

EMS31 Kwartaal 4 Week 8: UML deel 2

Leerdoelen week 8 les 2. Je leert hoe je:

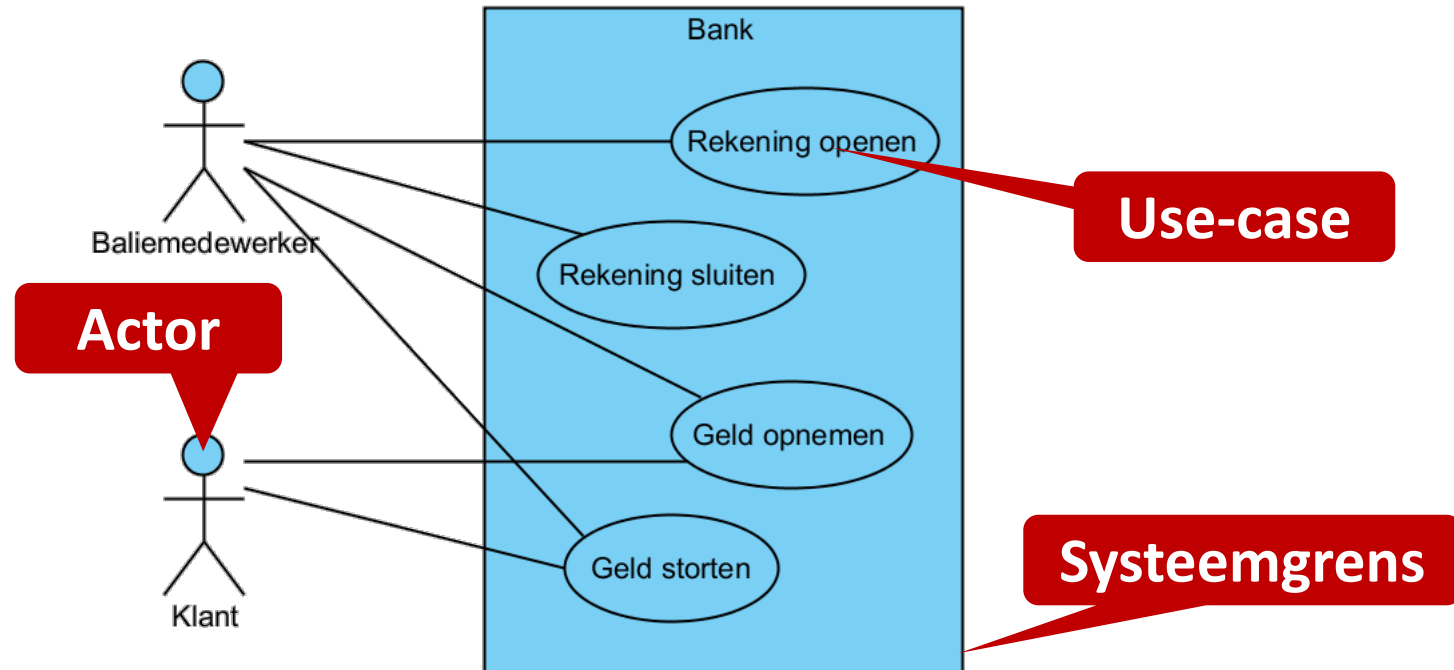
- UML kunt gebruiken om een model van de software te maken;
- Het dynamisch **gedrag** van een systeem te modelleren in UML.

- **Statische structuur** van programma.
 - UML Klassediagram.
 - UML Objectdiagram.
- **Dynamisch gedrag** van programma.
 - UML Use-case-diagram.
 - UML Sequentiediagram.
 - UML Communicatiediagram.
 - UML Toestanddiagram.
 - UML Activiteitsdiagram.

Het **Use-case-diagram** beschrijft het *gedrag* van het programma gezien vanuit de *gebruikers* van het programma.

UML Use-case-diagram

Wordt gebruikt voor vastleggen van de **functionele** eisen.



Use-case beschrijving

Naam	Rekening openen
Actor	Baliemedewerker
Aannamen	Baliemedewerker heeft beschikking over de NAW-gegevens van de Klant. De Klant kan zich legitimeren.
Beschrijving	<ol style="list-style-type: none">1. De baliemedewerker maakt aan het systeem bekend dat een nieuwe rekening aangemaakt moet worden en voert de NAW-gegevens van de Klant in.2. Als de Klant een bedrijf is wordt het KvK nummer ingevoerd.3. Het systeem controleert of de Klant al rekeningen heeft en of de Klant rood staat op een van deze rekeningen. In dat geval treedt een uitzondering [rood staan] op.4. Het systeem maakt het nummer van de nieuwe rekening bekend aan de baliemedewerker.
Uitzonderingen	[rood staan] De baliemedewerker kan naar de use-case Geld storten overgaan om de Klant de gelegenheid te geven het tekort aan te vullen. Als het tekort is aangevuld wordt de use-case vervolgd bij stap 3.
Resultaat	De Klant heeft minstens 1 rekening.

Statische structuur van programma.

- UML Klassediagram.
- UML Objectdiagram.

Dynamisch gedrag van programma.

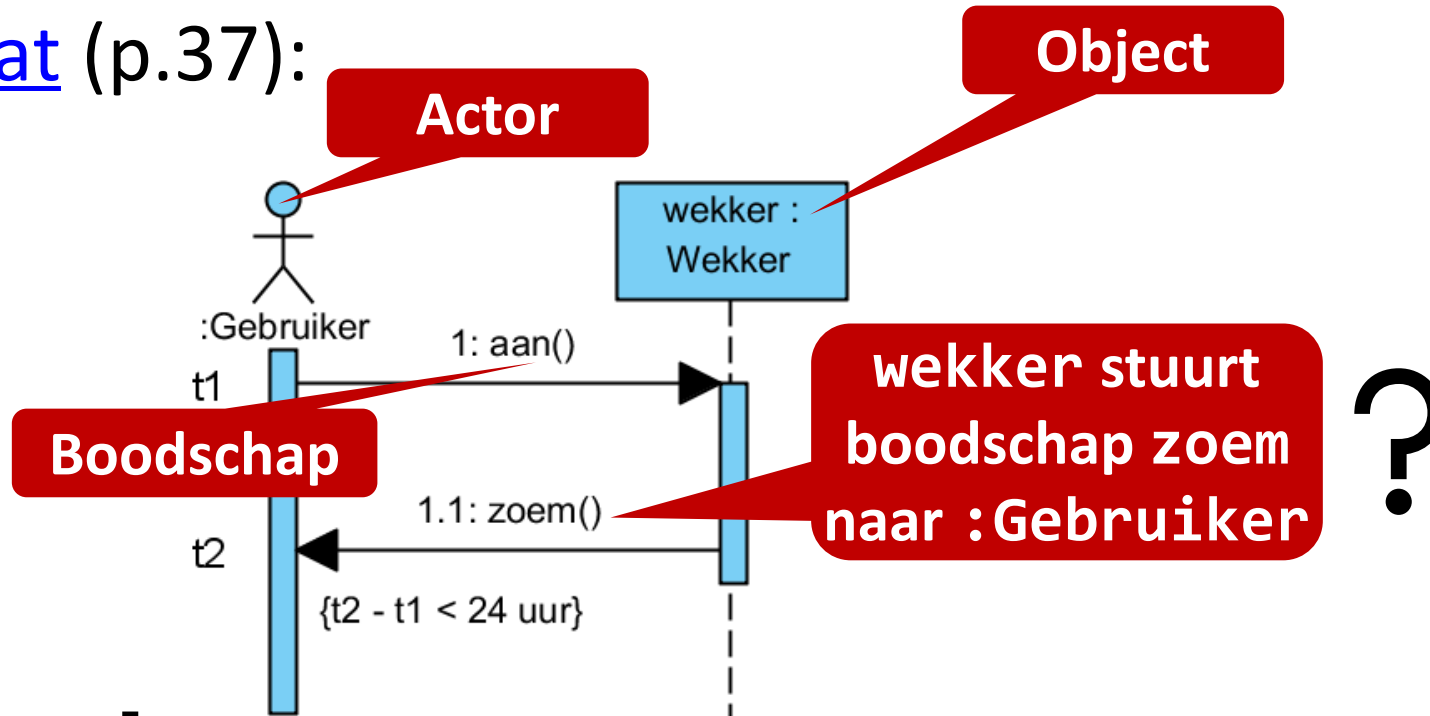
- UML Use-case-diagram.
- UML Sequentiediagram.
- UML Communicatiediagram.
- UML Toestanddiagram.
- UML Activiteitsdiagram.

Sequentiediagram laat zien in welke **volgorde** objecten elkaar berichten sturen.

Communicatiediagram laat zien welke objecten **elkaar** berichten sturen.

Sequentiediagram

Dictaat (p.37):

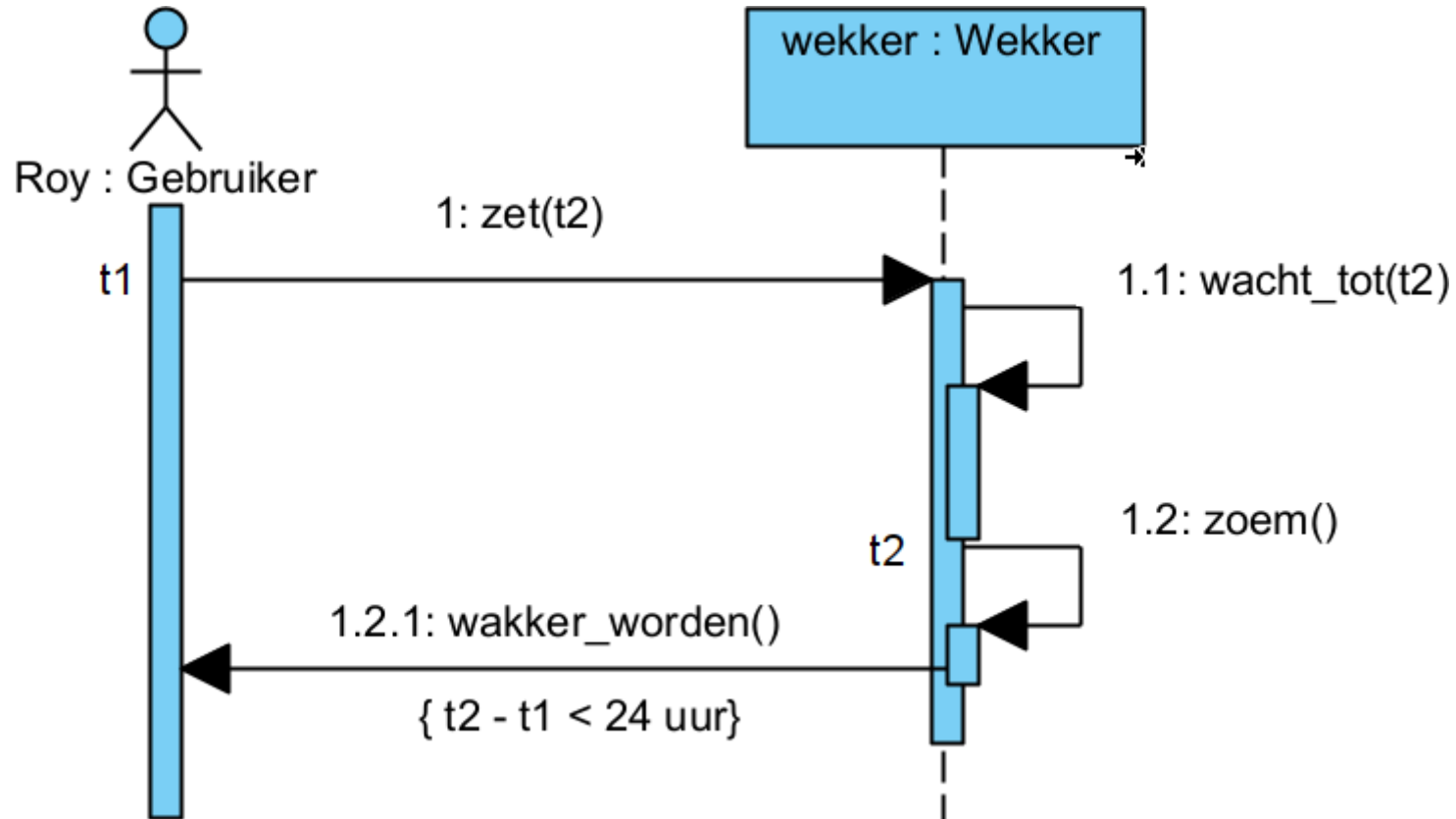


Tijd
↓

Actor kan **boodschap** sturen naar een **object**.
Object kan boodschap sturen naar een actor, andere objecten en naar zichzelf.

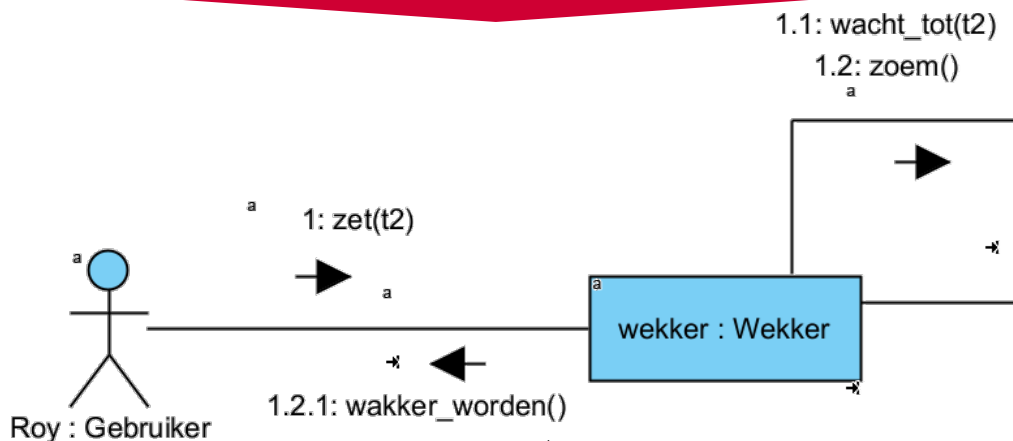
Sequentiediagram

Verbeterde versie:

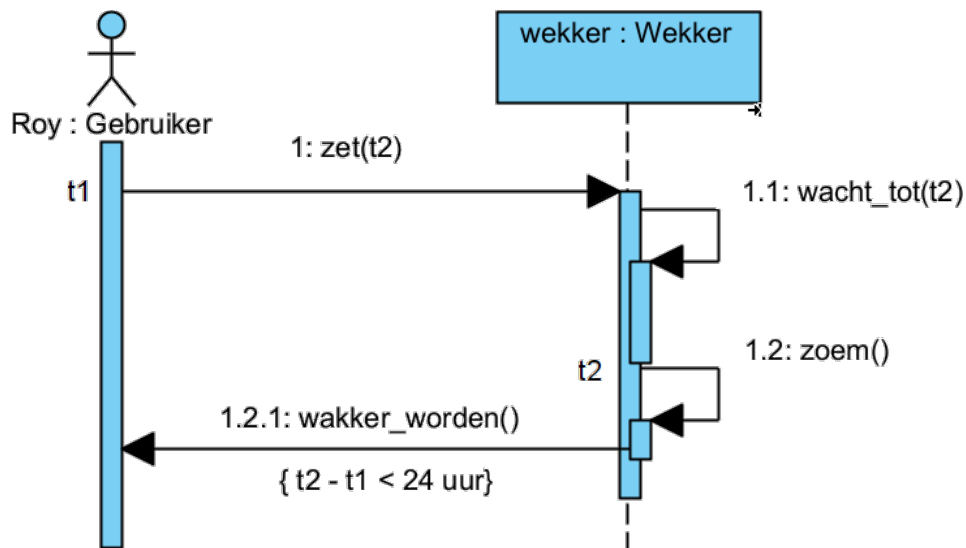
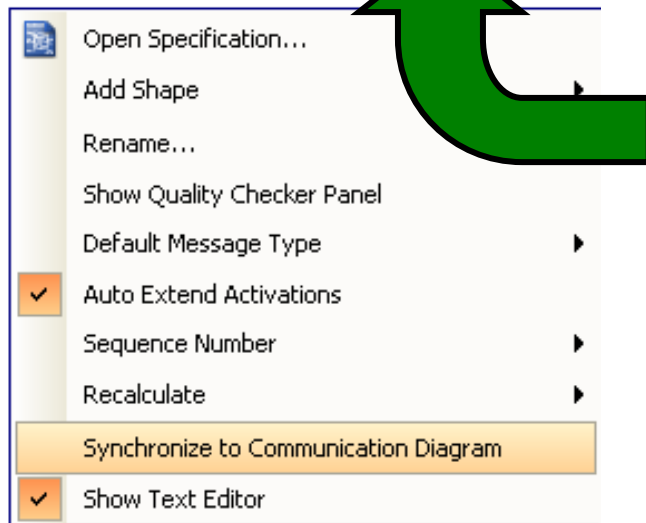


Communicatiediagram

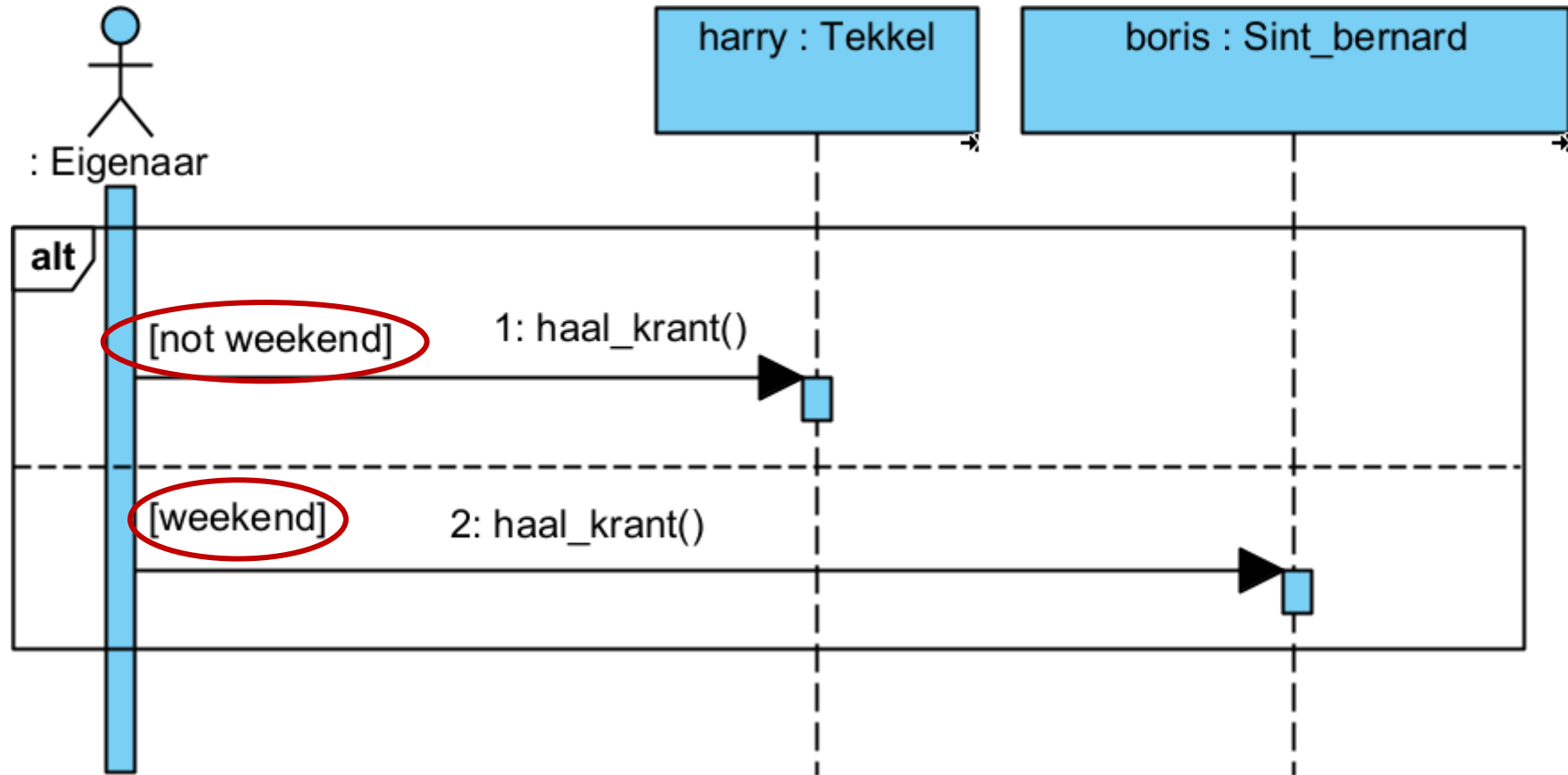
EMBEDDED SYSTEMS



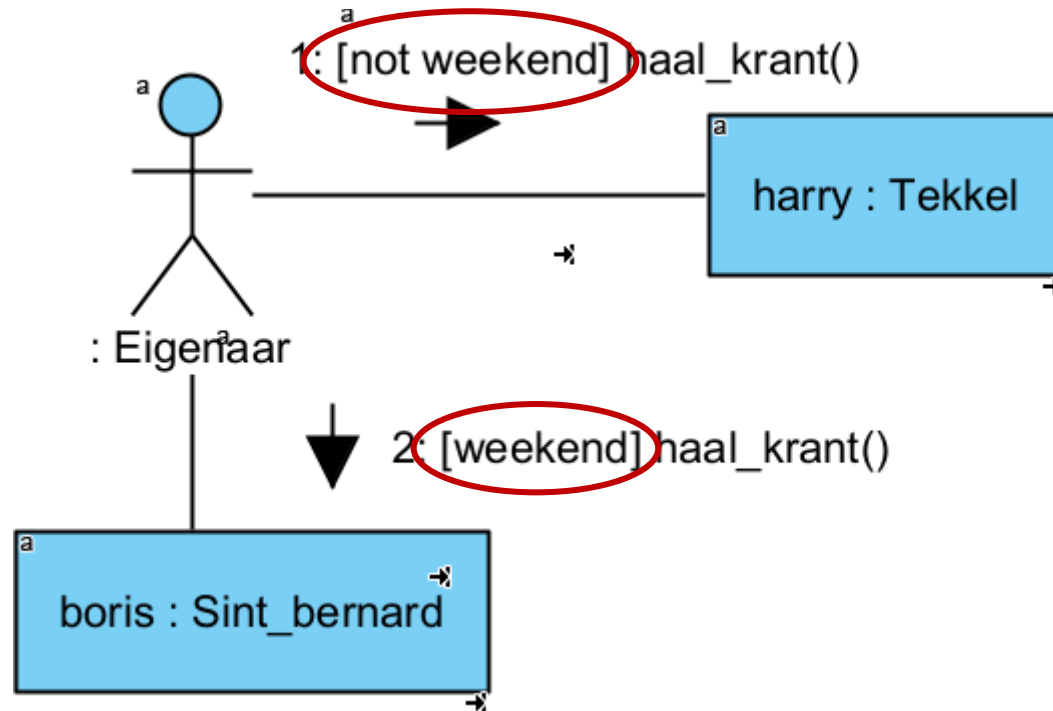
Bevat **dezelfde informatie** als een sequentiediagram.



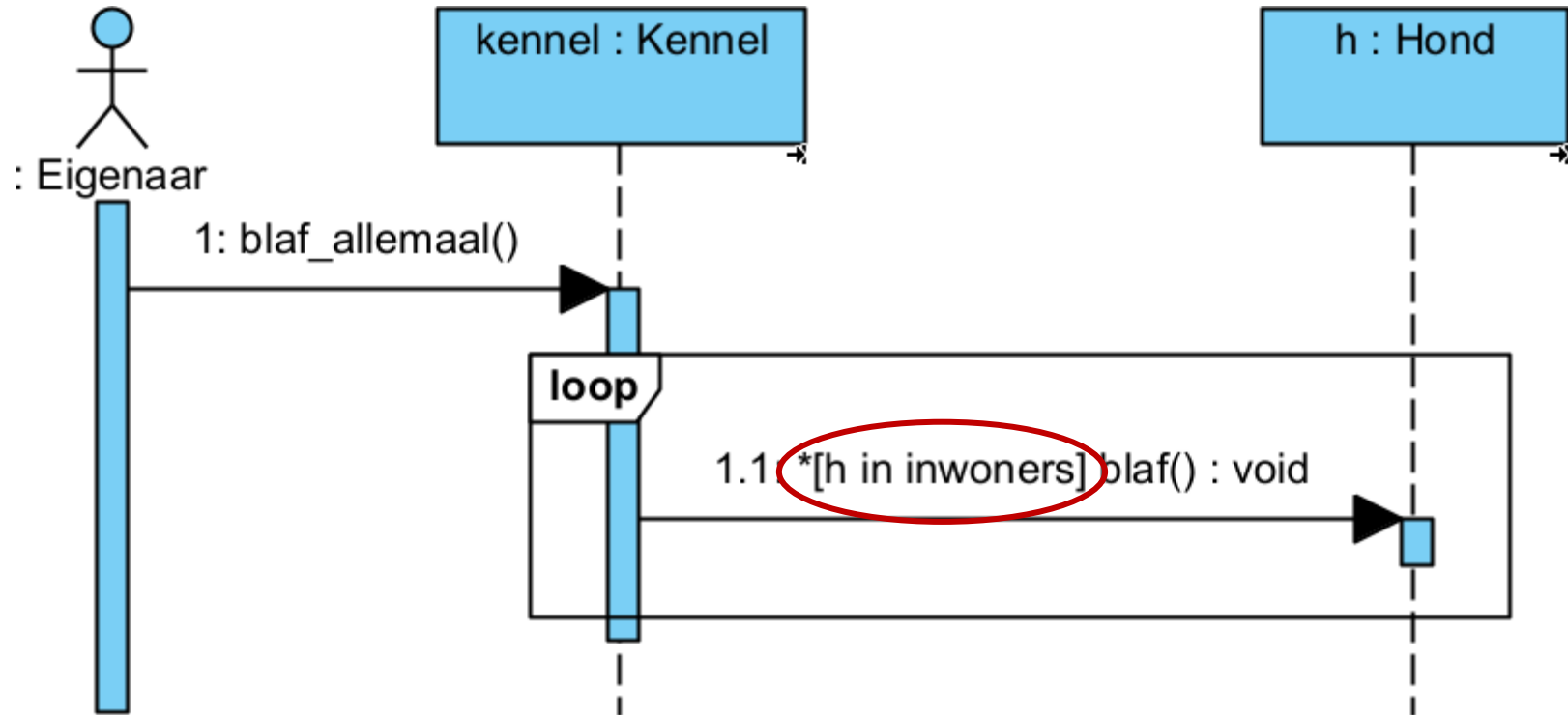
Conditionele boodschappen



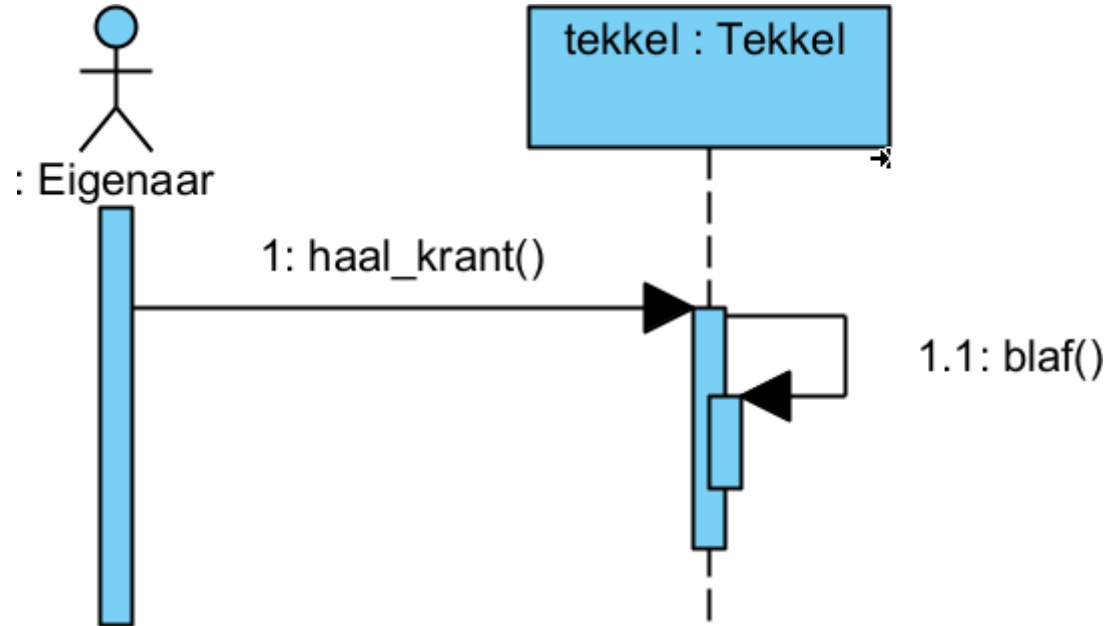
Conditionele boodschappen



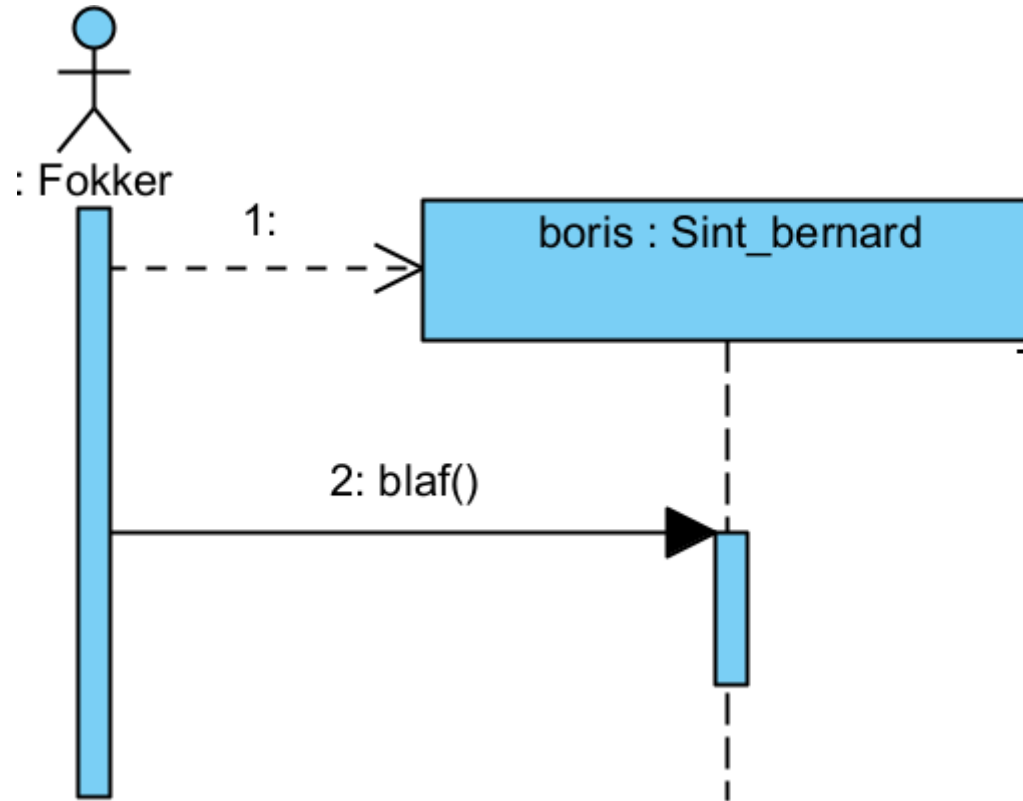
Iteratie van boodschappen



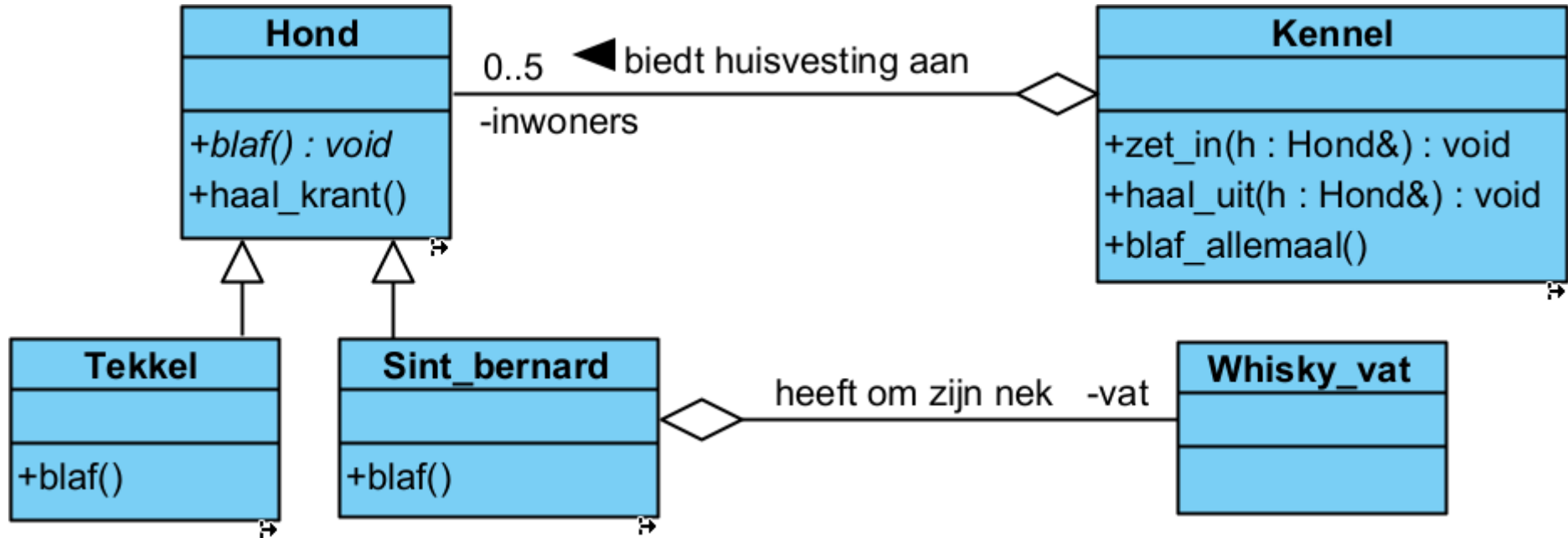
Boodschap aan jezelf



Constructor



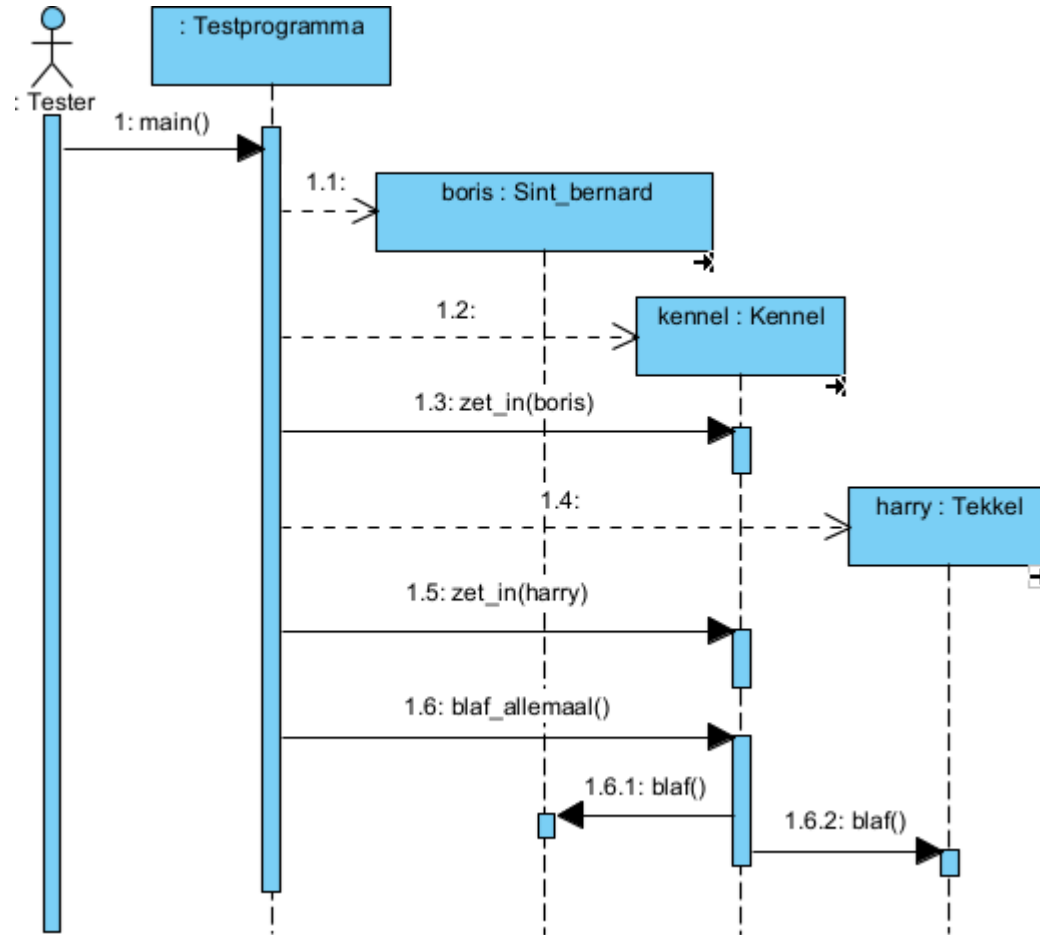
Voorbeeld (vervolg...)



Maak een sequentiediagram waarin een testprogramma de volgende acties uitvoert:

- Zet een `Sint_bernard` genaamd `boris` in de `Kennel` genaamd `k`.
- Zet een `Tekkel` genaamd `harry` in de `Kennel` `k`.
- Laat alle honden in de `Kennel` `k` blaffen.

Uitwerking



Statische structuur van programma.

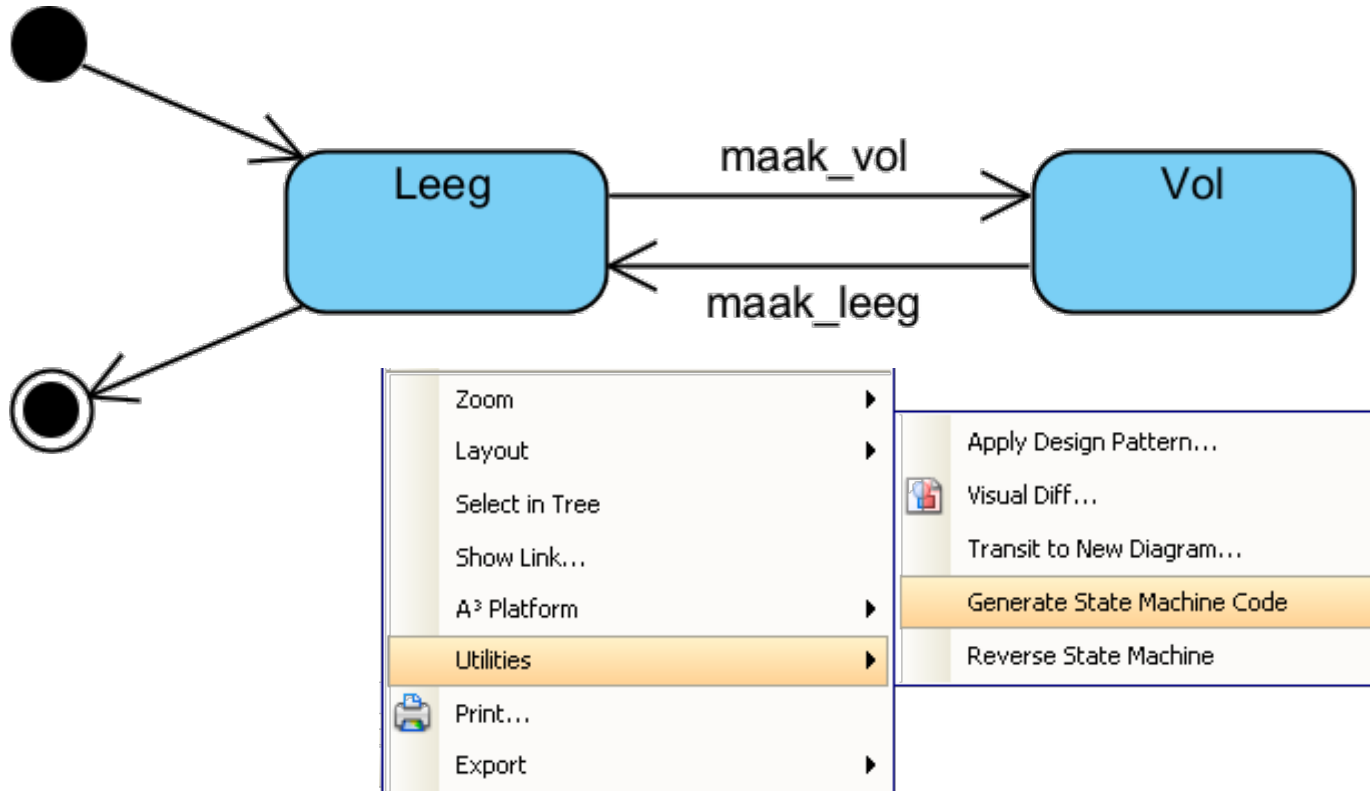
- UML Klassediagram.

Dynamisch gedrag van programma.

- UML Use-case-diagram.
- UML Sequentiediagram.
- UML Communicatiediagram.
- UML Toestandsdiagram.
- UML Activiteitsdiagram.

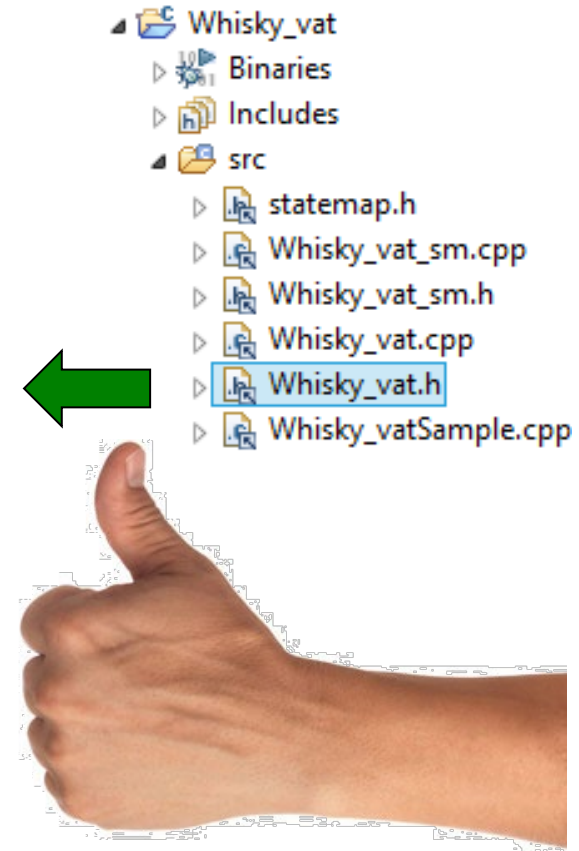
Toestandsdiagram laat de **toestanden** en **toestandsovergangen** van een klasse zien.

Toestand van een WhiskeyVat



Toestandsdiagram

```
#include "Whisky_vat_sm.h"
class Whisky_vat {
private:
    Whisky_vatContext _fsm;
public:
    Whisky_vat() : _fsm(*this) {
    }
    Whisky_vatContext& getContext() {
        return _fsm;
    }
    void maak_vol() {
        _fsm.maak_vol();
    }
    void maak_leeg() {
        _fsm.maak_leeg();
    }
};
```

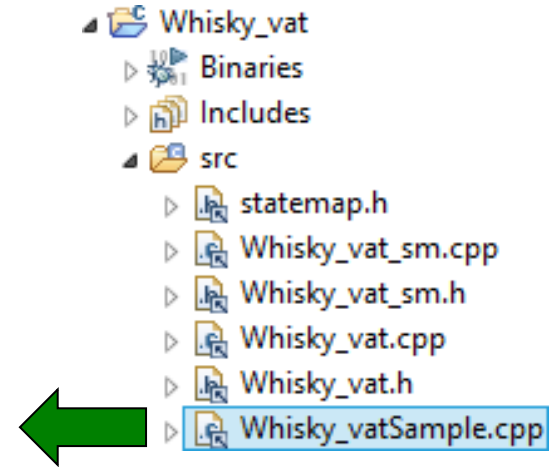


Toestandsdiagram

```
#include "Whisky_vat.h"
```

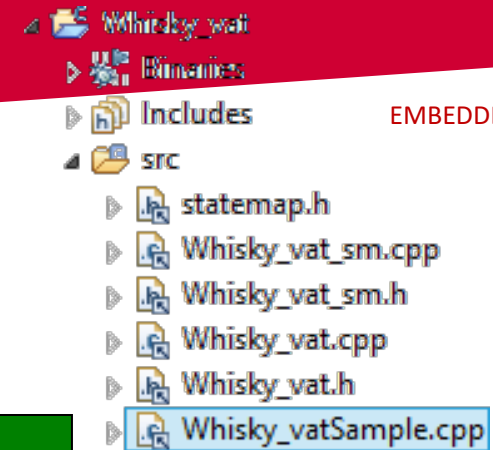
```
void state_Leeg(Whisky_vat *aWhisky_vat) {  
    printf("Please select transition:\n");  
    printf("1. maak_vol\n");  
    printf("0. quit\n");  
    int choice;  
    scanf("%d", &choice);  
    switch (choice) {  
        case 1:  
            aWhisky_vat->maak_vol();  
            break;  
        case 0:  
            exit(0);  
    }  
}
```

```
void state_Vol(Whisky_vat *aWhisky_vat) {
```



Toestandsdiagram

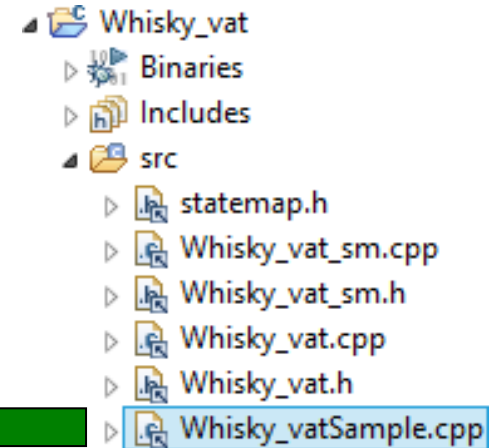
```
int main(int argc, char **argv) {
    Whisky_vat lWhisky_vat;
    while (true) {
        printf("Current state: %s\n",
            lWhisky_vat.getContext().getState().getName());
        if (&lWhisky_vat.getContext().getState() ==
            &Whisky_vatFSM::Leeg) {
            state_Leeg(&lWhisky_vat);
        }
        else if (&lWhisky_vat.getContext().getState() ==
            &Whisky_vatFSM::Vol) {
            state_Vol(&lWhisky_vat);
        }
    }
}
```



EMBEDDED SYSTEMS

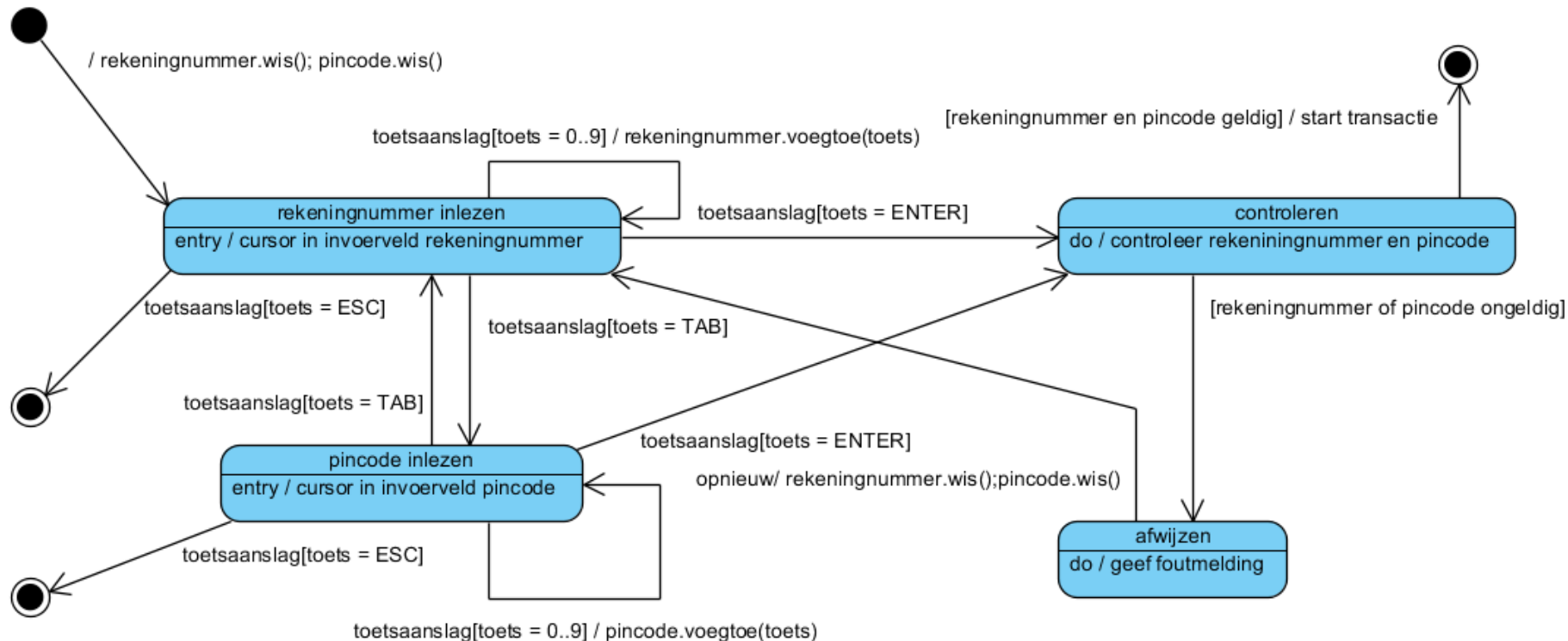
Toestandsdiagram

```
$ ./whisky_vat.exe
Current state: whisky_vatFSM::Leeg
Please select transition:
1. maak_vol
0. quit
1
Current state: whisky_vatFSM::Vol
Please select transition:
1. maak_leeg
0. quit
1
Current state: whisky_vatFSM::Leeg
Please select transition:
1. maak_vol
0. quit
0
```



Teken een toestandsdiagram dat het **login** proces van een online bankapplicatie modelleert.

- Om in te loggen moet de klant een **rekeningnummer** en een **pincode** invoeren in twee invoervelden.
- In deze invoervelden mogen alleen **cijfers** worden ingevoerd.
- Wisselen tussen invoervelden kan met de **TAB** toets.
- Als de **ENTER** toets wordt ingedrukt dan moeten het ingevoerde **rekeningnummer** en **pincode** worden gecontroleerd.
 - Geldig: start transactie.
 - Ongeldig: geef foutmelding en wis invoervelden.
- Als op de **ESC** toets wordt gedrukt moet de login procedure worden afgebroken.



Statische structuur van programma.

- UML Klassediagram.

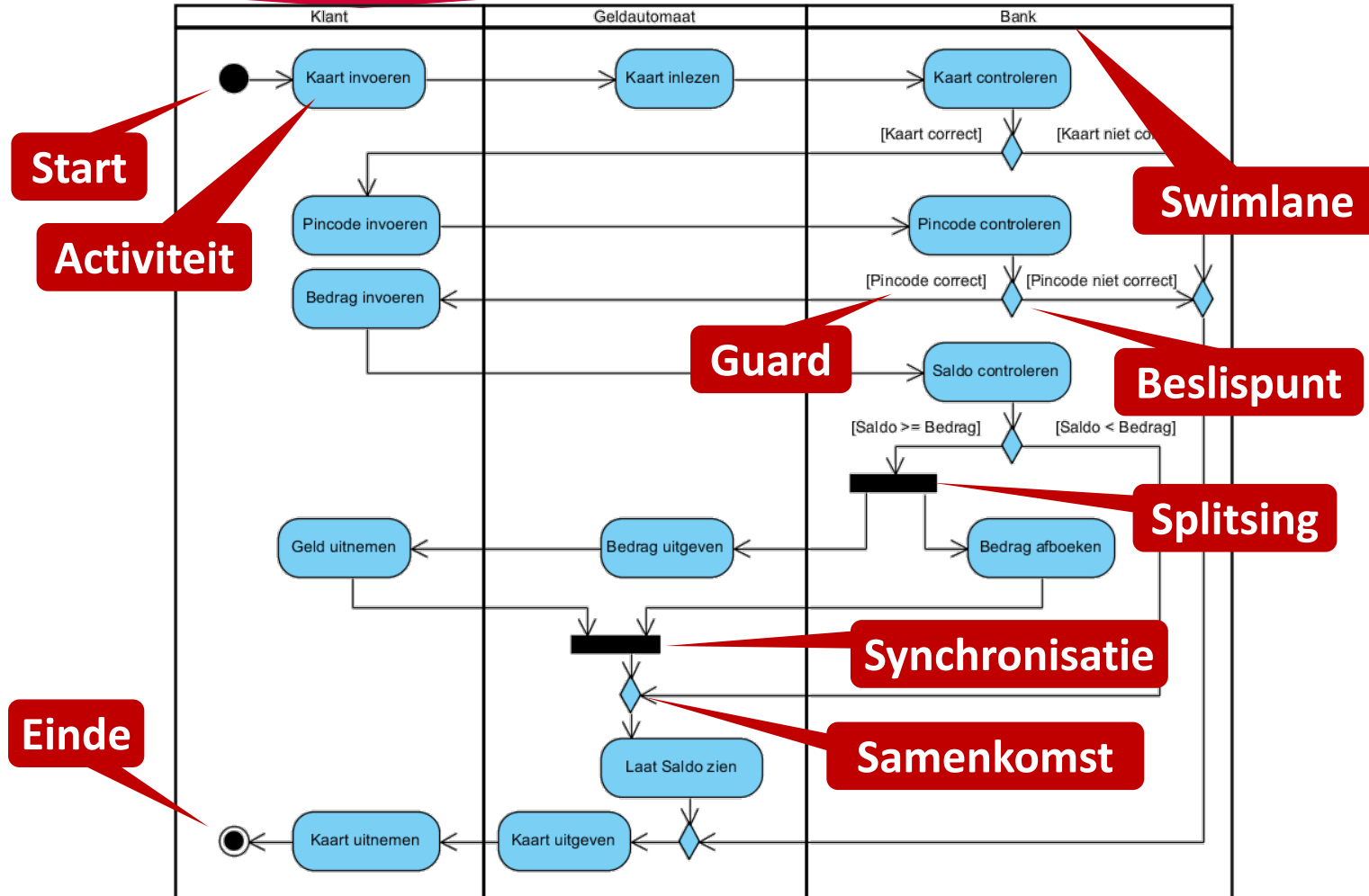
Dynamisch gedrag van programma.

- UML Use-case-diagram.
- UML Sequentiediagram.
- UML Communicatiediagram.
- UML Toestanddiagram.
- UML **Activiteitsdiagram**.

Een **Activiteitsdiagram** laat een **stroom** van activiteiten zien.

Activiteitsdiagram voorbeeld

EMBEDDED SYSTEMS



Aan de slag!

EMBEDDED SYSTEMS

Verder met [Eindopdracht_2.pdf](#)

