Modeling by State Machine Diagrams

Prof. Hugo Vieira Neto
2020/1

Objective

- To use state machine diagrams as a tool for dynamic modeling of embedded systems:
  – States and transitions
  – Actions and activities
  – Events and conditions

Embedded Systems

- Reactive artifacts
  – React to external events
  – Can generate internal events
  – React to these internal events
- React = generate outputs, change states, change internal variables (part of the state)

Embedded Systems

- It is necessary to represent the dynamic behavior of the system as a function of time and specific events, indicating how it will react to these events (modeling)
- Statecharts serve very well to the purpose of modeling the dynamic behavior of a system

State Machine Diagrams

- Possible states that a given system can go through, as well as transitions between them (associated to each triggering event and under which constraints)
- Used by hardware and software designers to represent finite state machines (FSM)

Finite State Machines

- State = stable situation of an FSM
- Transition = indicates the possibility of exiting one state and entering another
  – Trigger: event that causes the transition
  – Guard: necessary constraint to carry out the transition (besides the occurrence of the event)
  – Behavior: action taken during the transition
- Event = relevant event in a well-defined instant of time – can be internal or external

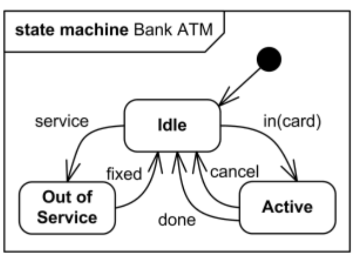
Origins: Statecharts

- Professor David Harel
  - Weizmann Institute (Israel), 1984
- Contributions:
  - History
  - Hierarchy
  - Concurrence
- Reference:
  - Harel, D., Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming 8(3), pp. 231-274, 1987.

Basic Graphical Notation

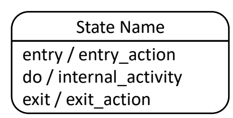
- Frame with box for the name of the FSM
- Initial pseudo-state ●
- Final pseudo-state ◉
- State ▭
- Transition ⟶

Basic Graphical Notation



Graphical Notation: States

- State (state name)
  - Internal activity : **do /**
  - Actions generated by internal event: **event /**
  - Actions generated by entry or exit: **entry /**, **exit /**



Actions vs Activities

- Actions occur in transitions (atomic)
  - Entry actions are performed when the transition crosses the state boundary while entering
  - Exit actions are performed when the transition crosses the state boundary while exiting
- Activities take place while within the state (end → change of state)

Graphical Notation: Transitions

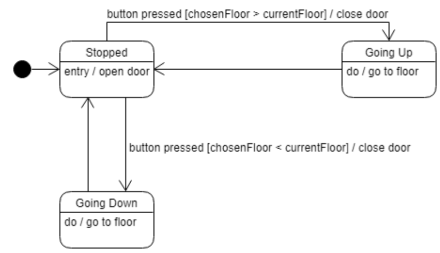
- Transition:
  - Trigger (trigger name)
  - Guard: logical expression (implicit **if**)
  - Behavior: list of actions separated by “;”
    - Assignment, function call, output activation, etc.

trigger [guard] / behavior ⟶

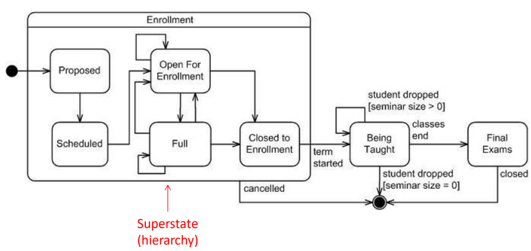
Events vs Guards

- Types of triggers:
  - Function call
  - Asynchronous signal arrival: IRQ, message
  - Passage of time (counted from entering the state): **after(period)** or **at(instant)**
- Types of guards:
  - Logical operators: ==, !=, <, >, ...
  - Generic operators: is_in(state), ...
  - [else]

Example: Elevator Control



Example: Enrollment Process



Exercise 1

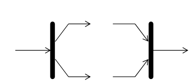
- Sketch a state machine diagram that models the dynamic behavior of operating mode changes for an ARM Cortex-M4 core
- Consider only the operating situation in which special registers PRIMASK = 1, FAULTMASK = 1 and CONTROL = 0
- Start by asking what are the events of interest for the operating situation above

Exercise 1

- Consider intermediate states of stacking, unstacking and tail-chaining, if that is the case
- In the representation of transitions caused by the execution of an instruction that causes an exception return (ex: BX LR), consider possible pending exceptions (bit)
- Use UML notation to characterize transitions: event [guard] / action

Advanced Graphical Notation

- Initial pseudo-state ●
- Final pseudo-state ◉
- Junction •
- Selection ◇
- Termination ✕
- Fork / Join (concurrence)

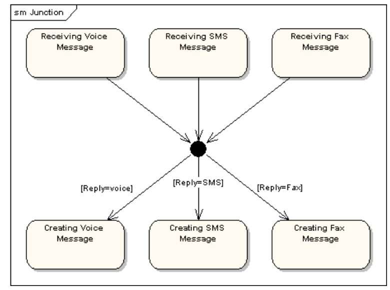


- Shallow history and deep history (H) (H*)

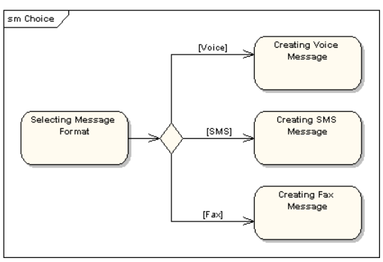
Initial and Final Pseudo-states

- States linked to the initial pseudo-state are those in which the system can enter when initialized
  - At least one is needed
  - If there is more than one, the entry conditions for each state must be specified
- States linked to the final pseudo-state are those from which the system will no longer leave
  - Any quantity is allowed
  - No transitions to other states

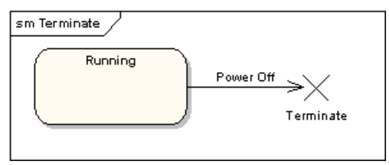
Example: Junction



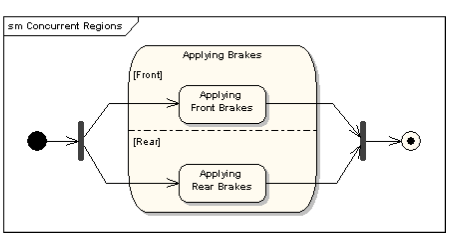
Example: Selection



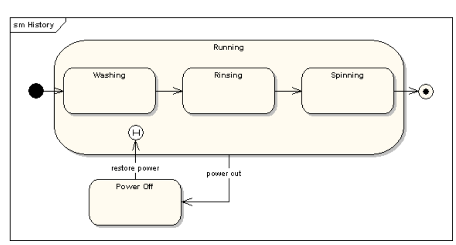
Example: Termination



Example: Concurrence (Fork/Join)



Example: Shallow History



FSM Implementation

- Approaches:
  1. State selection
  2. Event selection
  3. State-Event Matrix
- State identification:
  - Situation (outputs)
  - Memory (variables)

FSM Implementation

- The current state is stored in a variable, usually an enumeration of the states that make up the FSM
- The event is detected (example: IRQ) and its occurrence is reported by:
  - Variable changed by ISR (bare metal)
  - Asynchronous message from ISR to thread (RTOS)

State Selection

```
typedef enum {State_0, State_1, State_2} state_t;
volatile uint8_t Event = 0; // altered by ISR

void thread(void){
  state_t State = State_0; // FSM initial state
  while(1){
    switch(State){
      case State_0:
        if(Event == 1){
          // actions and change in state
        } // if
        break;
      ⋮
    } // switch
  } // while
} // thread
```

Project “fsm_state” from workspace“EK-TM4C1294XL_IAR8”

Exercise 2

- Sketch a state machine diagram that describes the dynamic behavior of the “fsm_states” project
- How to implement entry and exit actions (entry / and exit /) in the states and their activities (do /) in the state selection approach?

Event Selection

```
typedef enum {State_0, State_1, State_3} state_t;
volatile uint8_t Event = 0; // altered by ISR

void thread(void){
  state_t State = State_0; // FSM initial state
  while(1){
    if(Event){
      switch(State){
        case State_0:
          // actions and change in estate
          break;
        ⋮
      } // switch
    } // if
  } // while
} // thread
```

Project “fsm_event” from workspace “EK-TM4C1294XL_IAR8”

Exercise 3

- Change the “fsm_event” project to show the forward sequence of the 3-bit Gray Code on LEDs D1, D2 and D3 of the EK-TM4C1294XL kit
- Tip: use a different state for each binary output pattern
  - 000 → 001 → 011 → 010 → 110 → 111 → 101 → 100 → 000 → …

State-Event Matrix

- Functions for each state are defined:

```
state_t func1(state_t curr){
  // actions
  return next; // mudança do estado
} // func1
```

- An array of pointers to the functions is created:

```
state_t (*matrix[N_EV][N_ST])(state_t) =
  {{func1, func2, func3},
   {func4, func5, func6}};
```

- The function corresponding to each detected event is executed:

```
State = (*matrix[Event][State])(State);
```

State-Event Matrix Implementation

```
state_t f_0(state_t curr){
  // actions
  return next; // change in state
} // f_0
'
void thread(void){
  state_t (*matrix[N][M])(state_t) = {{f_0, f_1, f_2},
                                      {f_3, f_4, f_5}};
  state_t State = State _0; // FSM initial state
  while(1){
    if(Event){
      State = (*matrix[Event - 1][State])(State);
      Event = 0;
    } // if
  } // while
} // thread
```

Project “fsm_matrix” from workspace “EK-TM4C1294XL_IAR8”

Exercise 4

- Sketch a state machine diagram that uses the concept of hierarchy to describe the dynamic behavior of the “fsm_matrix” project
- How to implement entry and exit actions (entry / and exit /) in the states and their activities (do /) in the state-event matrix approach?

Supplementary Material

- Software based Finite State Machine (FSM) with general purpose processors (Joseph Yiu)
- Blog: Máquina de Estados em C (Sergio Prado):
  - https://sergioprado.org/maquina-de-estados-em-c/
- Videos: State Machine Diagram (YouTube):
  - https://www.youtube.com/watch?v=_6TFVzBW7oo
  - https://www.youtube.com/watch?v=UzUUZRK_Q6Y
  - https://www.youtube.com/watch?v=ABA3TGQVhTg

Useful Tools

- Draw.io (drawing in the cloud)
  - https://www.draw.io/
- UMLetino (drawing in the cloud)
  - http://www.umlet.com/umletino/umletino.html
- Yakindu Statechart Tools (drawing/simulation)
  - http://statecharts.org/

References

- UML State Machine Diagrams
  - http://www.uml-diagrams.org/state-machine-diagrams.html
- State Machine Diagrams: An Agile Introduction
  - http://agilemodeling.com/artifacts/stateMachineDiagram.htm
- Sparx Systems – State Machine Diagram Tutorial
  - https://sparxsystems.com.au/resources/tutorials/uml2/state-diagram.html
- Lucidchart - State Machine Diagram Tutorial
  - https://www.lucidchart.com/pages/uml-state-machine-diagram