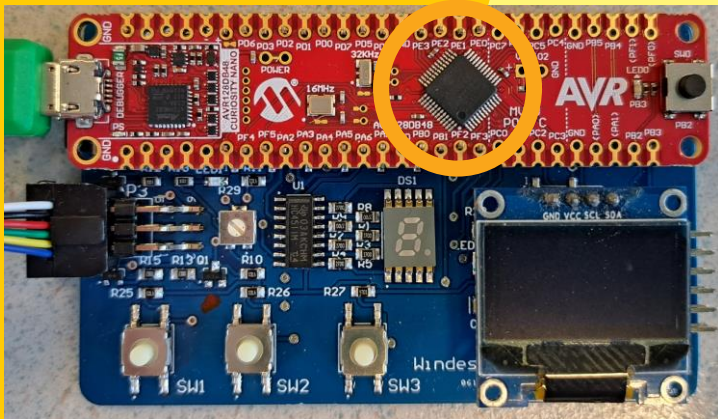


Microcontroller Programmeren

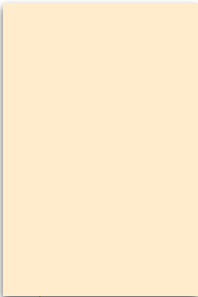
Lesweek 2

Docenten:



Wie is wie?

Lab-beheerders




Henk Bouwman



Marcello Roosenstein

Docenten



Douwe Schotanus



Marco Winkelman



Richard Rosing

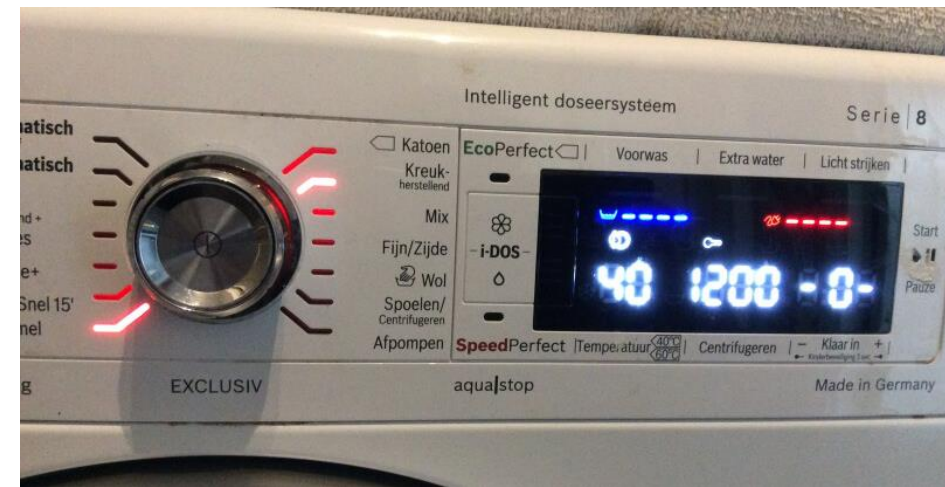


Wat gaan we doen?

Week 1:	Inleiding, Microcontroller Architectuur, GPIO, debuggen
Week2:	Programma flow, toestanden, state machine
Week3:	Interrupts
Week4:	Timers
Week5:	PWM
Week6:	ADC



Besturingen



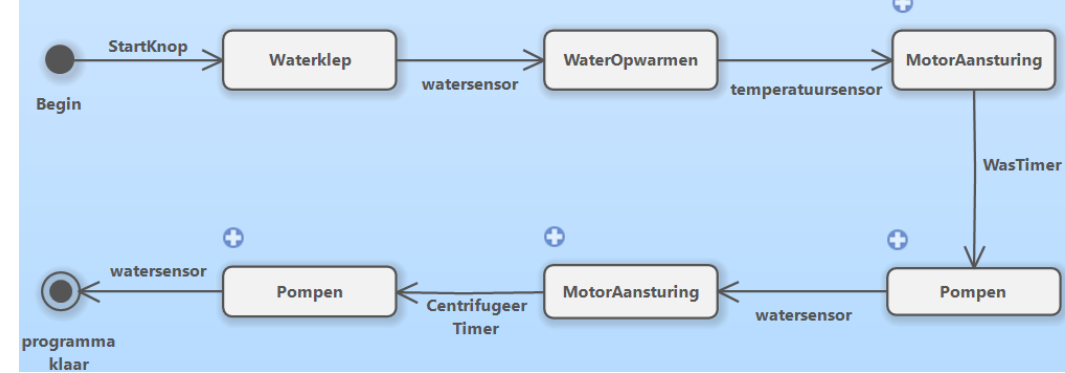
Veel apparaten om je heen bevatten een besturing op basis van een microcontroller, dus in software:

- huishoudelijke apparatuur zoals oplaadbare stofzuiger, espressomachine, wasmachine
- consumenten elektronica zoals een TV, mediaplayer, smartwatch
- medische apparatuur, zoals bloeddrukmeter, beademingsapparaat
- systemen in de auto, zoals de ABS, motormanagement en navigatie
- verkeersregelingen , industriële apparatuur (lasrobot, logistiek), bagage-afhandeling

De laatste worden meestal bestuurd door een PLC of SCADA systeem



Besturingen



Een besturing zorgt dat een apparaat zich **gedraagt** volgens een bepaald **plan**:

- **Aflopen** van een bepaald **programma**, zoals bij een wasmachine, of espressoapparaat, of volgen van een route zoals een navigatie systeem
 - Het **regelen** van een bepaalde **grootte**, zoals temperatuur
 - Het **gedrag afhankelijk** maken van de **situatie (toestand)**, bv een User Interface
 - Of een **combinatie** van alles
-
- Het **gewenste gedrag** wordt bepaald uit de **eisen** aan het systeem
 - Wordt vastgelegd in **ontwerp** van het product, en bv de gebruiksaanwijzing



State Machines

Het is slim om de ervaring te gebruiken van ontwerpen van de besturing van “echte” apparaten:

We willen voorkomen dat we spaghetti code krijgen

- Overzicht in structuur door het opdelen in subsystemen
- Overzicht in gedrag door een “event-driven” systeem te ontwerpen

Deze les kijken we naar **State Machines** als besturing van een apparaat,

Dat is een goede manier om een systeem event-driven te maken

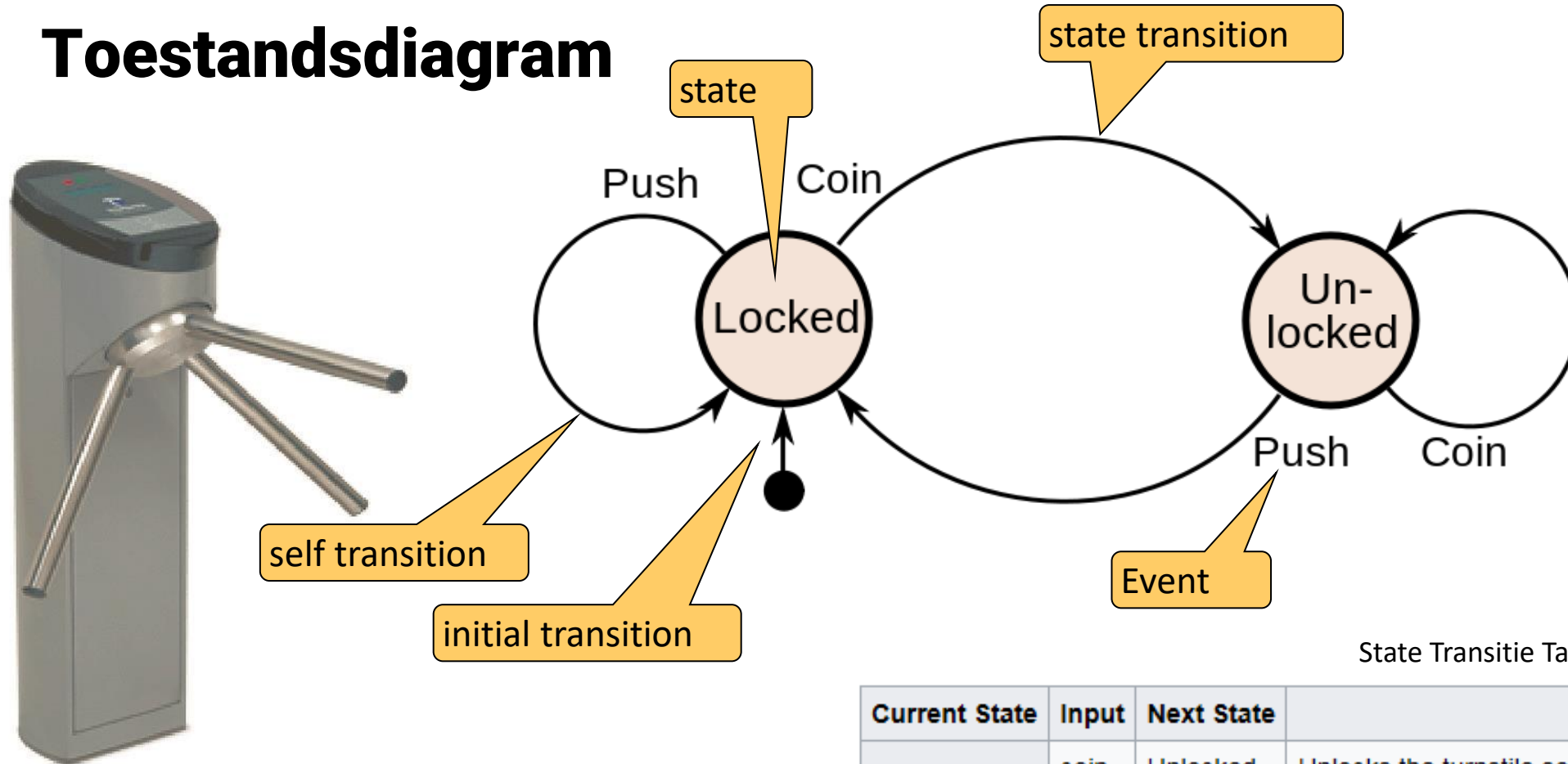
Het bepaalt het **gedrag**

- UML State Diagram (NL: toestandsdiagram model)
- Coderen van een state machine in C



State Diagram

Toestandendiagram



State Transitië Tabel

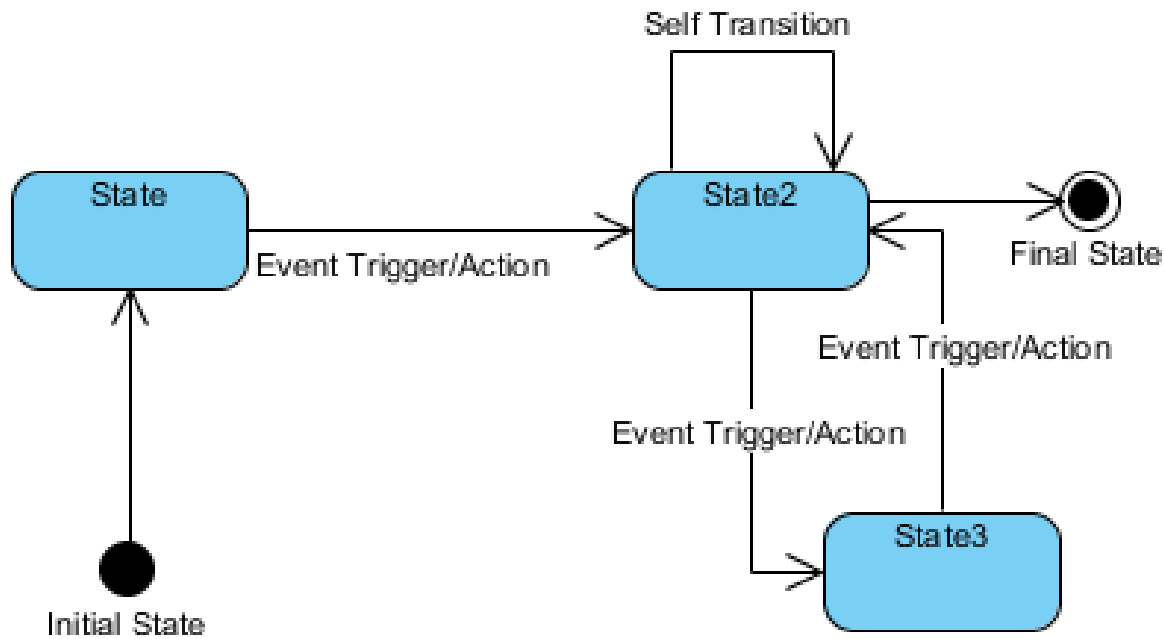
Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlocks the turnstile so that the customer can push through.
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When the customer has pushed through, locks the turnstile.



State Diagram

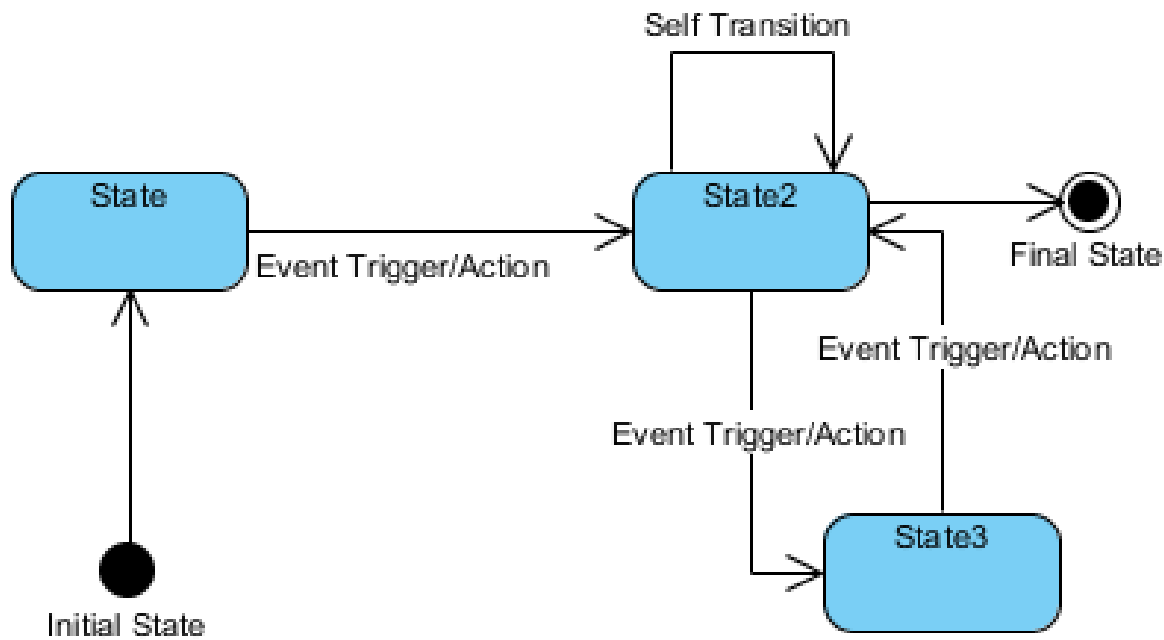
Toestandsdiagram (UML)

We gaan in het vervolg de UML (Unified Modeling Language) notatie gebruiken



State Diagram

Toestand (state)



- Een systeem of object bevindt zich op bepaald moment in één van de toestanden (**states**)
- Toestand wordt getekend dmv een rechthoek met afgeronde hoeken
- Een toestand heeft in ieder geval een naam



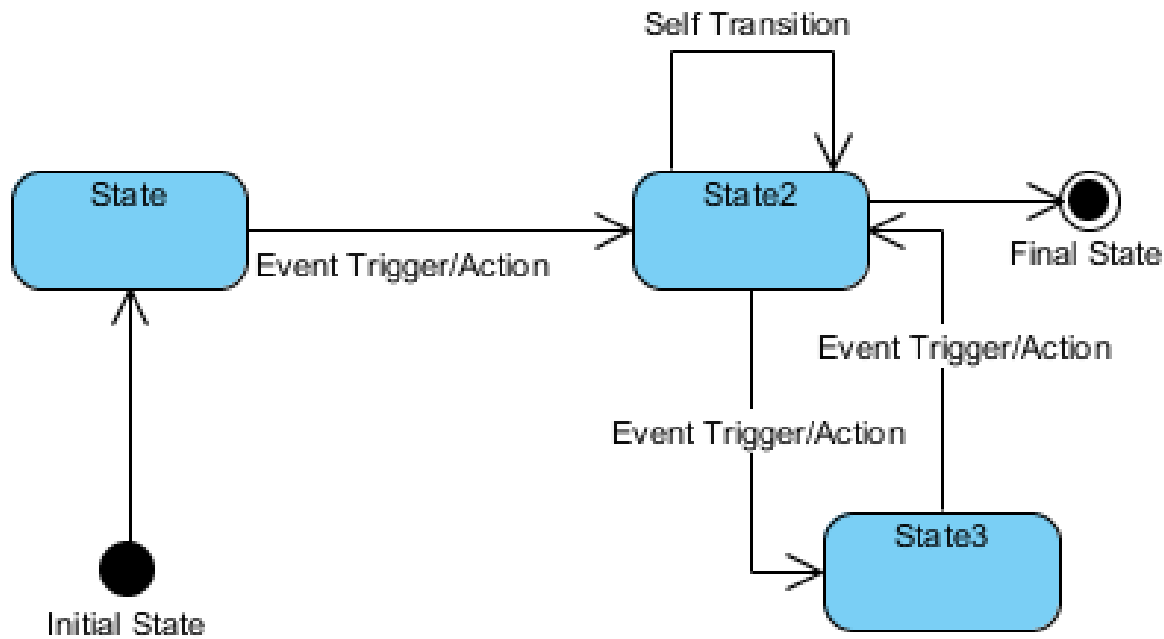
State Diagram

Begin en eind toestand

- Begintoestand (één)



- Eindtoestand (meerdere mogelijk)

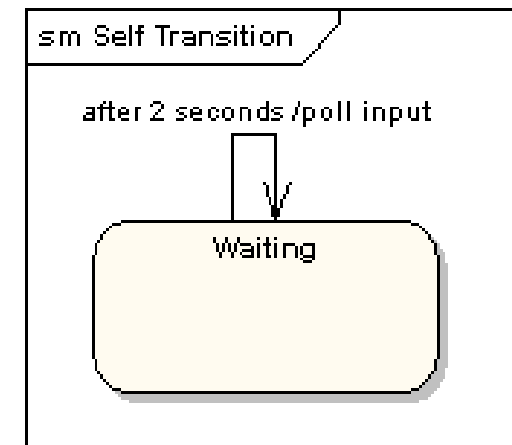
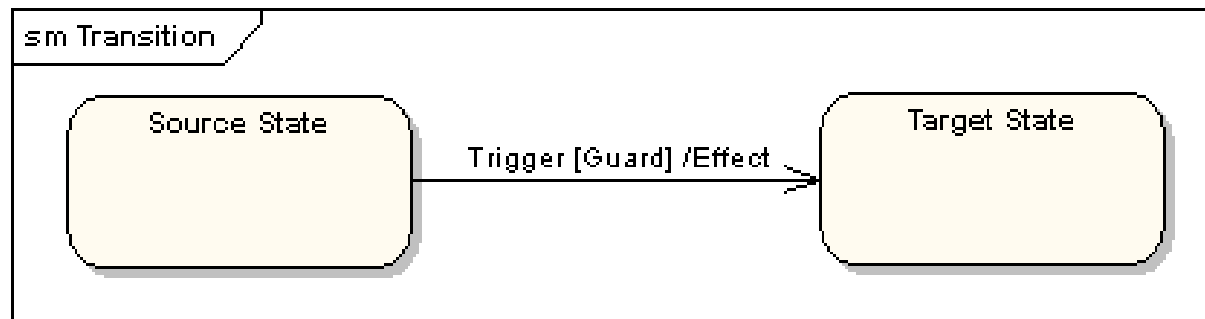


State Diagram

Transitie

Overgang van een toestand naar een (andere) toestand

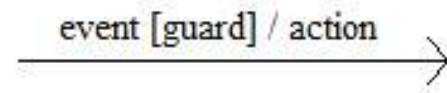
- Een transitie wordt veroorzaakt door een event (trigger)
- Een event is een gebeurtenis die geen tijd kost
- Kan ook naar dezelfde toestand (self transition)
- Notatie: pijl met event dat de transitie triggert



State Diagram

Event

- Een event is een gebeurtenis die geen tijd kost
- Komt binnen bij het systeem of object:
 - Gebruiker drukt op een knop (of een andere I/O activiteit)
 - Bericht van een (ander) object of systeem
 - Sensor meet een bepaalde waarde of gebeurtenis
 - Timeout van een timer

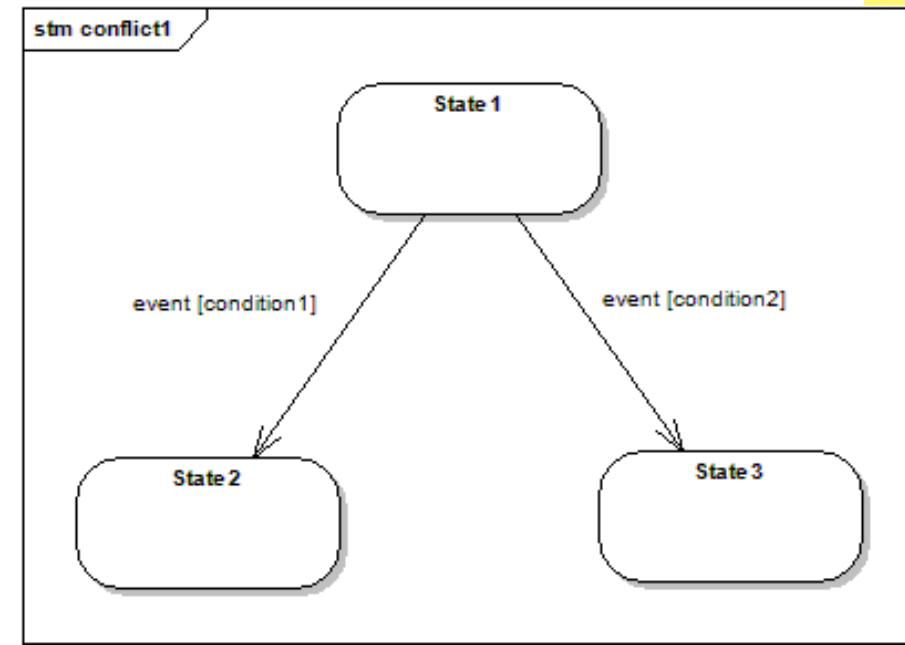
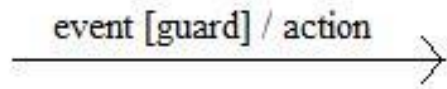


State Diagram

Guard

Guard (andere woorden: conditie of voorwaarde)

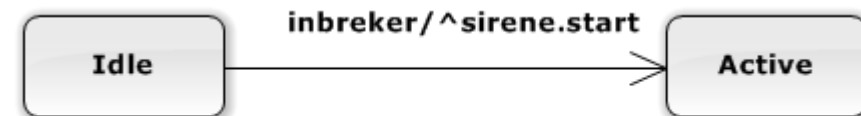
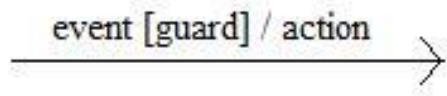
- Booleaanse expressie (T/F) die aangeeft of de transitie plaats kan vinden
- Transitie kan conditioneel zijn
- Notatie bij een transitie: event [conditie] / actie



State Diagram

Actie

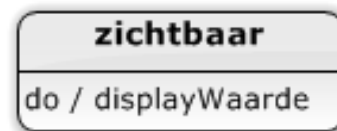
- Treedt op bij een transitie
- Atomaire operatie: duurt kort, niet te onderbreken (verschil met activiteit)
- Neveneffect van transitie
- Notatie: achter de event /
- Verzenden boodschap (^) naar ander object, of aanpassen attributen



State Diagram

Activiteit

- Vindt plaats binnen de state
- Loopt vanzelf af, of wordt onderbroken als de state wordt verlaten
- Kan dus ook een continue activiteit zijn, bv verwarmen of rijden
- Notatie binnen een state: do / activiteit



State Diagram

Entry en exit activiteiten

- entry activiteiten
 - acties die worden uitgevoerd als state binnen komt vanuit een andere state
 - notatie: entry / activiteit
- exit activiteiten
 - acties die worden uitgevoerd als de state verlaten wordt naar een andere state
 - notatie: exit / activiteit



- Duren kort



State Diagram

Toestandsvariabelen

- State variables
- Lokale attributen die tijdens de toestand gelden
- Later meer

- Voorbeeld?



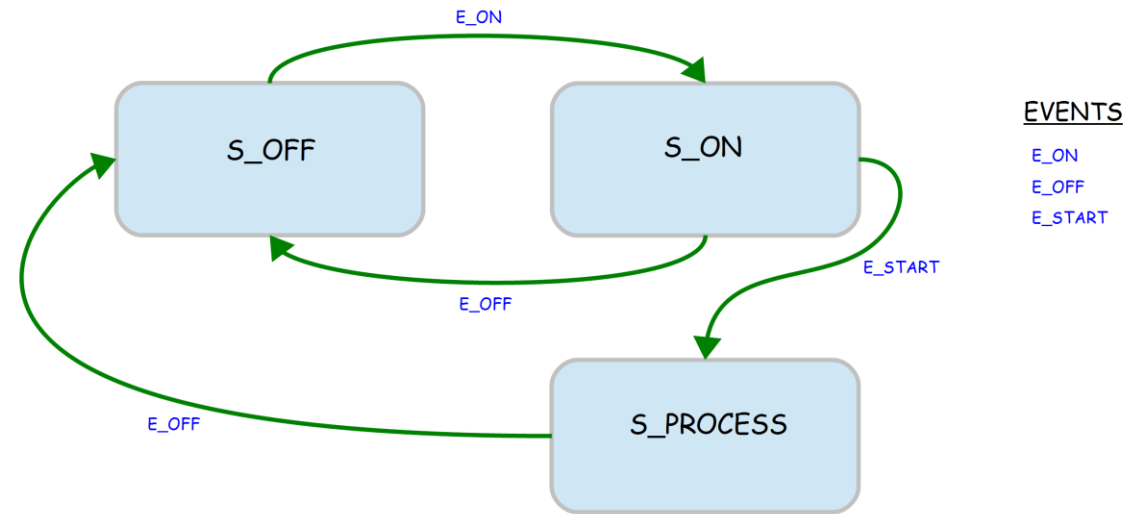
Stappenplan toestandsdiagram

1. Vind de toestanden waarin een object zich kan bevinden.
2. Vind de events die kunnen plaatsvinden
3. Vind voor iedere event de bijbehorende transitie(s).
4. Voeg eventueel begin- en eindtoestanden toe.
5. Voeg acties toe.
6. Voeg activiteiten toe.
7. Voeg eventueel overige informatie toe.



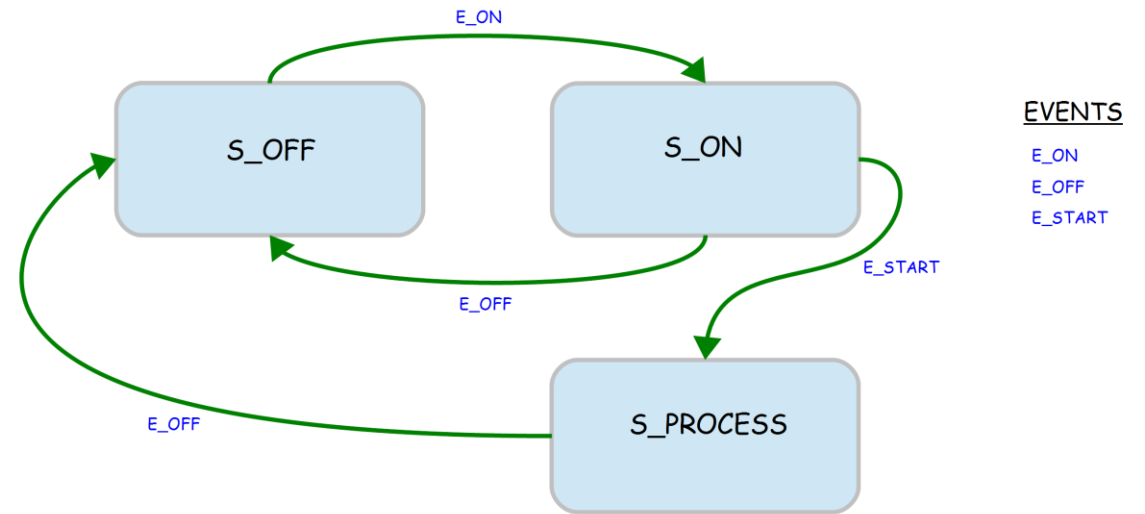
State Machine in code

- Gebruik enum voor states
- Gebruik enum voor events
- While (true)
- Lees inputs voor de events
- Maak een geneste switch case
- Eerst switch op de state
 - o binnen een state een switch op events
 - Bij event check eventueel op een guard conditie
 - Executeer de actie voor dat event
 - Bepaal de volgende state
- State wordt volgende state

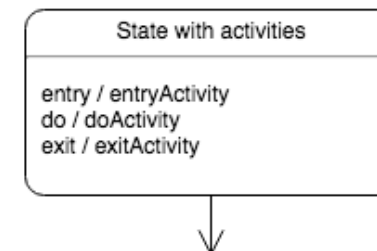


State Machine in code

- Gebruik enum voor states
- Gebruik enum voor events
- While (true)
- Lees inputs voor de events
- Maak een geneste switch case
- Eerst switch op de state
 - o binnen een state een switch op events
 - Bij event check eventueel op een guard conditie
 - Executeer de actie voor dat event
 - Bepaal de volgende state
- Als alle states gecheckt zijn: Als next state ongelijk aan huidige state:
 - o executeer dan de eventuele exit actie van de huidige state
 - o executeer dan de eventuele entry actie van de volgende state
- State wordt volgende state

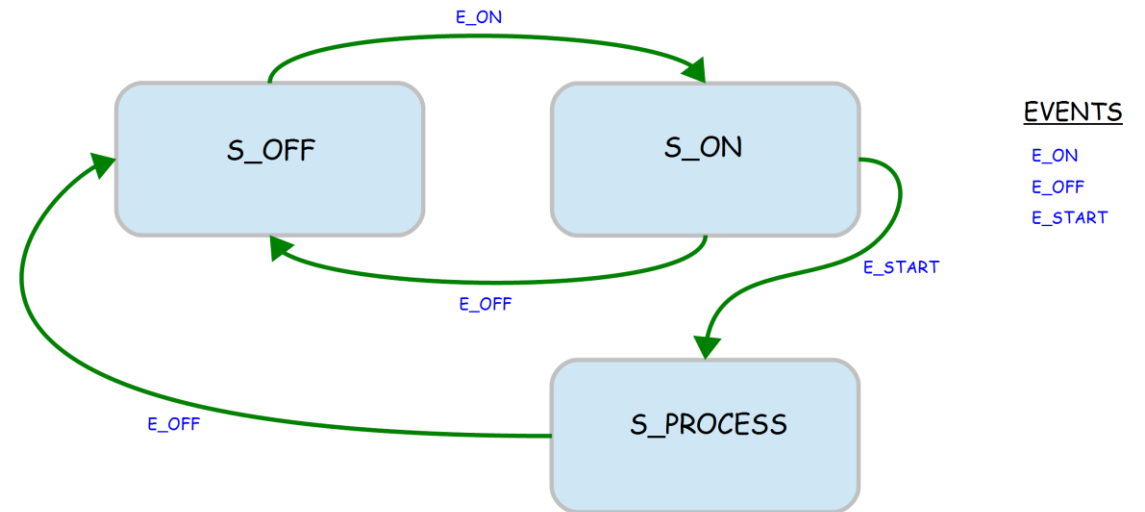


Als je een systeem hebt met activiteiten in de states:



State Machine in code

- Gebruik enum voor states
- Gebruik enum voor events
- While (true)
- Lees inputs voor de events
- Verwerk de event in de state machine



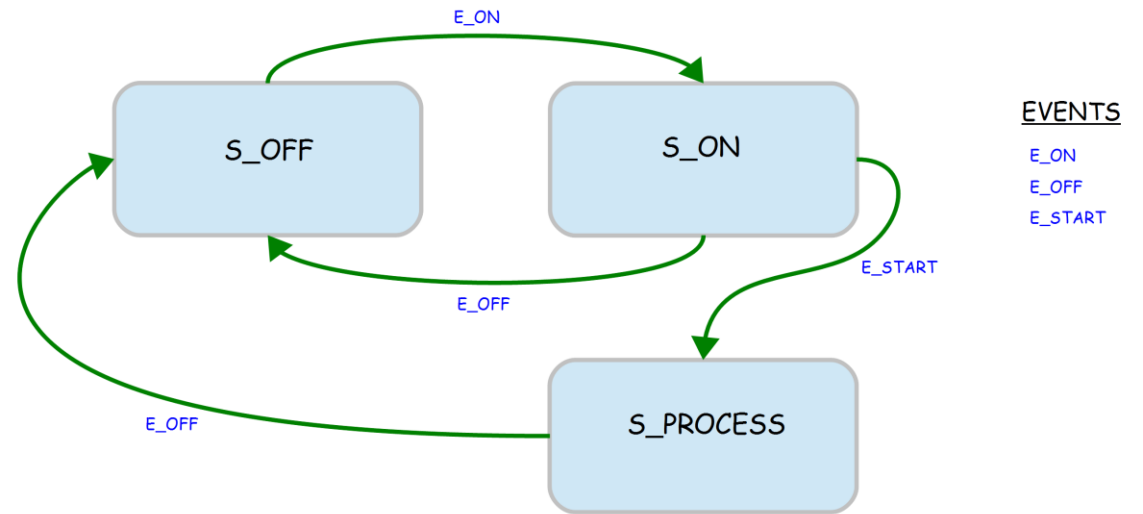
```
/*****  
 * State definitions, states are the routine the program jumps to, these are prefixed  
 * with an 'S' for easy identification  
 *****/  
typedef enum { S_OFF, S_ON, S_PROCESS, S_MAX } states;  
  
/*****  
 * Event definitions, events trigger a transition and are prefixed with an 'E' for  
 * easy identification  
 *****/  
typedef enum { E_OFF, E_ON, E_START, E_MAX } events;  
  
/*****  
 * Global variable  
 *****/  
states Current_State = S_OFF;  
events Current_Event = E_OFF;  
  
while (Current_State != S_MAX )  
{  
    Current_Event = HandleInput();  
    StateMachine(Current_Event);  
}
```



State Machine in code

- Gebruik enum voor states
- Gebruik enum voor events
- While (true)
- Lees inputs voor de events
- Verwerk de event in de state machine
- Maak een geneste switch case
- Eerst switch op de state
 - o binnen een state een switch op events
 - Bij event check eventueel op een guard conditie
 - Executeer de actie voor dat event
 - Bepaal de volgende state
- State wordt volgende state

Volledige code in demo en op de Brightspace



EVENTS

E_ON
E_OFF
E_START

```
/*****  
 * State Machine function definition containing the code to execute  
 *****/  
void StateMachine(events event)  
{  
    states Next_State = Current_State;  
  
    switch ( Current_State )  
    {  
        case S_OFF:  
            switch (event )  
            {  
                // A transition to the next state will occur here  
                case E_ON:  
                    // Execute Action for this event  
                    Next_State = S_ON;  
                    break;  
            }  
            break;  
        case S_ON:  
            switch (event )  
            {  
                // A transition to the next state will occur here  
                case E_OFF:  
                    // Execute Action for this event  
                    Next_State = S_OFF;  
                    break;  
            }  
            break;  
        case S_PROCESS:  
            switch (event )  
            {  
                // A transition to the next state will occur here  
                case E_OFF:  
                    // Execute Action for this event  
                    Next_State = S_OFF;  
                    break;  
            }  
            break;  
    }  
}
```



Aan het werk!

- Werk het State Diagram van jouw case uit
- Let op: elk object heeft een eigen state diagram

