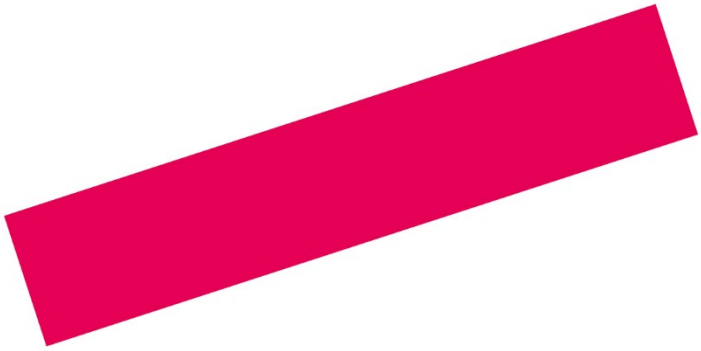


VAN STATE CHART NAAR FINITE STATE MACHINE_

[ondertitel]

John van den Hooven

28 oktober 2021

HAN_UNIVERSITY OF APPLIED SCIENCES

INHOUDSOPGAVE

INLEIDING

Bij de opdracht voor de EVL PROG2 (voltijd) en de EVL Inleiding Software Engineering Software Architectuur wordt gebruik gemaakt van een Finite State Machine (FSM).
Ten behoeve van deze opdracht is een C bibliotheek ontwikkeld die je kunt gebruiken om de ontworpen FSM te implementeren, te simuleren en te testen.

Deze bibliotheek bestaat uit twee delen:

- de Finite State Machine functies (fsm.h)
- een set simulatie en test console functies (devConsole.h e.a.)

Het gebruik van de FSM biblotheek wordt in dit document beschreven. De console functies zijn beschreven in hoofdstuk … van Introduction C Programming.

Deze instructie start nadat je op basis van de functionele specificaties een ontwerp voor de aparaat besturing hebt gemaakt. Dat betekent dat je bij het begin van deze instructie beschikt over:

- een architectuur ontwerp van de aparaat besturing
- een state chart die behoort bij de aparaat besturing

Het raamwerk van de FSM implementatie is opgesteld door Hugo Arends en vervolgens verder ontwikkeld voor gebruik in de opdracht van deze EVL.

Voor dat je aan dit document begint moet je eerst:

- ***Introduction C Programming Chapter*** (Hooven, 2021-2022 v0.1)
- ***UML Tutorial Finite Statemachines*** (Martin, June 98)
- ***The Assignment***

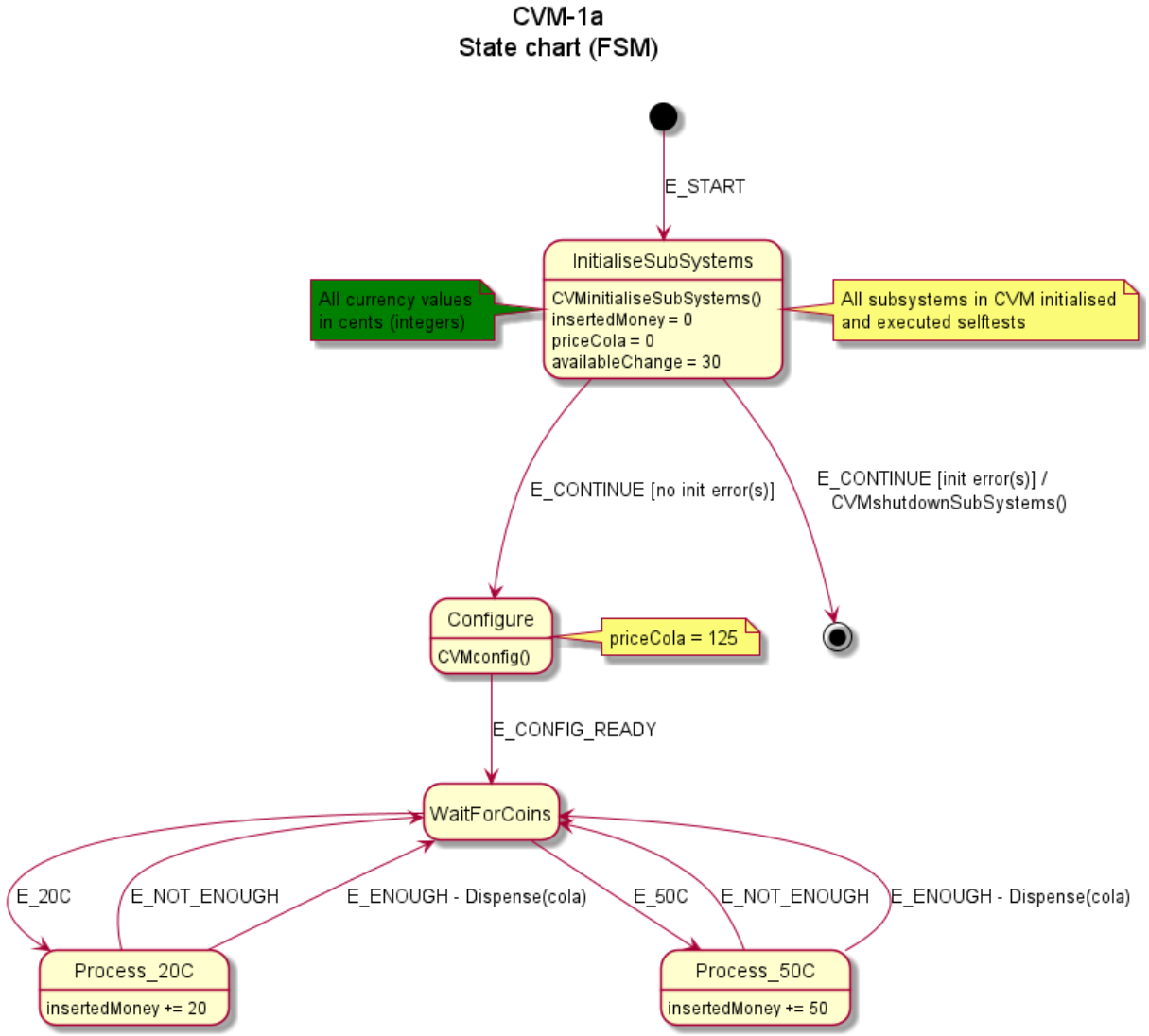
bestuderen en een state chart van het gekozen apparaat maken.

Verder ter inspiratie: https://youtu.be/7gW1hmYqdwo (Inventions, 2017)

1 VAN STATE CHART NAAR FSM IN HET PROGRAMMA.

1.1 De state chart

Begin met het goed opzetten van de State Chart van het apparaat dat bestuurt gaat worden.



Een eenvoudig model dat als voorbeeld wordt gebruikt

In de state chart worden de statussen, gebeurtenissen en transities weergegeven. Ook wordt per status beschreven welke acties er uitgevoerd (moeten) gaan worden. De implementatie van de state machine (FSM) gaat uit van een functie die wordt uitgevoerd bij het binnengaan van de status en een functie die wordt uitgevoerd bij het verlaten van de status.
In de state chart worden deze functies benoemt en wordt er informatie toegevoegd die beschrijft wat er binnen deze functies aan acties moeten plaatsvinden.

Figuur 1 is een voorbeeld van een state chart, deze state chart zal als voorbeeld worden gebruikt in de codering van deze instructie.
Zorg ervoor dat alle statussen (states) en gebeurtenissen (events) een correcte naam hebben gekregen en dat duidelijk is welke acties er moeten plaatsvinden tijdens een status transitie. Wordt een van deze functies niet gebruikt (hij is leeg) dan kan dat worden aangeduid met NULL. Deze aanduiding zal ook bij het coderen van de state machine worden gehanteerd.

2 DE FINITE STATE MACHINE FUNCTIES

De bibliotheek van de Finite State Machine zoals deze is opgenomen in het raamwerk bevat functies die gebruikt worden om de state machine te definiëren (configureren) en functies die gebeurtenis (event) kunnen genereren of simuleren.

- Het genereren van een event wordt gebruikt voor een interne gebeurtenis (intern event), een intern event wordt geactiveerd door de statemachine zelf (in de code dus). Een voorbeeld van een intern event is het afronden van de initialisatie (S_INIT) van de automaat (E_CONTINUE).
- Het simuleren van een event wordt gebruikt om een externe gebeurtenis (external event) te activeren. Een voorbeeld van een external event is de selectie door de user om te openen of het systeem af te sluiten. In de state chart staat een tweede voorbeeld, dat is het signaal van een sensor die aangeeft dat iemand de poort passeert (E_PASSED).

2.1 Het definiëren van de State Machine

Het definiëren van de state machine gebeurt in een aantal stappen:

- Definiëren van de events en states als symbolic constants
- Definiëren en configureren van de state machine

2.1.1 Definiëren van de events en states als symbolic constants

1) Maak een de enumerated symbolic constants aan voor de events (*events.h*)

```
typedef enum {
    E_NO,                ///< Used for initialisation of an event variable
    E_START,             ///< First event after example is switched on
    E_CONTINUE,          ///< Initialising subsystems is ready
    E_CONTINUE_ERROR,    ///< Initialisation error
    E_CONFIG_READY,      ///< User action is open gate
    E_20C,               ///< Coin acceptor detects 20c coin
    E_50C,               ///< Coin acceptor detects 50c coin
    E_NOT_ENOUGH,        ///< Total inserted money is not enough
    E_ENOUGH,            ///< Total inserted money is enough
    E_EXIT_SYSTEM,       ///< System exception or shutdown
} event_t;
```

2) Maak een bijpassende array aan om deze constants te vertalen naar een tekst (*events.c*)

```
char * eventEnumToText[] =
{
   "E_NO",                ///< Used for initialisation of an event
variable
   "E_START",             ///< First event after example is switched on
   "E_CONTINUE",          ///< Initialising subsystems is ready
   "E_CONTINUE_ERROR",    ///< Initialisation error
   "E_CONFIG_READY",      ///< User action is open gate
   "E_20C",               ///< Coin acceptor detects 20c coin
   "E_50C",               ///< Coin acceptor detects 50c coin
   "E_NOT_ENOUGH",        ///< Total inserted money is not enough
   "E_ENOUGH",            ///< Total inserted money is enough
   "E_EXIT_SYSTEM",       ///< System exception or shutdown
};
```

3) Maak een de enumerated symbolic constants aan voor de states (*states.h*)

```
typedef enum {
   S_NO,                        ///< Used for initialisation if state is
not yet known
   S_START,                     ///< Initial state
   S_INITIALISESUBSYSTEMS,      ///< Initialised subsystems
   S_CONFIGURE,                 ///< Ask user for action
   S_PROCESS_20C,               ///< Handle the 20c insert
   S_PROCESS_50C,               ///< Handle the 50c insert
   S_WAITFORCOINS,              ///< Wait for the user to insert coins
   //end
} state_t;
```

Maak een bijpassende array aan om deze constants te vertalen naar een tekst (*states.c*)

```
char * stateEnumToText[] =
{
     "S_NO",                      ///< Used for initialisation if
                                  /// state is not yet known
     "S_START",                   ///< Initial state
     "S_INITIALISESUBSYSTEMS",    ///< Initialised subsystems
     "S_CONFIGURE",               ///< Ask user for action
     "S_PROCESS_20C",             ///< Handle the 20c insert
     "S_PROCESS_50C",             ///< Handle the 50c insert
     "S_WAITFORCOINS",            ///< Wait for the user to insert
                                  /// coins
     "S_SHUTDOWN_SYSTEM",         ///< Exception or system exit
                                  /// shutdown requested
};
```

2.1.2 Definiëren en configureren van de state machine

Voor de state machine wordt de informatie van het model in de state chart in het model van het programma opgeslagen. Dit betekent dat van alle states de naam van de state, een verwijzing naar de

functie voor "onEntry" en een verwijzing voor de functie voor "onExit" moet worden geconfigureerd. Dit gebeurt met een functie `FSM_AddState();`

Vervolgens moeten de transities (reacties op een gebeurtenis) worden geconfigureerd. Dit betekent dat de state waarin de statemachine verkeerd, het event dat optreedt en de nieuwe state als gevolg van het event moet worden ingesteld. Dit gebeurt met een functie `FSM_AddTransition();`

Omdat het model de basis vormt van de state machine moeten beide sets functies bij de start van het programma te worden uitgevoerd. <<tekenining maken>>

FSM_Addstate()

```
void    FSM_AddState(const state_t state, const state_funcs_t *funcs);
```

`state`	de symbolic name van de state
`*funcs`	een pointer naar een struct waarin de "onEntry" en de "onExit" functies worden gedefinieerd. Deze struct is van het type `state_funcs_t` en bevat de elementen: • `funcs.onEntry`, een verwijzing naar de "onEntry" functie • `funcs.onExit`, een verwijzing naar de "onExit" functie Als een van deze twee waarden een NULL waarde krijgen, wordt er geen functie uitgevoerd bij de bijbehorende transitie.

In Table 1 Is een de voorbeeldcode opgenomen die past bij het hier gebruikte model.

Which came first, the chicken or the egg?
When coding you may face the Chicken or Egg problem. The functions you use in the model must also be created (defined and implemented), ie what must happen when entering or leaving a state. It is wise to make these functions (empty if necessary) and implement them before you start configuring the model. But you don't have to, you can also create them after configuring the model. The important thing is that you create them before you compile the code for the first time, otherwise you will get a compiler error for every function included in ***FSM_AddState()****. It is also advisable to first set up the functions as a prototype and as a "skeleton".*

FSM_AddTransition()

```
void    FSM_AddTransition(const transition_t *transition);
```

`*transition`	Dit is een struct waarin de transitie kan worden gedefinieerd. Deze struct bevat de volgende elementen: • `transition.from` bij welke state behoort deze transitie • `transition.event` het event dat de transitie triggert • `transition.to` naar welke state gaat deze transitie

In Table 2 is de voorbeeldcode passend bij het hier behandelde model opgenomen.

Table 1 Code om states toe te voegen aan het model van de state machine

```
//           State                                  onEntry()                        onExit()
FSM_AddState(S_INITIALISESUBSYSTEMS,&(state_funcs_t){ S_InitialiseSubsystems_onEntry, S_InitialiseSubsystems_onExit });
FSM_AddState(S_CONFIGURE,           &(state_funcs_t){ S_Configure_onEntry,           S_Configure_onExit});
FSM_AddState(S_WAITFORCOINS,        &(state_funcs_t){ S_WaitForCoins_onEntry,        S_WaitForCoins_onExit });
FSM_AddState(S_PROCESS_20C,         &(state_funcs_t){ S_Process_20C_onEntry,         S_Process_20C_onExit });
FSM_AddState(S_PROCESS_50C,         &(state_funcs_t){ S_Process_50C_onEntry,         S_Process_50C_onExit });
```

Table 2 Code om de transities uit het model toe te voegen

```
//                                  From                      Event                To
FSM_AddTransition(&(transition_t){ S_START,                   E_START,         S_INITIALISESUBSYSTEMS    });
FSM_AddTransition(&(transition_t){ S_INITIALISESUBSYSTEMS,    E_CONTINUE,      S_CONFIGURE });
FSM_AddTransition(&(transition_t){ S_CONFIGURE,               E_CONFIG_READY,  S_WAITFORCOINS });
FSM_AddTransition(&(transition_t){ S_WAITFORCOINS,            E_20C,           S_PROCESS_20C });
FSM_AddTransition(&(transition_t){ S_WAITFORCOINS,            E_50C,           S_PROCESS_50C });
FSM_AddTransition(&(transition_t){ S_PROCESS_20C,             E_NOT_ENOUGH,    S_WAITFORCOINS });
FSM_AddTransition(&(transition_t){ S_PROCESS_50C,             E_NOT_ENOUGH,    S_WAITFORCOINS });
FSM_AddTransition(&(transition_t){ S_PROCESS_20C,             E_ENOUGH,        S_WAITFORCOINS });
FSM_AddTransition(&(transition_t){ S_PROCESS_50C,             E_ENOUGH,        S_WAITFORCOINS });
```

2.2 De state machine

De state machine wordt geïmplementeerd door een oneindige lus die de events en de transities afhandelt. De actuele status wordt bewaard in de globale variabele `state` en het actuele event wordt bewaard in de globale variabele `event`.

De state machine wordt "gestart" door de initiële state in te stellen en het eerste event te genereren (of te simuleren)[1]:

```
// Set the finite state machine in the S_START state
// and add a E_START event into the event buffer
state = S_START;
FSM_AddEvent(E_START);
```

Vervolgens start de state machine en voert achtereenvolgens de volgende functies uit:

```
while(1)
{
   if(!FSM_NoEvents())
   {
      // Get the event and handle it
      event = FSM_GetEvent();
      state = FSM_EventHandler(state, event);
   }
}
```

`state_n = FSM_EventHandler(state_c, event);`[2]

Deze functie behandelt de reactie op *`event`:*

1) bepaald wordt welke transitie er plaatsvindt als `event` optreedt bij `state_c`, dit levert een nieuwe status op `state_n`.
2) De ***_onExit*** functie van `state_c` wordt uitgevoerd
3) De ***_onEntry*** functie van `state_n` wordt uitgevoerd

[1] Normaal gesproken zal dit eerste event extern worden gegenereerd bijv. door het aanzetten van het apparaat. In het programma wordt dit gesimuleerd door een `FSM_AddEvent()`.
[2] De functie die gebruikt wordt in de code is `state = FSM_EventHandler(state, event);`, hier wordt state_c en state_n gebruikt om de twee variabelen uit elkaar te houden.

3 INRICHTING VAN DE _ONENTRY EN _ONEXIT FUNCTIES.

3.1 onEntry en onExit functies

De _onEntry en de _onExit functies zorgen voor het besturen van het systeem. De _onEntry wordt uitgevoerd bij het binnengaan van een status. De _onExit functie wordt uitgevoerd bij het verlaten van de status.
In de _onEntry functie kunnen de acties plaatsvinden die binnen de status moeten worden uitgevoerd tijdens de status en deze acties leiden vervolgens tot een nieuwe gebeurtenis, een event dus.

3.2 Voorbeeld aan de hand van de state chart.

Tijdens de status `S_WAIT_FOR_COINS` heeft de gebruiker van het systeem de mogelijkheid om een activiteit te bewerksteliggen. De gebruiker kan daarbij een munt inwerpen van 20c of van 50c. Bij het apparaat dat bestuurd wordt betekent dat dat de gebruiker een van de twee munten kan inwerpen in de munt-automaat..
Afhankelijk van de munt die ingeworpen wordt zal deze functie een event genereren: `E_20C` of `E_50C`. *Bij een simulatie van het systeem wordt dit door een vraag op de ontwikkelconsole (terminal) aan de gebruiker te stellen gesimuleerd.*
Vervolgens zal de functie het nieuwe event aan de FSM doorgeven door een `FSMaddEvent()` functie. De FSM zal vervolgens het event afhandelen en doorstappen naar de nieuwe status (`S_PROCES_20C` of `S_PROCES_20C`).
De code voor dit voorbeeld kan er als volgt uit zien:

```
void S_WaitForCoins_onEntry(void)
{
   event_t coin = getCoin(); ///Get coin information coinacceptor Subsystem
   FSM_AddEvent(coin);       ///Inserted coin is a (simulated) external
event
}
```

De `getCoin()` functie is van het type `event_t`:

```
event_t getCoin(void)            /// coinacceptor
{
   char kb;
   event_t coin = E_EXIT_SYSTEM;
   kb = DCSsimulationSystemInputChar("Please insert a coin enter 1 for 20c
enter 2 for 50c", "12");
   switch (kb)
   {
      case '1':
         coin = E_20C;
         break;
      case '2':
         coin = E_50C;
         break;
      default:
```

```
            coin = E_EXIT_SYSTEM;
    }
    return coin;
}
```

3.3 Voorbeeld indien er meerdere events zijn die tot een transitie leiden.

In het voorbeeld wordt tijdens de state S_PROCESS_20C gecontroleerd of er genoeg geld is ingegooid. Als er niet genoeg geld is wordt opnieuw gewacht op munten (E_NOT_ENOUGH → S_WAIT_FOR_COINS), is er wel genoeg geld dan zal in dit model ***voor*** het terugkeren naar S_WAIT_FOR_COINS een blikje Cola geleverd moeten worden en het saldo van ingegooid geld wordt weer op 0 gezet[3].
Dit kan in de `_onExit` functie worden afgehandeld:

```
void S_Process_20C_onExit(void)
{
/*
 * In this onExit function first the transition event is checked
 * This gives information about enoughe/no-enough money
 * to get a can of cola
 *
*/
   DCSdebugSystemInfo("Curent state: %s, Current event: %s", stateEnumToText[state],
eventEnumToText[event]);   ///Debug info

   if (event == E_ENOUGH)
   {
      dispenseCola();      /// Let the dispense subsystem deliver a can of Cola
      insertedMoney = 0;   /// Reset the debet
   }
   else
   {
      DSPshow(6, "Not enough money inserted");     /// Not enough, ask for more :-)
      DSPshow(7, "Please insert coins");
   }
}
```

Voor de volledigheid, als je kunt dus door gebruik te maken en te testen van de variabelen event bepalen wat het event is dat tot de transitie en indien nodig op basis daarvan een extra handeling verrichten.

[3] Dit kan ook anders ontworpen worden, dan wordt het leveren van het blikje en alle handeling die daarbij horen een aparte status. Daar kan namelijk ook nog van alles gebeuren. In het model gaan we er van uit dat dit altijd "goed lukt". Dat voor de eenvoud.

4 CODE STIJL AFSPRAKEN

Om je code goed leesbaar te houden voor anderen (en ook voor je zelf) maken we naast de in de codestyle guide vermelde stijl afspraken nog een aantal afspraken die direct betrekking hebben op de implementatie van de Finite State Machine.

- Zorg dat er altijd een duidelijk verband is tussen de namen van de gebeurtenissen (events) en statussen (states) en bij behorende functies of variabelen.
- Statusnamen beginnen met S_ en worden in hoofdletters geschreven (S_INITIALISESUBSYSTEMS)[4]
- Gebeurtenisnamen (events) beginnen met E_ en worden in hoofdletters geschreven (E_CONTINUE)
- Functies die behoren bij een status beginnen met S_, gebruiken camelCasing en eindigen op _onEntry (binnenkomst) of _onExit (verlaten). (S_InitialiseSubSystems_onEntry en S_ InitialiseSubSystems _onExit). Kies daarbij namen die passen bij de state waarvoor ze geschreven zijn.

[4] In de statechart worden de namen met kleine letters en camel casing geschreven i.v.m. de leesbaarheid van de chart. Bij het coderen in C worden dit hoofdletters omdat deze namen als Enumerated Symbolic Constants worden gedefinieerd.

BIBLIOGRAFIE

Hooven, J. v. (2021-2022 v0.1). *Introduction C programming* (Vol. 2021). Arnhem: HAN.

Inventions, H. (2017, juni dag). *How to Make Cola Vending Machine at Home - DIY*. Opgehaald van Youtube: https://youtu.be/7gW1hmYqdwo

Martin, R. C. (June 98). UML Tutorial: Finite State Machines. *C++ Report,*(Engineering Notebook Column).