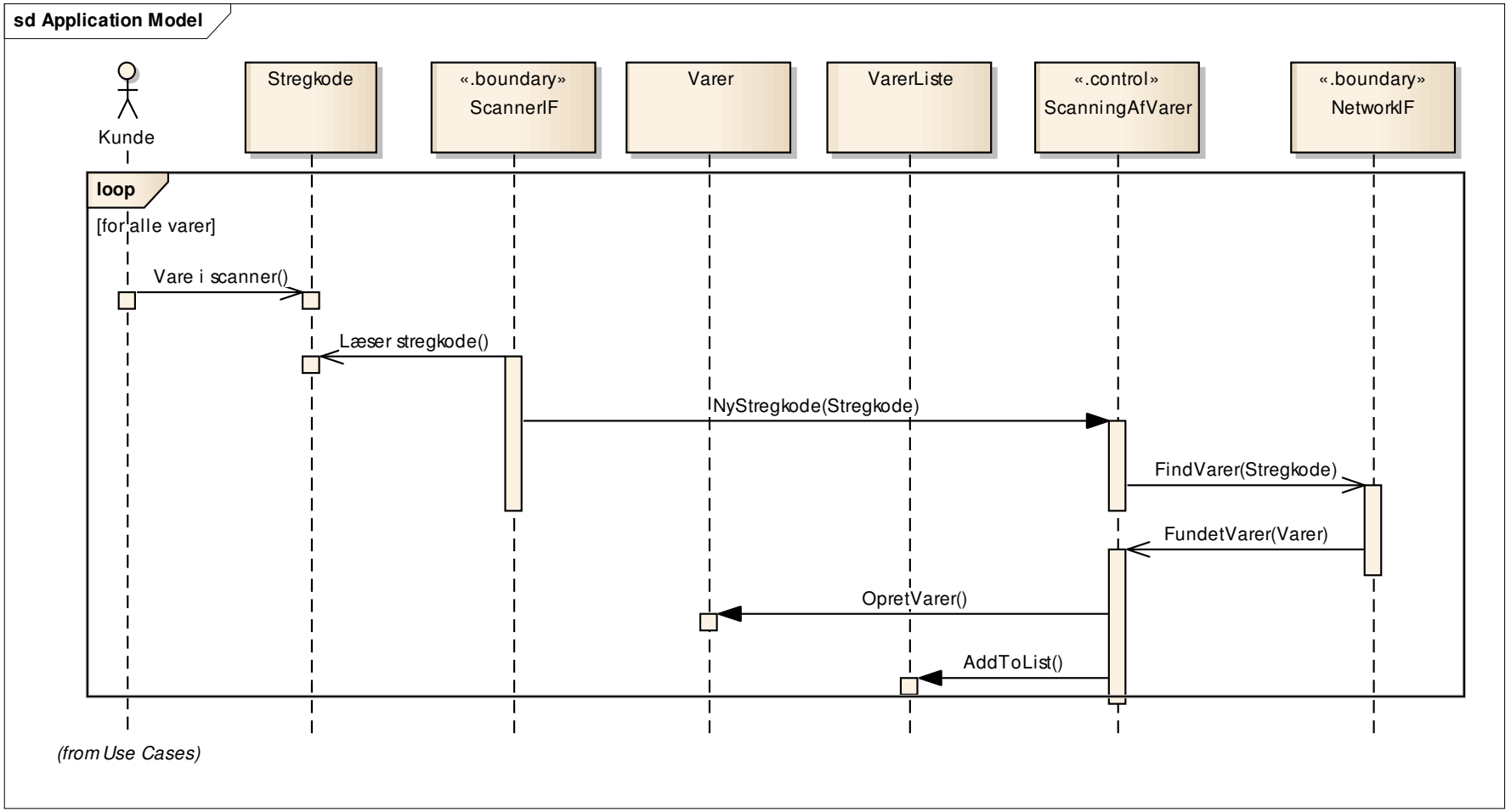
Domain model



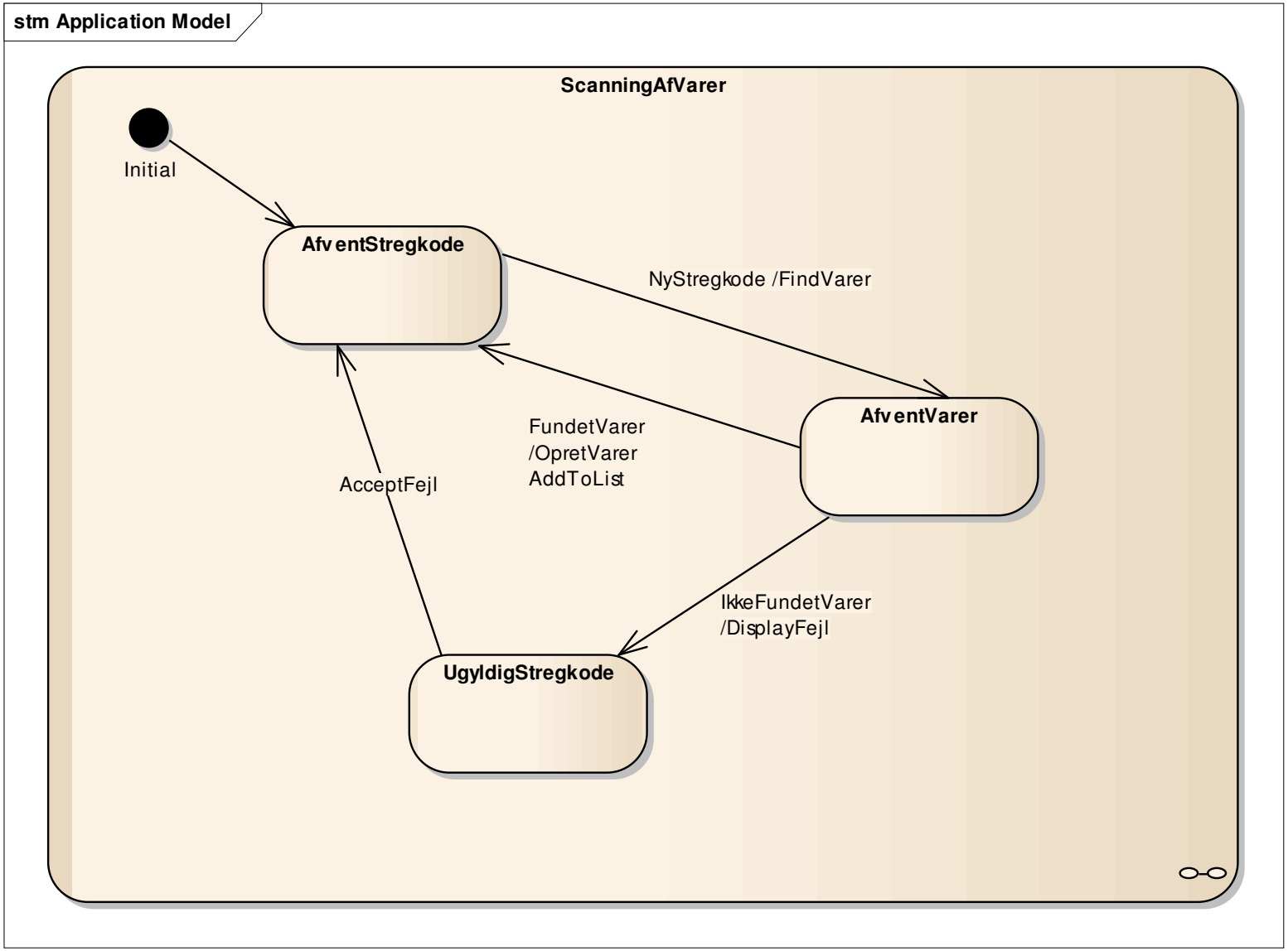
Applikationsmodel (UC – Scanning af varer)



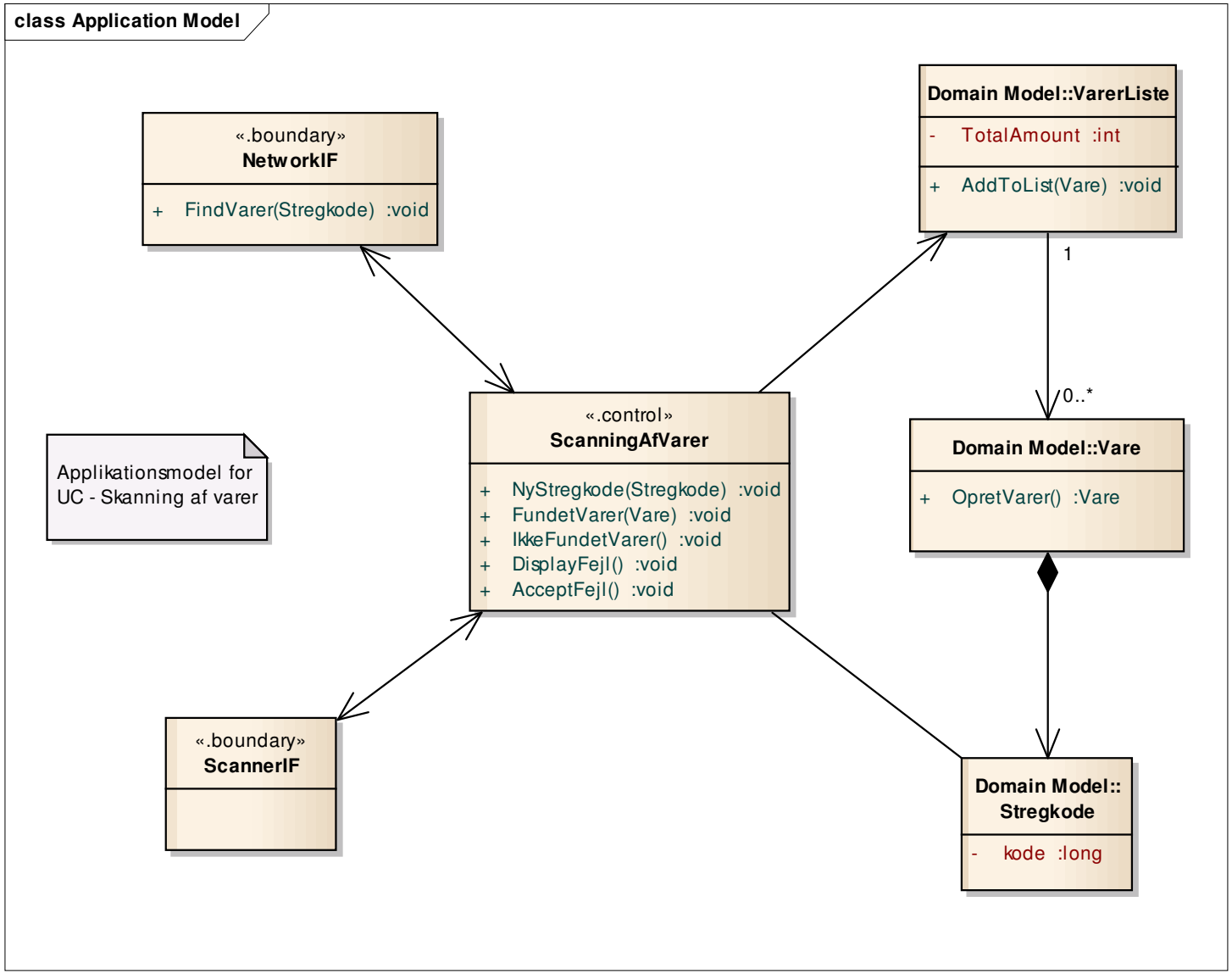
Sekvensdiagram (Applikationsmodel)



Statediagram (Control class)



Opdateret applikationsmodel



C-Koden

```
class ScanningAfVarer
{

public:
    ScanningAfVarer();
    virtual ~ScanningAfVarer();
    NetworkIF *m_NetworkIF;
    ScannerIF *m_ScannerIF;
    VarerListe *m_VarerListe;
    Stregkode *m_Stregkode;

    void NyStregkode(Stregkode stregkode);
    void FundetVarer(Varer varer);
    void IkkeFundetVarer();
    void DisplayFejl();
    void AcceptFejl();

};
```

```
class NetworkIF
{

public:
    NetworkIF();
    virtual ~NetworkIF();
    ScanningAfVarer *m_ScanningAfVarer;

    void FindVarer(Stregkode stregkode);

};
```

```
class VarerListe
{

public:
    VarerListe();
    virtual ~VarerListe();
    Varer *m_Varer;

    void AddToList(Varer varer);

private:
    int TotalAmount;

};
```

```
class Varer
{

public:
    Varer();
    virtual ~Varer();
    Stregkode *m_Stregkode;

    Varer OpretVarer();

};
```