UML-Light

(Note: UML-Light T133, ver. 2004)

Finn Overgaard Hansen, IHA

Programmering PRG1 + Semesterprojekter PRJ1+PRJ2

Version: 20-1-2004

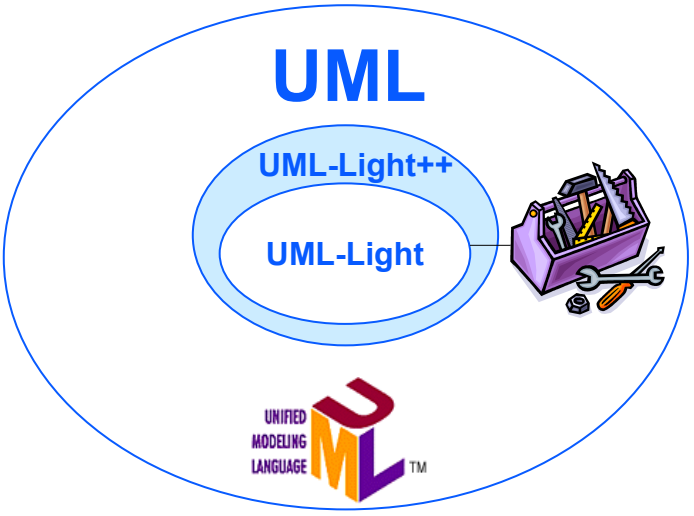
Indhold

- Første del:
  - Introduktion til UML-Light og UML
  - Klasser og objekter
  - Klassediagrammer
  - Tilstandsdiagrammer
  - Sekvensdiagrammer
- Anden del:
  - Sammenhæng mellem UML-Light og kode
  - Gennemgang af eksempler

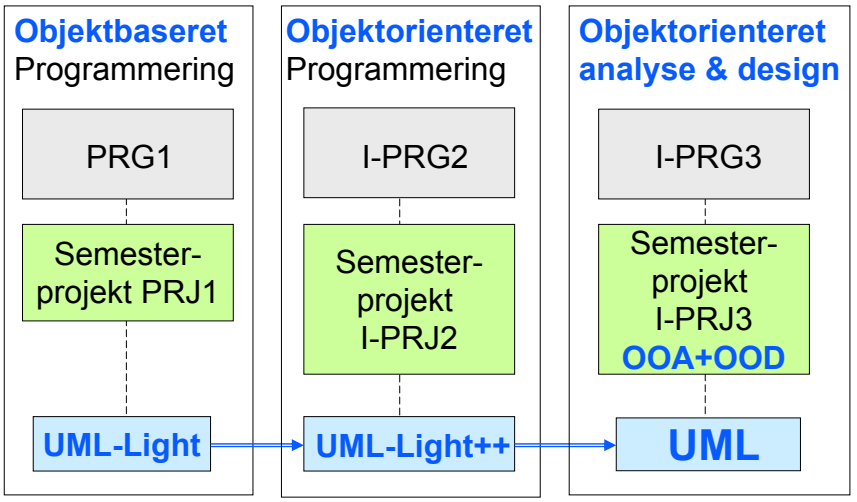
Introduktion til UML

- **UML** (*Unified Modeling Language*) er en OMG standard defineret i 1997
  - Seneste version er UML 1.4 – UML 2.0 er på vej.
- **OMG** (*Object Management Group*) er en sammenslutning af ca. 800 firmaer (www.omg.org)
- UML beskriver en **standardnotation** for objektorienteret udvikling
- UML beskriver **ikke** en udviklingsproces eller udviklingsmetode

UML-Light, -++ og UML



Oversigt over de tre første semestre



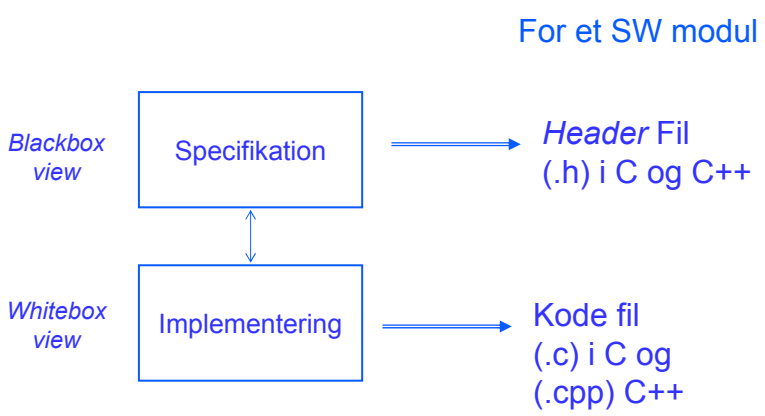
Objektbaseret og Objektorienteret udvikling

- **Objektbaseret udvikling (UML-Light)**
  - Baserer sig på *Information Hiding* og indkapslingsprincippet
  - Anvender klasser og objekter ved modelleringen
  - Kan implementeres i et ikke objektorienteret programmeringssprog som f.eks. C og assembler
- **Objektorienteret udvikling (UML-Light++,UML)**
  - Tilføjer begreber som generalisering/specialisering (nedarvning) og polymorfi (virtuelle operationer i C++)
  - Anvender Use Case teknikken ved kravspecifikation
  - Kræver et objektorienteret programmeringssprog ved kodningen som f.eks. C++, Java eller C#

UML-Light diagramtyper

- De vigtigste UML-Ligth diagrammer er:
  - **Klassediagrammer** (*Class Diagrams*)
  - **Sekvensdiagrammer** (*Sequence Diagrams*)
  - **Tilstandsdiagrammer** (*State Charts*)
- Derforuden medtager UML-Light også:
  - Deploymentdiagrammer (*Deployment Diagrams*)
    - kan vise et systems hardwarekomponenter
  - Aktivitetsdiagrammer (*Activity Diagrams*)
    - anvendes til detaljeret design af en operation

Specifikation og implementering af et modul

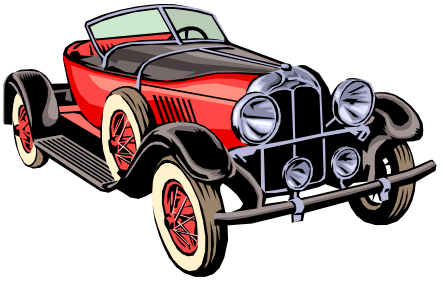


Figur 1

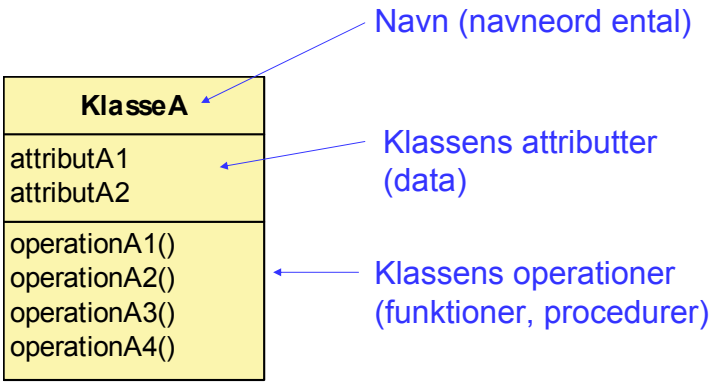
Indkapslingsprincippet

Information Hiding – David Parnas – 1972.

Eksempler på indkapsling



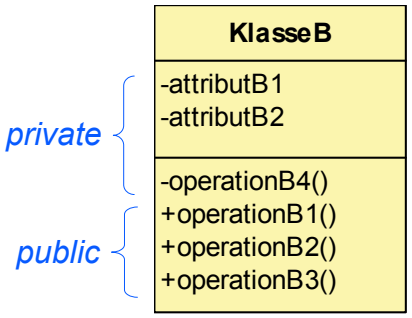
Notation for en klasse (1)



Figur 2

Notation for en klasse (2)

Med tilføjelse
af synlighed (*visibility*)

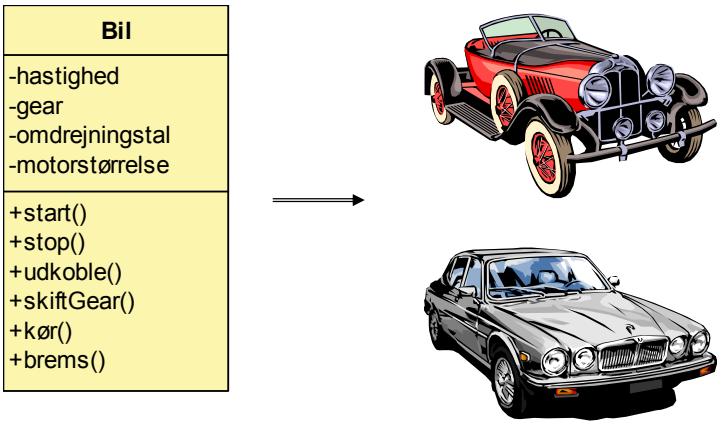


Figur 3

Et eksempel på en klasse

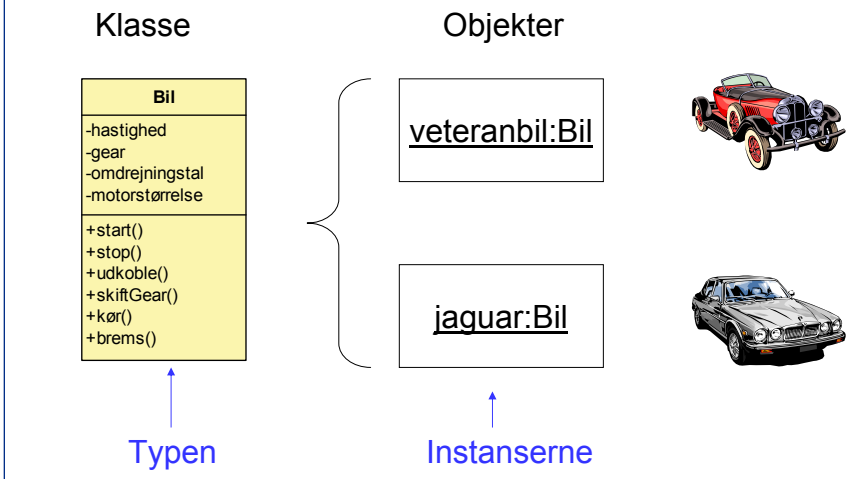
Klasse

To Bil objekter



Figur 4

UML Notation for objekter



Figur 5

Klasse og C++ kode

UML

Bil

-hastighed
-gear
-omdrejningstal
-motorstørrelse

+start()
+stop()
+udkoble()
+skiftGear()
+kør()
+brems()

Specifikation

Bil.h

```
class Bil
{
public:
    void start();
    void stop();
    void udkoble();
    void skiftGear();
    void koer();
    void brems();
private:
    long hastighed;
    unsigned char gear;
    float omdrejningstal;
    int motorstoerrelse;
}
```

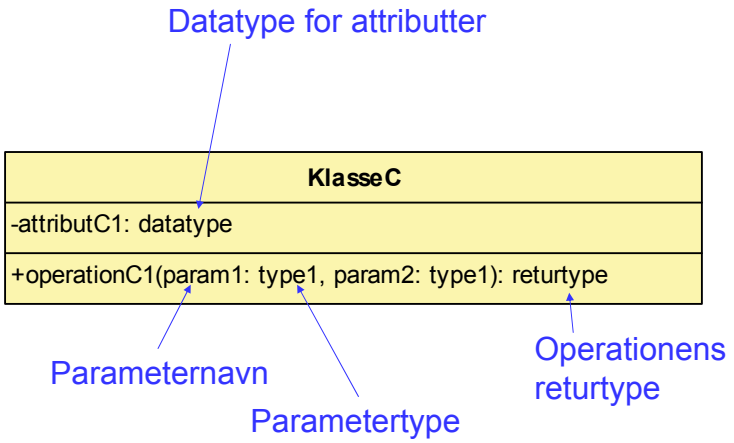
Bil.cpp

```
void Bil::start()
{
    // kode for start
    // operation
}

// etc.
```

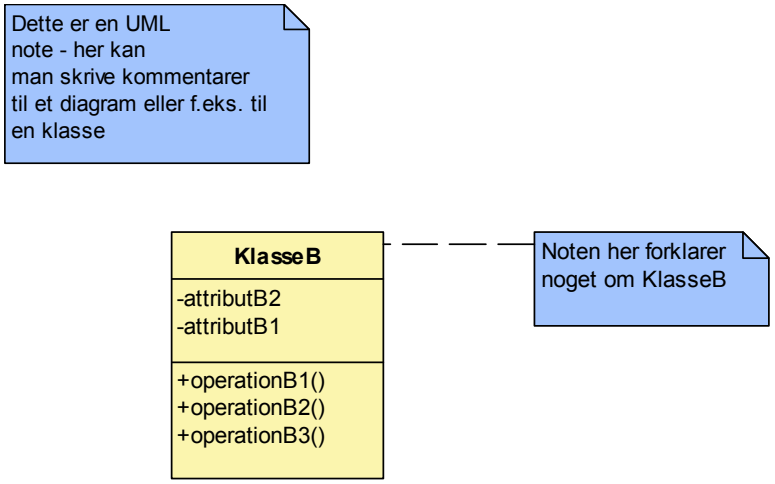
Implementering

Komplet specifikation af en klasse



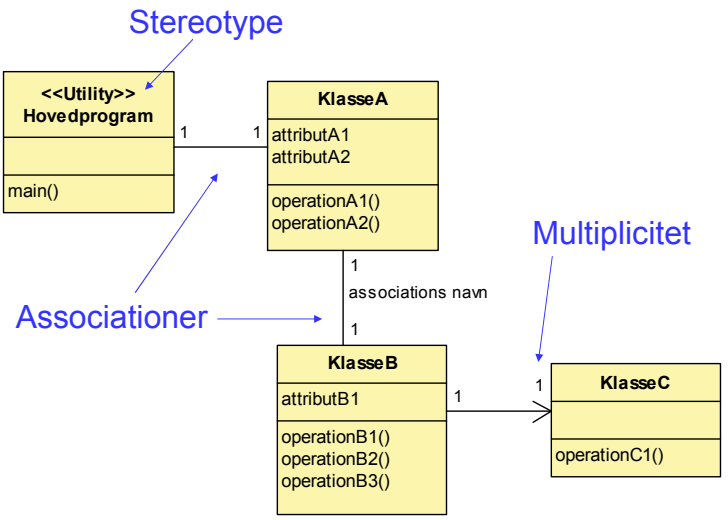
Figur 6

Notation for en note



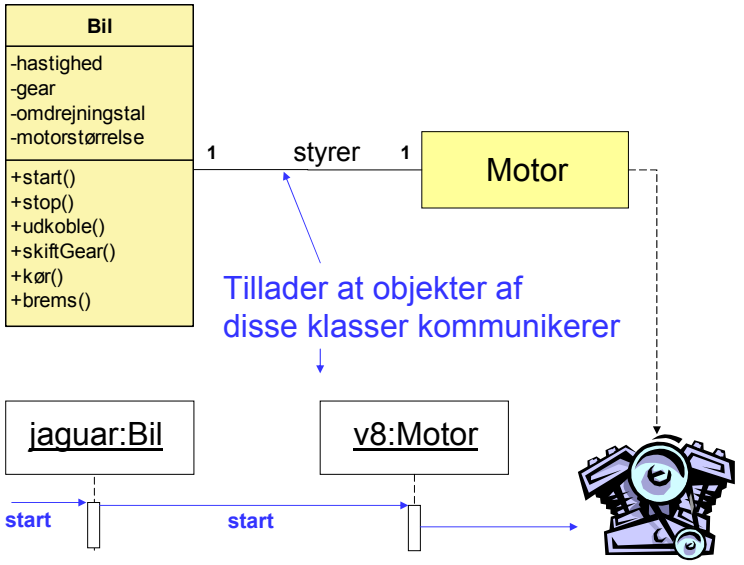
Figur 8,9

Notation for klassediagram

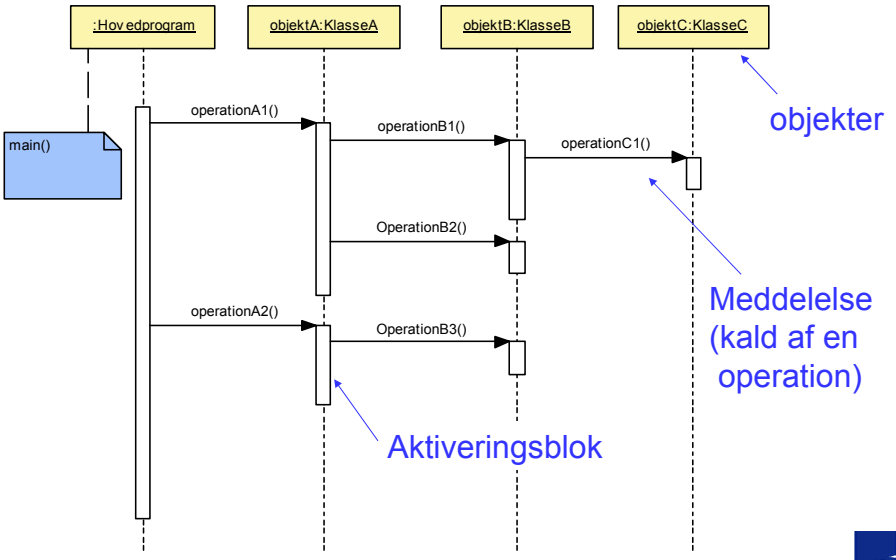


Figur 10

Associtationsbegrebet

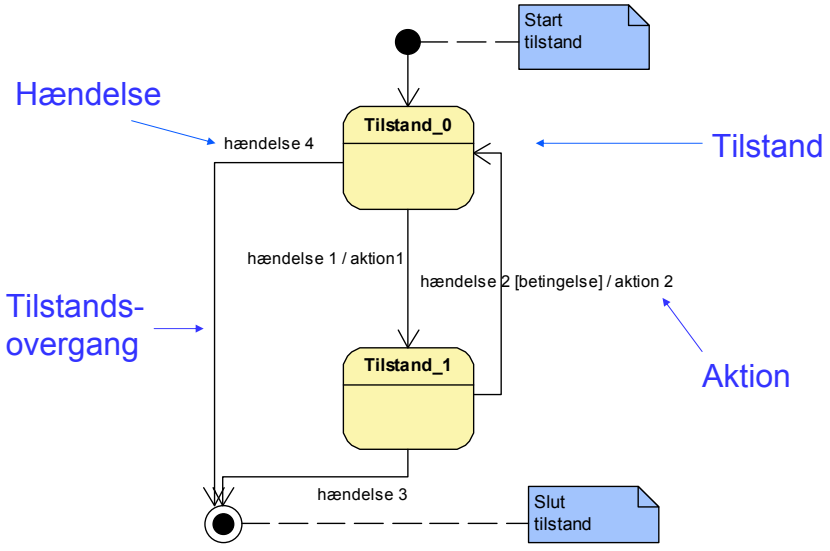


Notations for sekvensdiagram



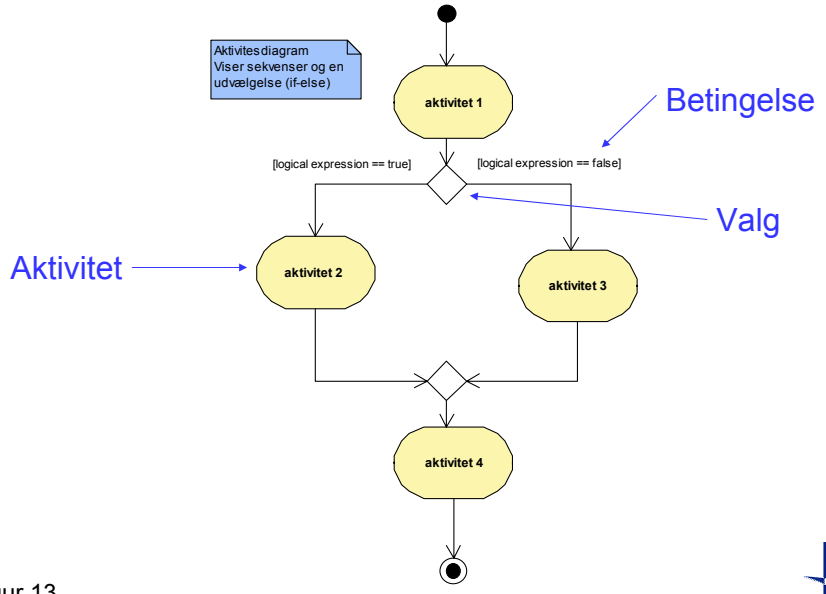
Figur 11

Notation for tilstandsdiagram



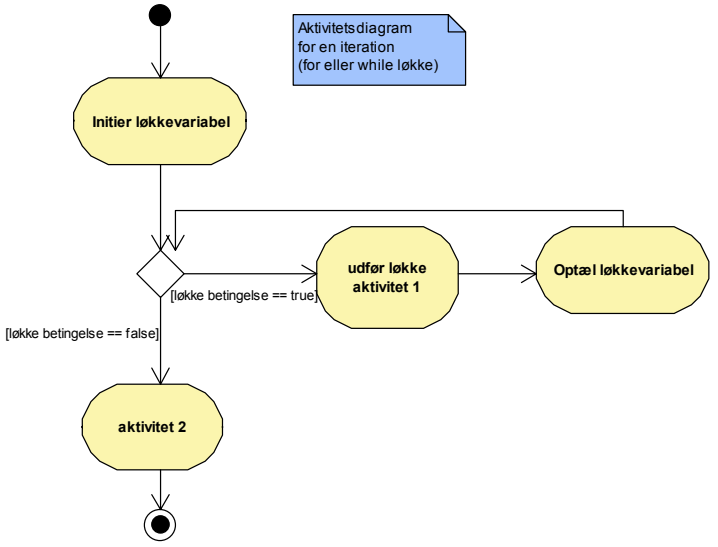
Figur 12

Notation for aktivitetsdiagram (1)



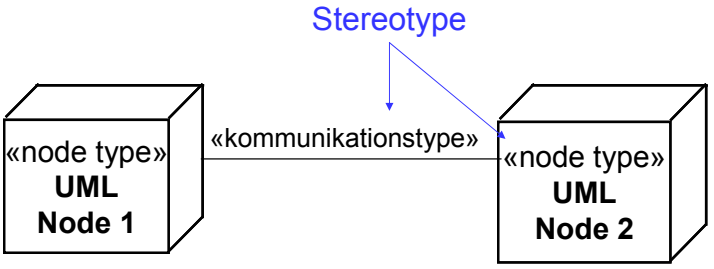
Figur 13

Notation for aktivitetsdiagram (2)



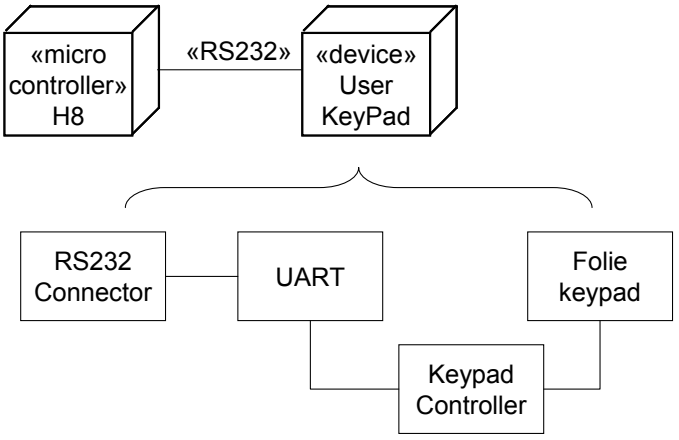
Figur 14

Notation for deploymentdiagram



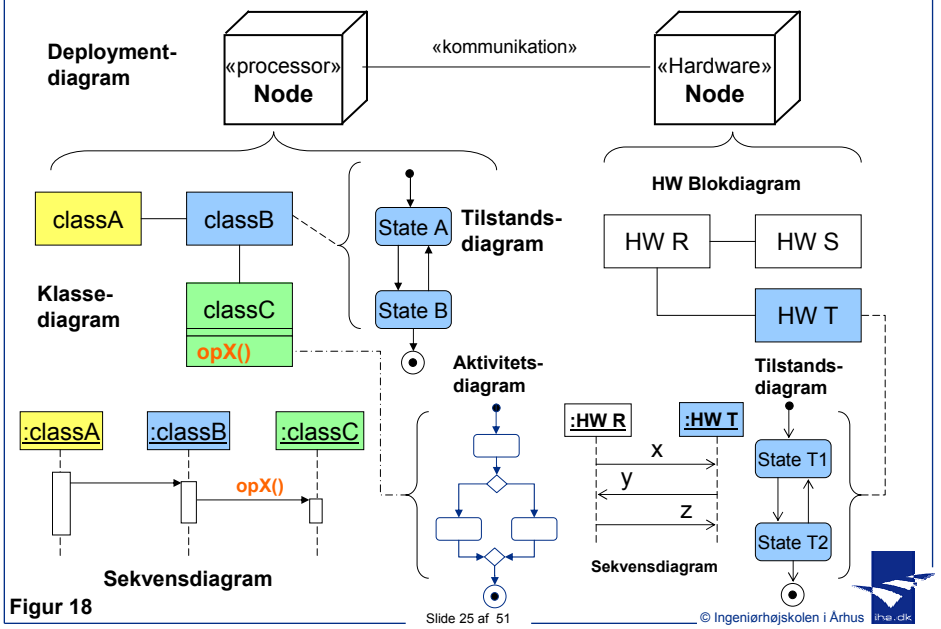
Figur 15

Deploymentdiagram eksempel



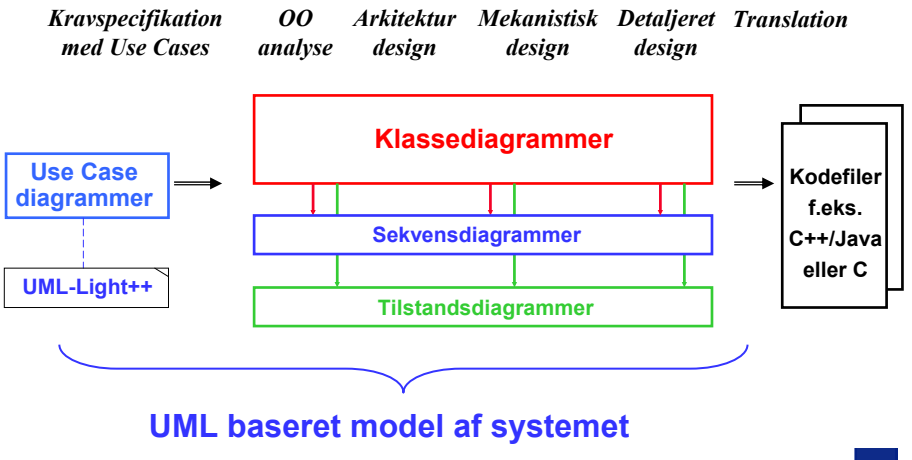
Figur 16

UML-Light oversigtsdiagram



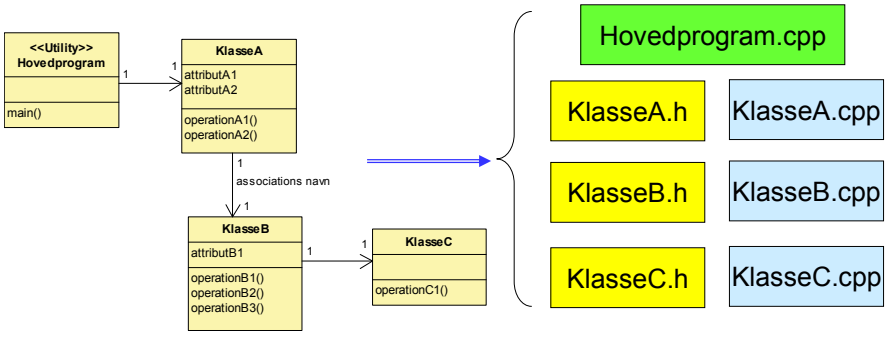
Figur 18

De fire primære UML diagramtyper



Anden del:
Sammenhæng mellem UML-Light og kode
+
Gennemgang af eksempler

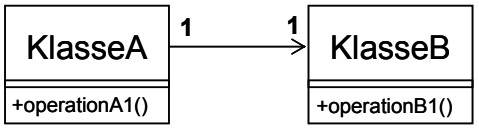
Sammenhæng mellem klassediagram og kode



Hovedprogram.cpp:	**#include "KlasseA.h"**
KlasseA.cpp:	#include "KlasseA.h"
	#include "KlasseB.h"
KlasseB.cpp:	#include "KlasseB.h"
	#include "KlasseC.h"
KlasseC.cpp:	#include "KlasseC.h"

Figur 19

Envejs 1-1 association




```
class KlasseA
{
public:
        KlasseA(KlasseB* pB);    // Constructor
        void operationA1();
private:
        KlasseB* pKlasseB; // implementerer associationen
}
```

Association implementeret vha. en pointer

Figur 20

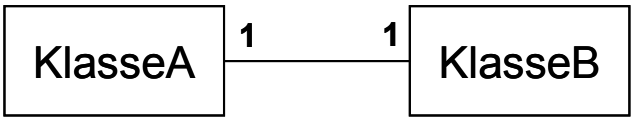
Implementering af envejs association

```
KlasseA::KlasseA(KlasseB* pB)      // constructor operationer
{
        pKlasseB= pB;
        // her indsættes den øvrige initaliseringskode
        // for klassens øvrige attributter
}
```

Som et eksempel på hvordan objekter af KlasseA og KlasseB oprettes og initialiseres, vises her et simpelt *main* program.

```
int main()
{
        KlasseB objektAfKlasseB;
        KlasseA objektAfKlasseA(&objektAfKlasseB);
        // nu har vi dannet forbindelsen således at én af
        // operationerne i KlasseA objektet kan kalde
        // operationB1() i det KlasseB objekt vi har oprettet
        objektAfKlasseA.operationA1();
        return (0);
}
```

Tovejs 1-1 association



KlasseA defineres som før.

```
class KlasseB
{
public:
        KlasseB();    // Constructor
        initAssociation(KlasseA* pA);
private:
        KlasseA* pKlasseA; // implementerer associationen
}
```

Figur 21

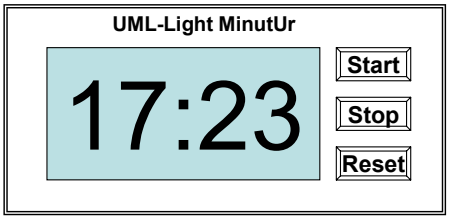
Implementering af tovejs association

KlasseA constructoren kommer til at se ud på følgende måde:

```
KlasseA::KlasseA(KlasseB* pB)
{
        pKlasseB= pB;
        pKlasseB->initAssociation(this);
}

int main()
{
        KlasseB objektAfKlasseB;          // NB! Skal oprettes først
        KlasseA objektAfKlasseA(&objektAfKlasseB);
        objektAfKlasseA.operationA1();
        objektAfKlasseB.operationB2();
        return (0);
}
```

Eksempel 1 – Et simpel minutur

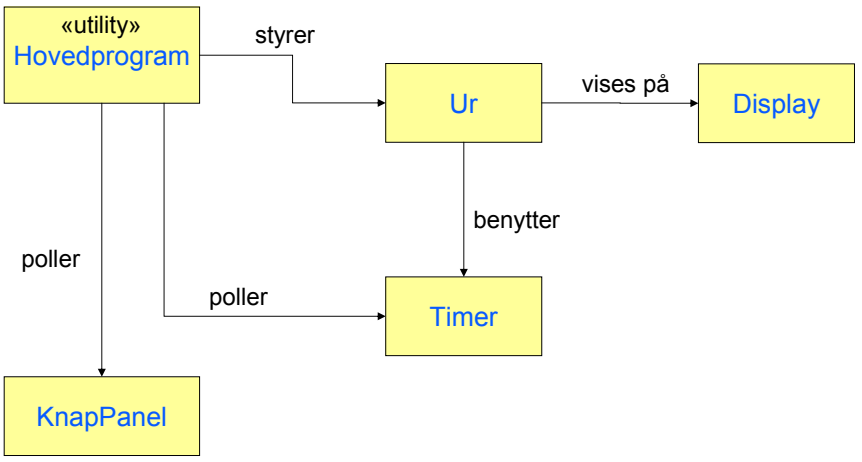


Minutur der viser minutter og sekunder.

Reset knappen nulstiller uret, hvis uret er stoppet.
Start knappen starter uret, der optæller tiden og viser denne i minutter og sekunder.
Stop knappen stopper uret.

Figur 22

Klassediagram for minutur



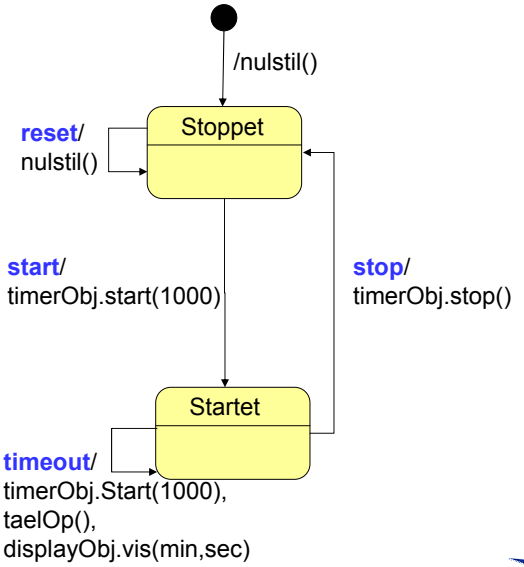
Tilstandsdiagram for klassen Ur

Hændelser:

- **reset**
- **start**
- **stop**

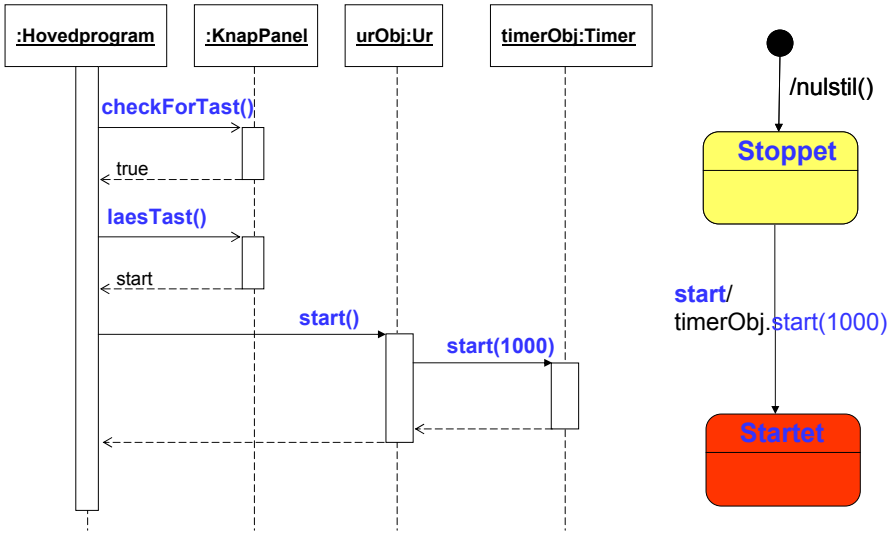
knap-tryk

timeout



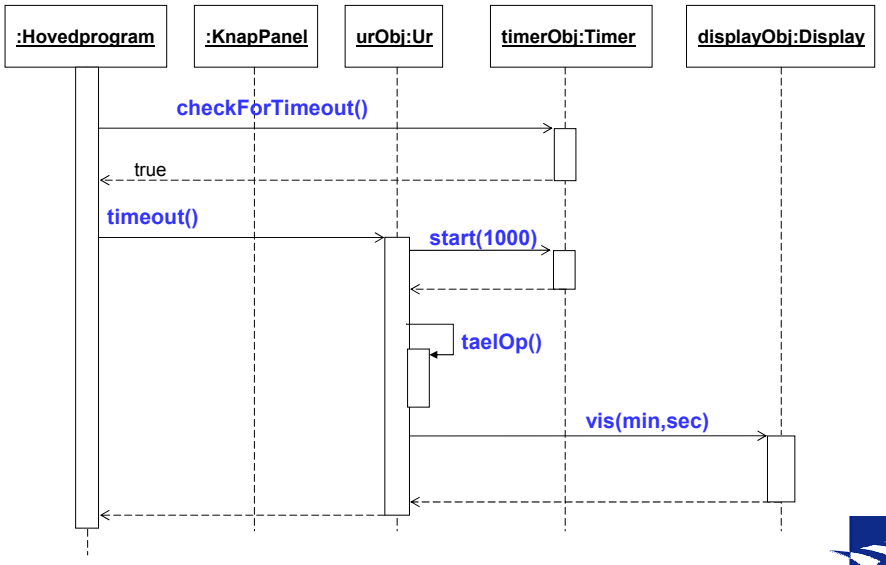
Figur 24

Sekvensdiagram – start scenario

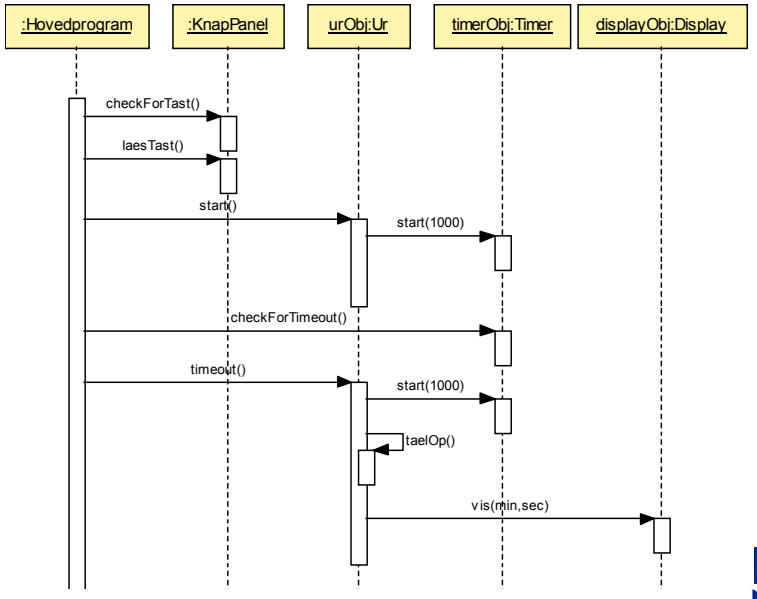


Figur 25

Sekvensdiagram – timeout scenario

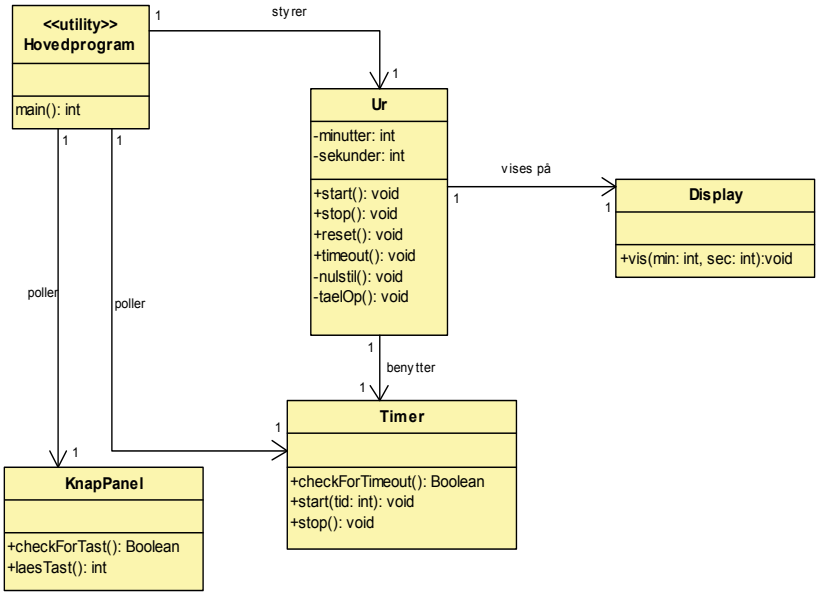


Sekvensdiagram for minutur



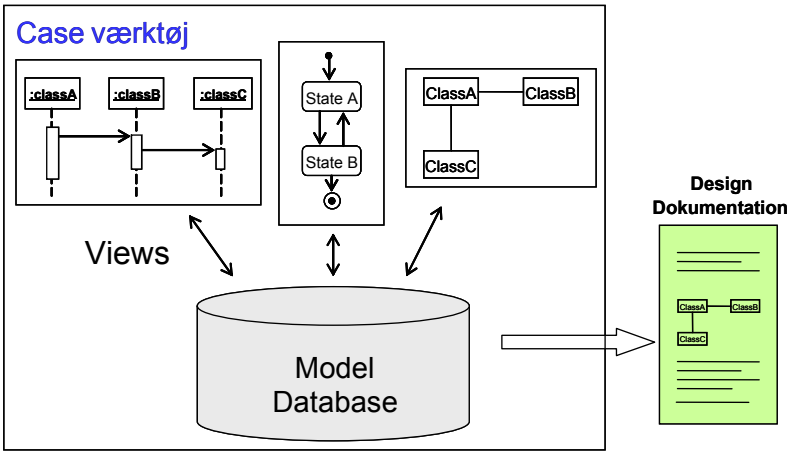
Figur 25

Klassediagram for minutur



Figur 23

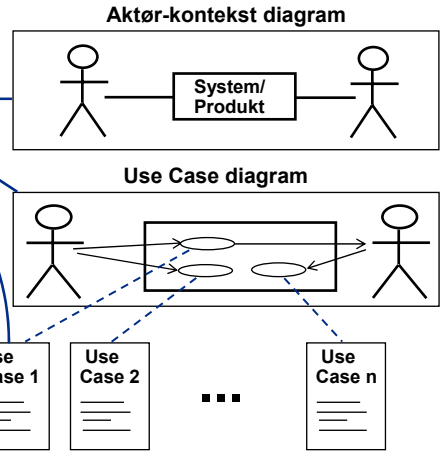
CASE værktøjer



Figur 26

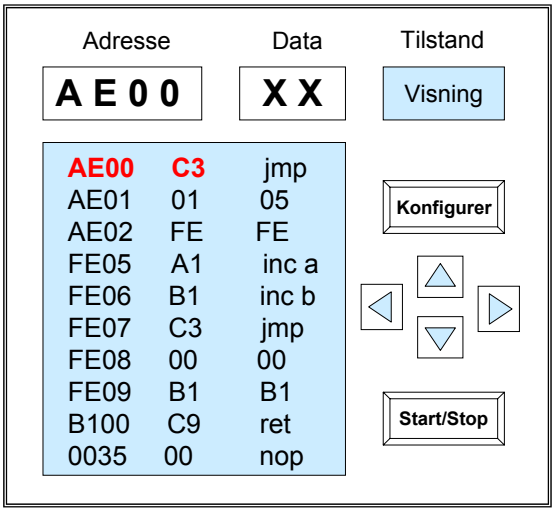
Kravspecifikation vha. Use Cases

1. Indledning
2. Generel beskrivelse
3. Funktionelle krav
4. Ekstern grænseflade
5. Krav til ydelse
6. Kvalitetsfaktorer
7. Design krav
8. Andre krav
9. Del-levering



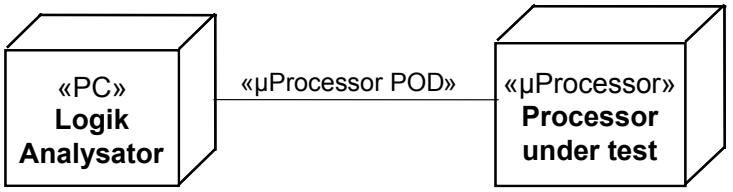
Figur 30

Eksempel 2. - Logikanalysator



Figur 37

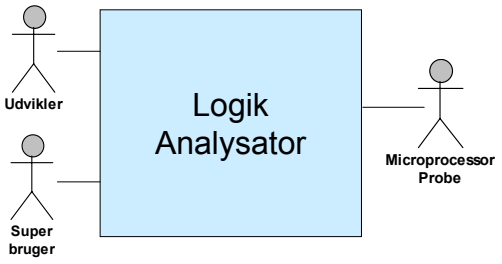
Deploymentdiagram for logikanalysator



Se klassediagram over softwarestrukturen på næste slide

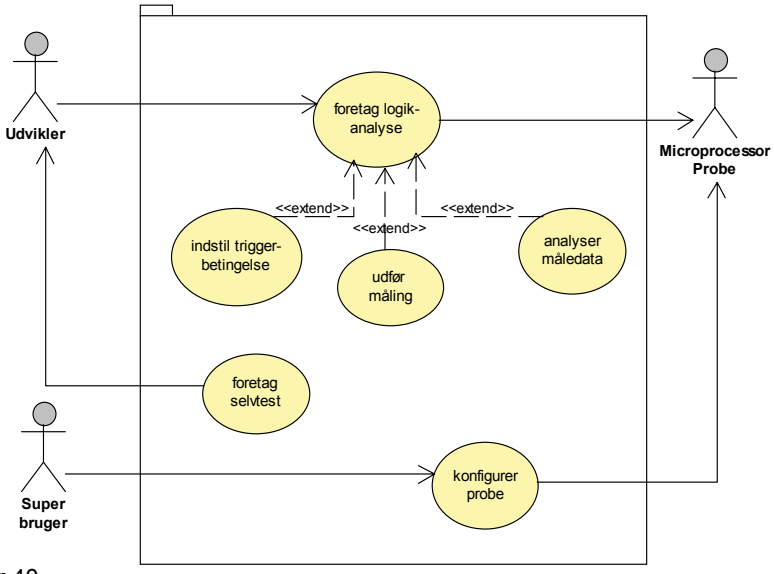
Figur 38

Aktør-kontekstdiagram



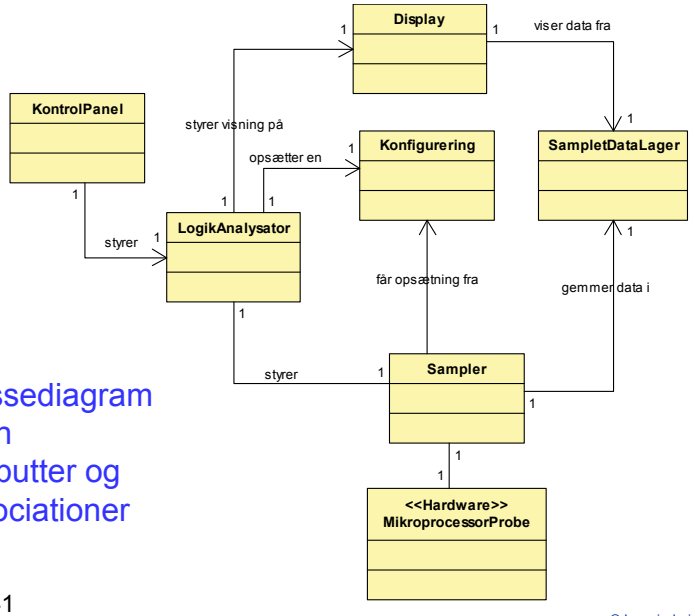
Figur 30

Use Case diagram for logikanalysator



Figur 40

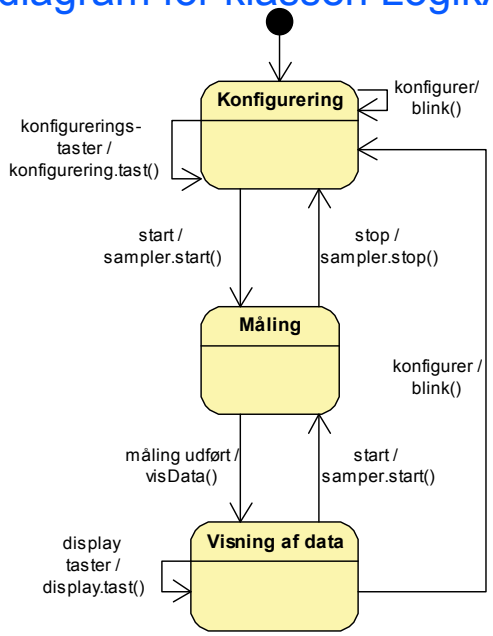
Klassediagram for logikanalysator SW



Klassediagram uden attributter og associationer

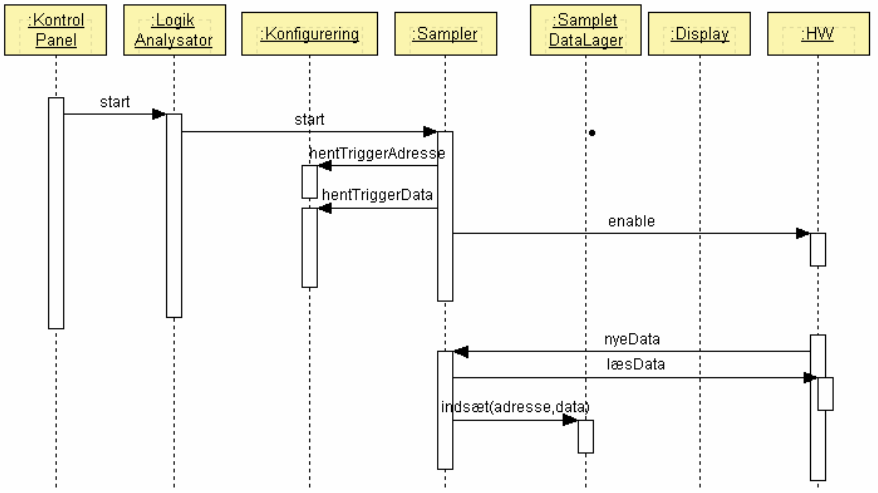
Figur 41

Tilstandsdiagram for klassen LogikAnalysator



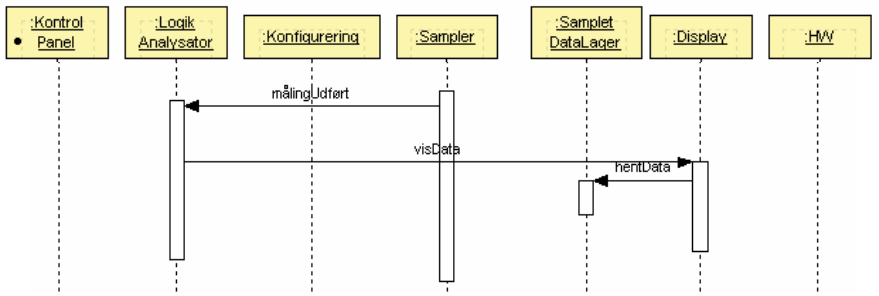
Figur 42

Sekvensdiagram – start scenario



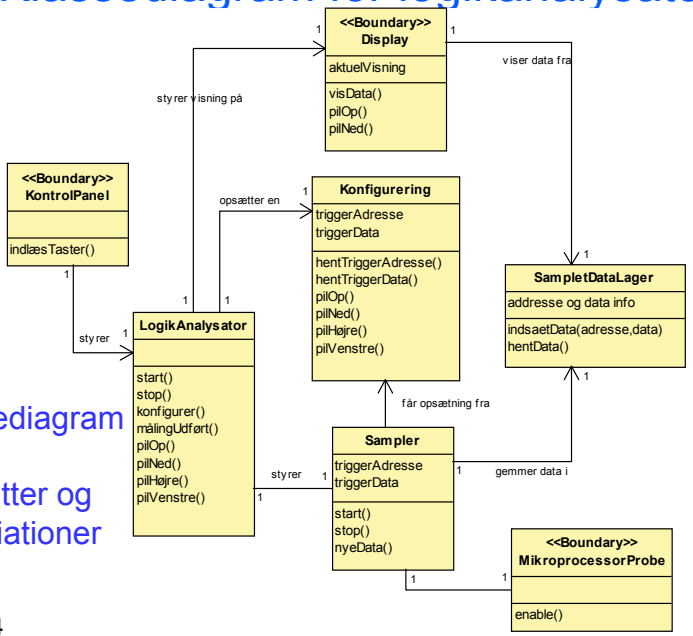
Figur 43

Sekvensdiagram – måling udført scenario



Figur 43

Klassediagram for logikanalysator



Klassediagram med attributter og associationer

Figur 44

Opsummering

- UML-Light er den delmængde af UML, der kan anvendes til Objektbaseret udvikling
- UML-Light kan anvendes til design/strukturering af såvel Software- som Hardwaredelen af et projekt
- UML-Light modellerne af softwaren kan implementeres i f.eks. C++, C eller assembler
- UML-Light tages i brug på 1. semester
- UML-Light++ anvendes på 2. semester ifm. objektorienteret udvikling i C++
- Den fulde UML introduceres på 3. semester sammen med en analyse- og designmetode