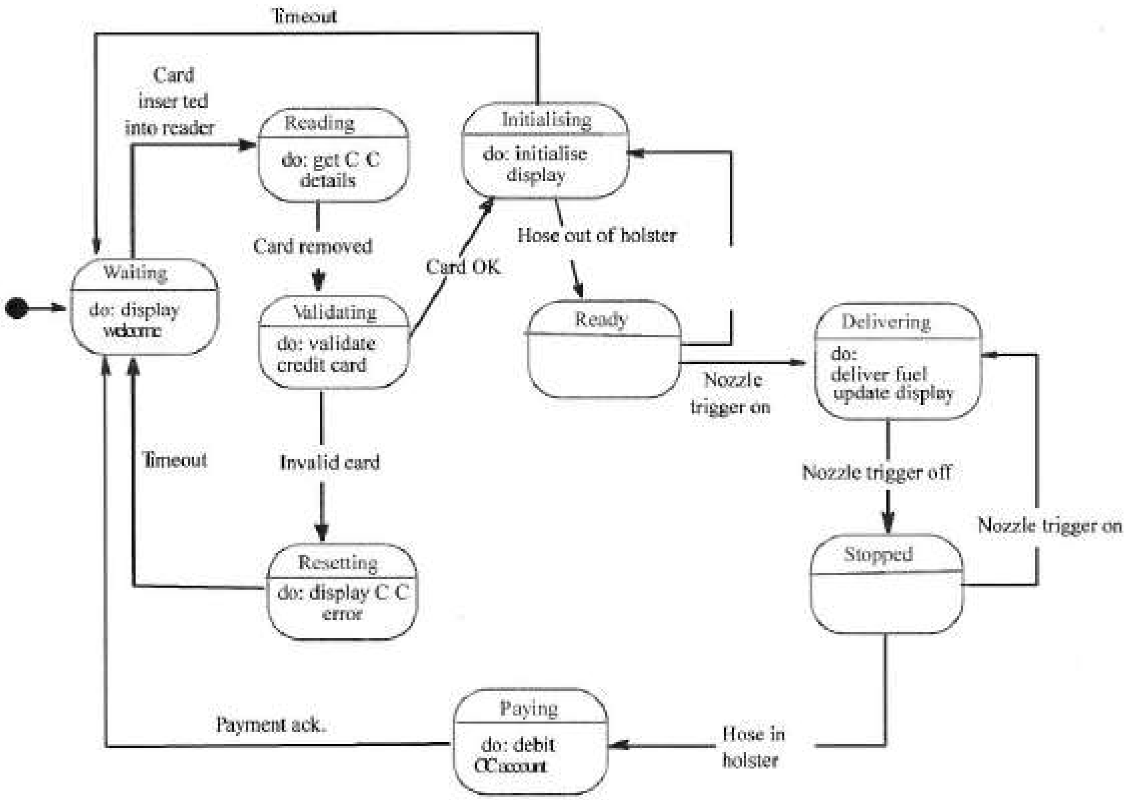
FINITE STATE MACHINES

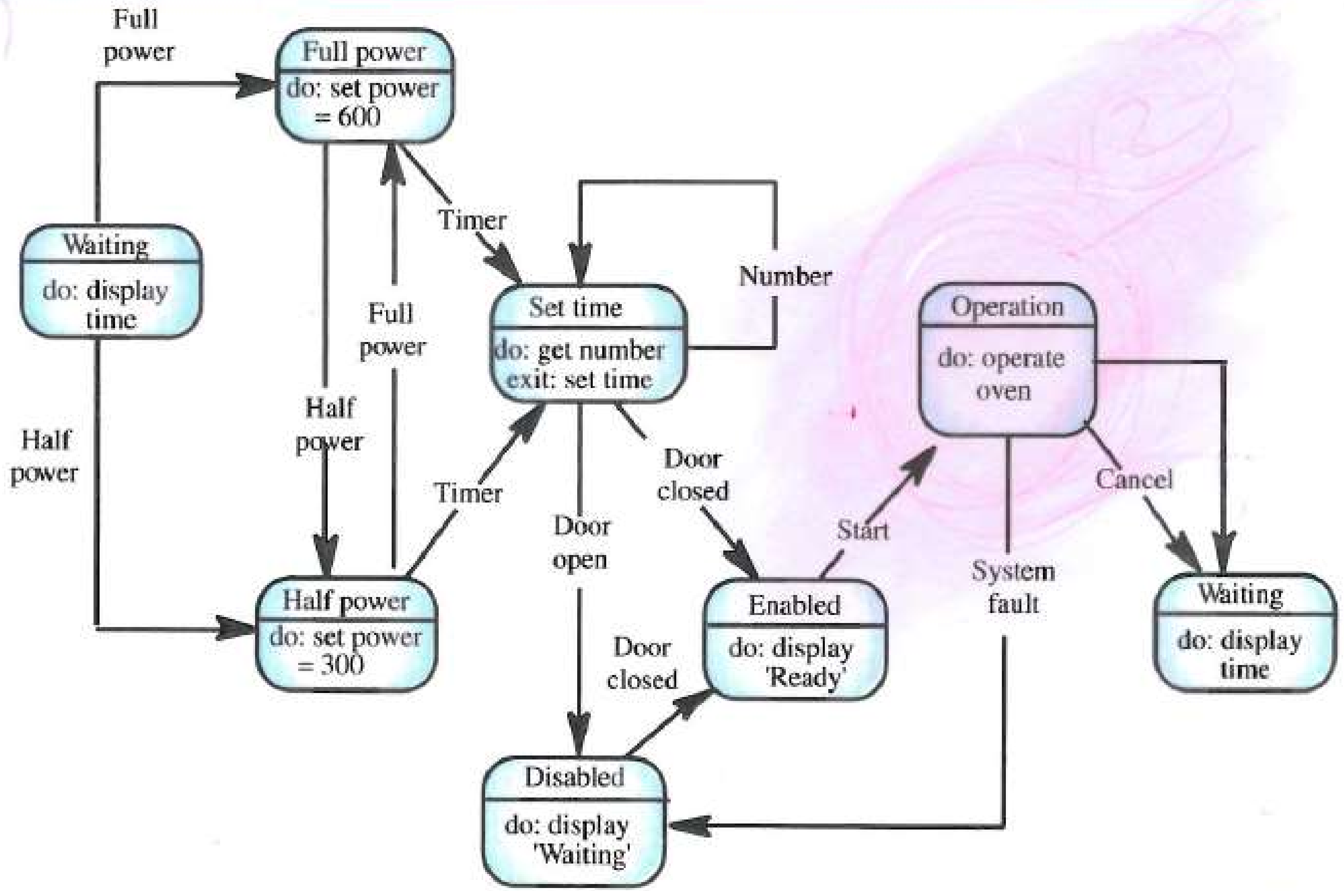
State machine modelling

- The effect of a stimulus in a real-time system is often to trigger a transition from one state to another.
- Finite state machines are therefore often an appropriate way of modelling real-time systems.
- The problem is the lack of structure in FSMs. Even simple systems are likely to have a complex model.
- Thread diagrams which show an event sequence are a means of managing the complexity in state machine models.

Petrol pump state model



Microwave oven state machine



Slide 17

State	Description
Half power on	The oven power output is set to 300 watts
Full power on	The oven power is set to 600 watts
Set time	The cooking time is set to the users input value
Operation disabled	Oven operation is disabled for safety. Interior oven light is on
Operation enabled	Oven operation is enabled. Interior oven light is off
Timed operation	Oven in operation cooking for the required time. Interior oven light is on.
Cooking complete	Timer has reached zero. Sound audible signal. Oven light is off.

Microwave oven State

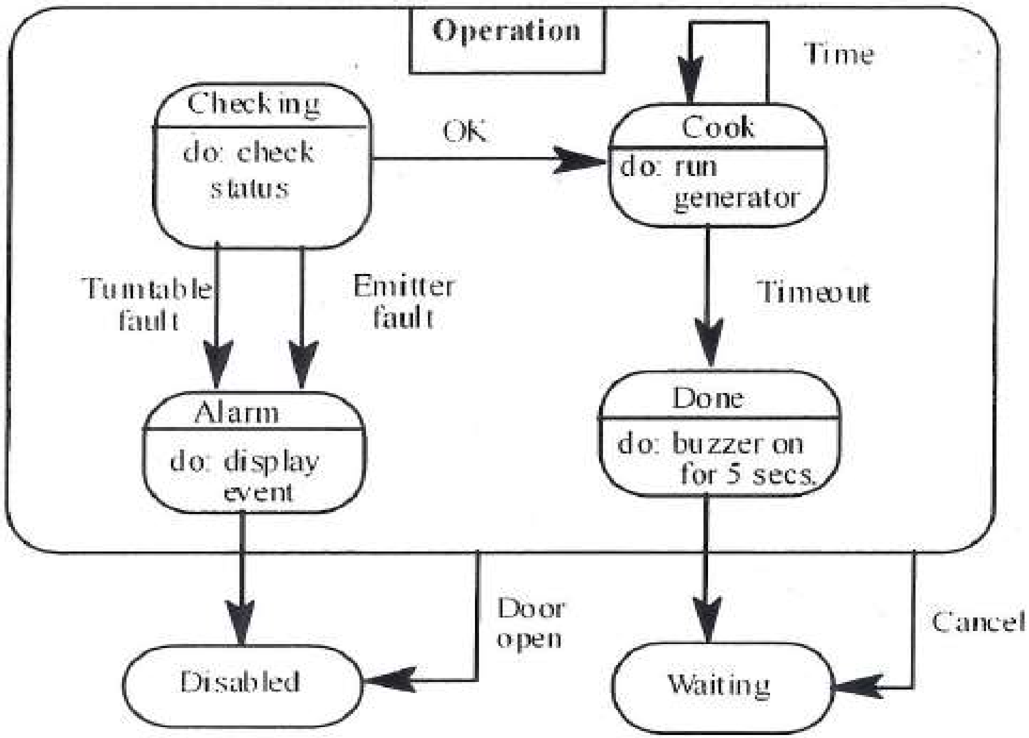
Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button.
Timer	The user has pressed one of the timer buttons
Door open	The oven door is not sealed
Door closed	The oven door is sealed.
Start	The user has pressed the start button
Timeout	The timer indicates that the set time has expired

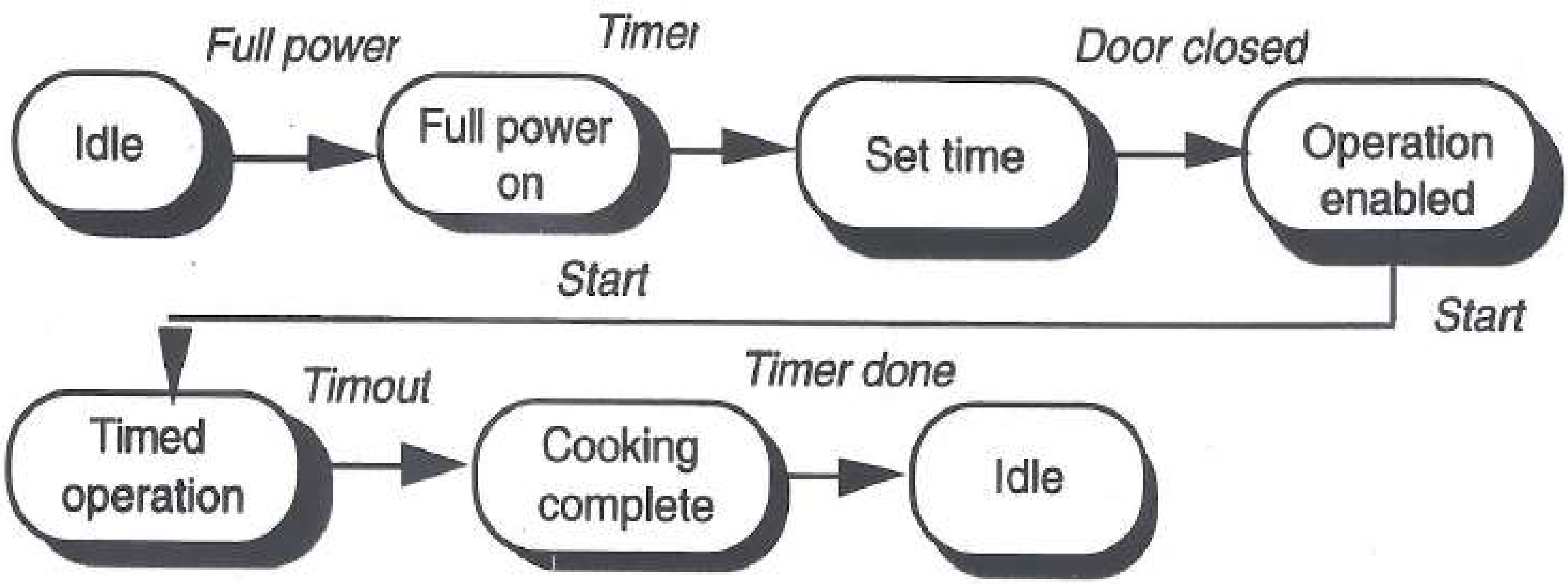
Microwave oven stimuli

Statecharts

- Allow the decomposition of a model into sub-models (see following slide)
- A brief description of the actions is included following the ‘do’ in each state
- Can be complemented by tables describing the states and the stimuli

Microwave oven operation





Thread diagram - full power cooking

PDL

```
task Building_monitor is
     entry Initialize ;
     entry Test ;
     entry Monitor ;
end Building_monitor ;

task body Building_monitor is
    type ROOMS is array (NATURAL range <>) of ROOM_NUMBER ;
    Move_sensor, Window_sensor, Door_sensor : SENSOR ;
    Move_sensor_locations: ROOMS (0..Number-of_move_sensors-1) ;
    Window_sensor_locations: ROOMS (0.. Number_of_window_sensors -1) ;
    Corridor_sensor_locations : ROOMS (0..Number_of_corridor_sensors-1) ;
    Next_movement_sensor, Next_window_sensor,
    Next_door_sensor: NATURAL := 0;
begin
     select
         accept Initialize do
              -- code here to read sensor locations from a file and
              -- initialize all location arrays
          end Initialize ;
    or
         accept  Test  do
              -- code here to activate a sensor test routine
          end Test ;
```

Detailed design of the Building_monitor process, 1

```
or
    accept Monitor do
      -- the main processing loop
      loop
        -- TIMING: Each movement sensor twice/second
        Next_move_sensor :=
           Next_move_sensor + 1     mod Number_of_move_sensors ;
        -- rendezvous with Movement detector process
        Movement_detector.Interrogate (Move_sensor) ;
        if Move_sensor /= OK then
           Alarm_system.Initiate (Move_sensor_locations (Next_move_sensor)) ;
        end if ;
      --  TIMING: Each window sensor twice/second
        -- rendevous with Window sensor process
        Next_window_sensor :=
           Next_window_sensor + 1     mod Number_of_window_sensors ;
        Window_sensor.Interrogate (Window_sensor) ;
        if Window_sensor /= OK then
           Alarm_system.Initiate (Window_sensor_locations (Next_move_sensor)) ;
        end if ;
      --  TIMING: Each door sensor twice/second
        -- rendevous with Door sensor process
        -- Comparable code here
      end loop ;
    end Monitor ;
end Building_monitor ;
```

Detailed design of the Building_monitor process, 2