ES Software Engineering

Lecture 4

State Machine Diagrams

In the previous lecture

1. Designing software using UML
2. UML diagrams
3. Class model – misunderstanding
4. Class diagram
5. Relation types in UML class diagrams

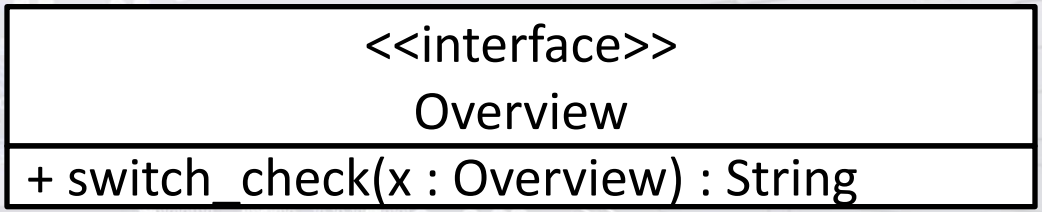
Plan of the lecture

1. Interfaces
2. Package diagrams
3. State machine diagrams
   - States, transitions
   - Notation in UML
   - Concurrent states, pseudo states
   - Examples

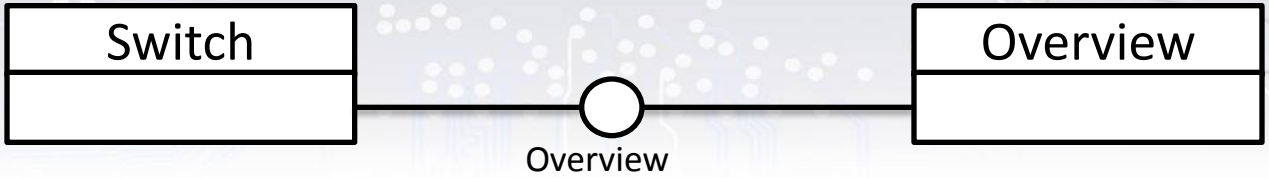
Interfaces

Interface – constitutes an abstract entry to classes. It contains declarations of attributes and methods but does not implement them.

- Notation in a form of a class:



- or:



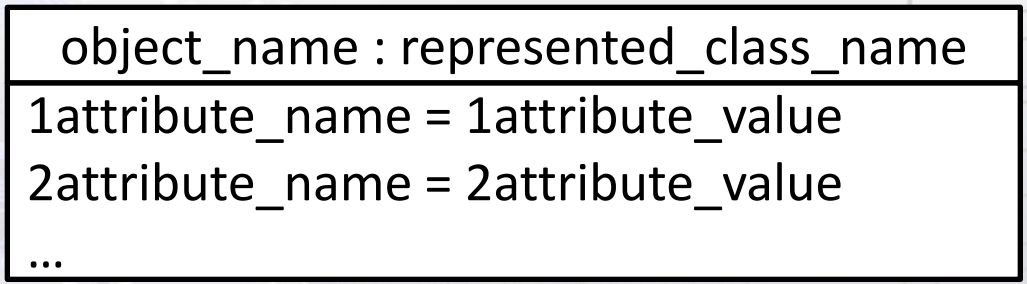
Object diagram

Object diagram (class instance diagram)

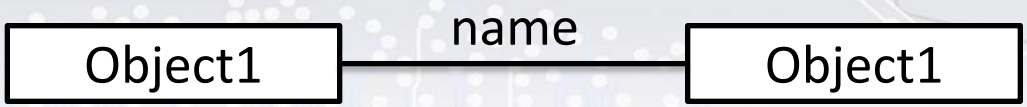
- is a functional extension of the class diagram
- used for modeling examples for understanding class diagrams
- contains actual case prototypes
- contains copies of elements of the class diagram
- the notation is simplified comparing to class diagrams
- is object-oriented and not relations between classes-oriented

Object diagram

Object diagram notation:



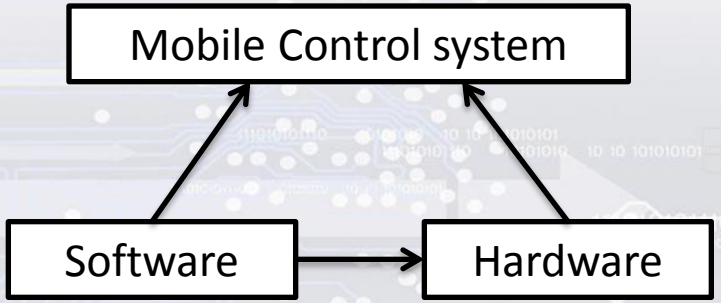
Association:



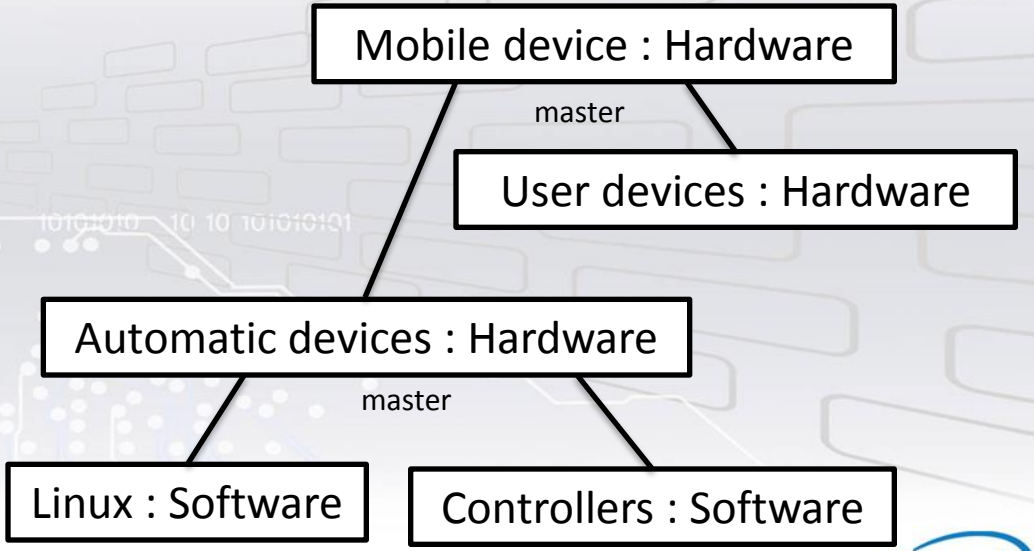
Object diagram

Example:

Class diagram:



Object diagram:



Package diagram

Growth of the modeled system imposes the need to simplify the model. Modeled entities can be grouped into packages:

- a package contains model elements similar in meaning (classes, interfaces, components)
- packages are loosely linked
- package elements are highly consistent inside a package
- package elements can be: classes (as a rule) but also interfaces, operations, diagrams, other packages etc.

Package diagram

Packages – when to use:

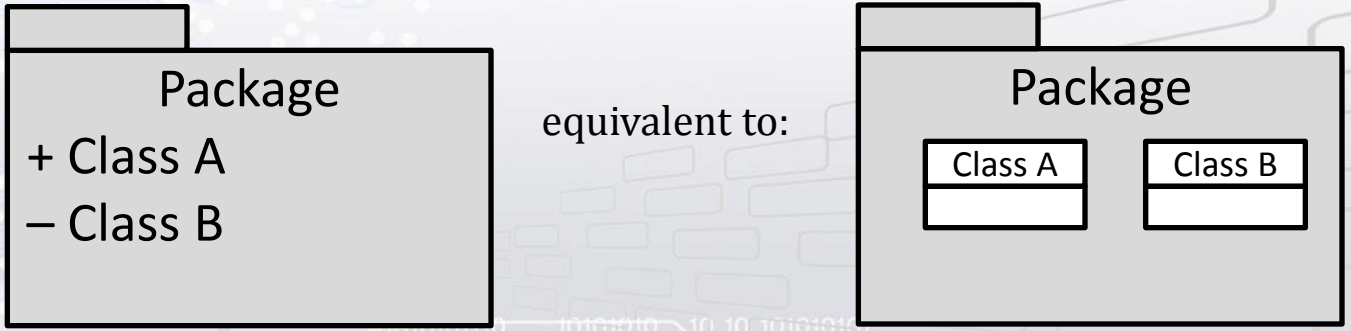
- In order to implement the tree structure of model elements
- For evaluating the quality of the model and coherence of links inside the model
- For designing systems consisting of systems
- For designing systems intended for continuous expansion
- A package introduces model encapsulation – reference to package elements is done using a full path i.e. containing the name of the package

When not to use?

- If the model contains circular dependencies

Package diagram

Package – graphical representation:



Notation:

package::Class A

package::Class B

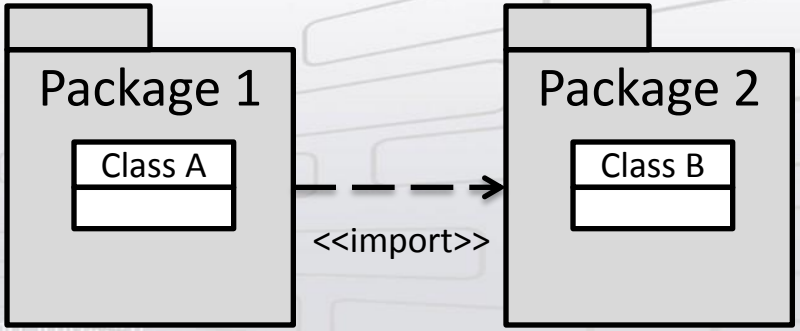
+ public

– private

Package diagram

Access to package elements is possible by different types of dependencies:

- <<import>> provides access in class A to elements of class B. Package 1 can use class B without giving a path to package 2. Subsequent import also provides access to class B (equivalent to public inheritance)
- <<access>> similarly to the import but the next access does not provide access to class B (equivalent to private inheritance)
- <<merge>> enables connecting class B to Package 1. Rarely used and non-functional.



State machine diagram

State machine diagram – diagram consisting of elements representing object states. Events appearing in a system activate transitions between states and the machine itself verifies conditions fulfilling of which is necessary to perform a specific transition.

State machine diagram

State machine elements:

- **state –** moment in the life cycle of an object, in which certain conditions are met, the object performs certain actions and expects events
- **event –** an atomic process or activity which may influence the object state
- **action –** an atomic operation execution
- **activity –** operation executed by an object in some state until it is interrupted by an event

State machine diagram

State machine elements:

- **state transistion** – relationship between two states of an object. Object may change a state in reaction to an event and if some conditions are fulfilled.
- **state machine** – abstract way of describing the behavior of system objects, their interactions in case of specific events. A graphical interpretation of a state machine is the state transistion diagram.

State machine diagram

State machines are divided into:

- Protocol state machines

Describe what is happening with a selected object, i.e. they depict transitions between object states.

- Behavior state machines

Describe transitions between states of many objects and explain how they affect the status of the system.

State machine diagram

Protocol state machine

- Not all transitions are included in the machine but only those which cause transitions of an object into another state. Similarly, the machine does not contain operations declared in classes which do not change object state.
- Machine description mostly contains of a series of states and transitions between states relating to a given object.

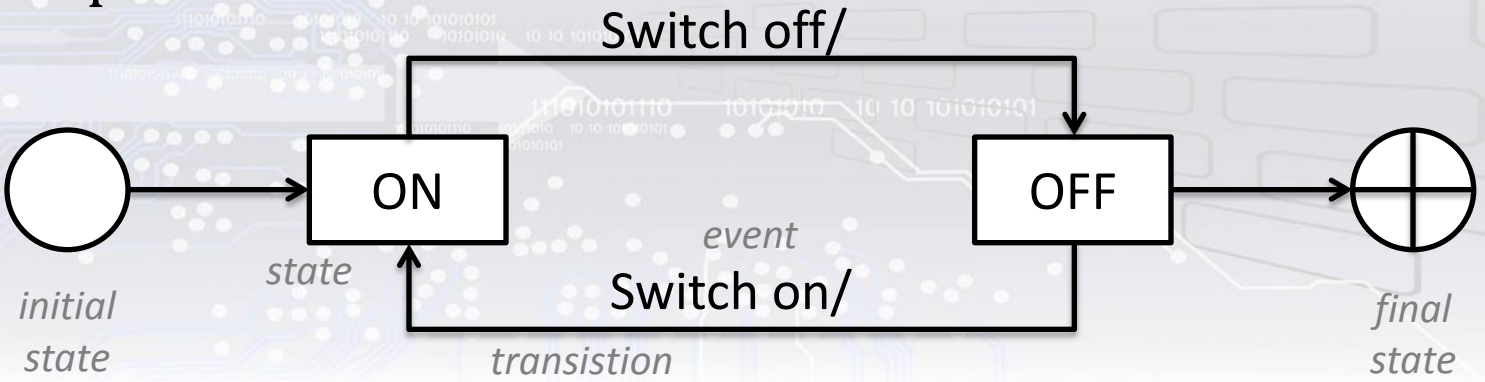
State machine diagram

Protocol state machine

Notation:

[initial_condition] operation_name/ [end_condition]

Example:



State machine diagram

Behavior state machine

Describes transitions between object states in context of the behavior of the system. Therefore the machine takes into account the concept of an event, a condition and actions.

Notation:

[event] [guard condition][/action]

State machine diagram

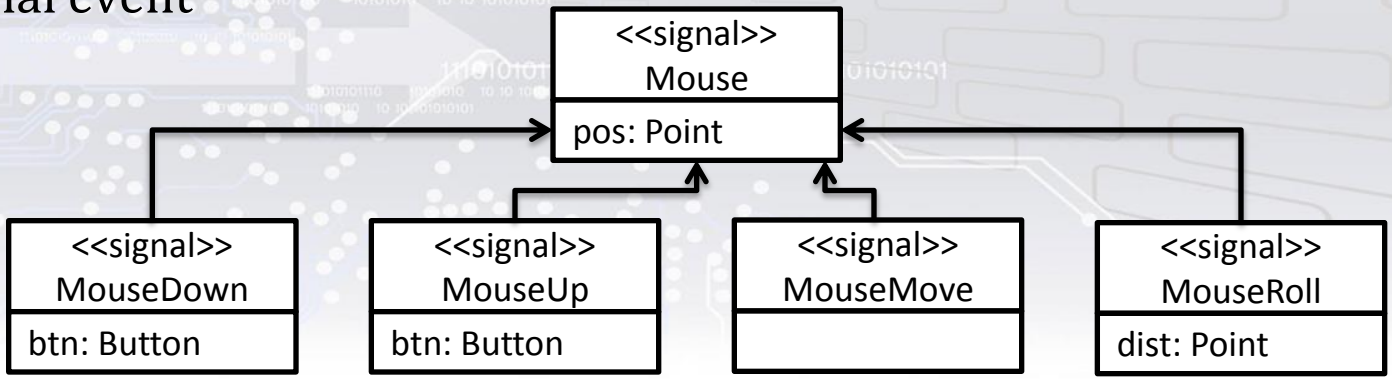
Kinds of events: [event] [guard condition][/action]

- Call event

Occurring when an operation should be performed

operation_name (parameter list)

- Signal event



State machine diagram

Kinds of events:

[event] [guard condition][/action]

•Change event

Occurring if a condition is met:

when condition_expression

•Time event occurring:

- after some time interval
  after time_expression
- in some time moment
  when time_expression

State machine diagram

Guard condition: [event] [guard condition][/action]

- Boolean expression containing:
  - triggered event parameters
  - object properties and states
  - when an object is in some state("in" *state name*)
- Guard condition is checked "after" triggering an event, and "before" firing a state transition

State machine diagram

Action: [event] [[condition]][/action]

- A sequence of operation calls sequentially executed
- Operations are separated with semicolons
- Operation arguments can use triggered event parameters

State machine diagram

State diagram – a graph corresponding to a state machine

- refers to an object
- describes the behavior of an object at a finite number of states and transitions between them
- diagram contains states connected by transition arrows
- transitions between states is triggered by asynchronous events

State machine diagram

State diagram (states+transitions)

State – moment in the life cycle of an object in which a certain condition is met. Each state has its own name. Notation:

[event] [[condition]][/action]

describes one state.

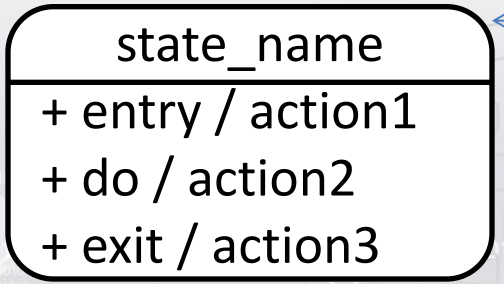
Every state, beside its name, also contains actions describing going into and out of the given state:

action-label / action

State machine diagram

State diagram (states+transitions)

State – graphical representation:



i.e.

entry / print ‘action in progress’ / begin action
exit / end action / print ‘done’

State machine diagram

State diagram (states+transitions)

State – actions:

- **entry** – action performed once, when the object adopts to a given state
- **do** – action still performed when the object remains in a given state
- **event** – action taken with the moment of a specific event occurring
- **include** – action of calling a nested state machine
- **exit** – action performed once, when the object leaves a given state

State machine diagram

State diagram (states+transitions)

State – properties:

- each action may call a new event
- number of states is unlimited for each object
- number of states in a diagram can be very large and equals:

number_of_machines * number_of_objects * number_of_states_of_each_object

- a state can be understood as a time period in which an object performs actions or waits for an event

State machine diagram

State diagram (states+transitions)

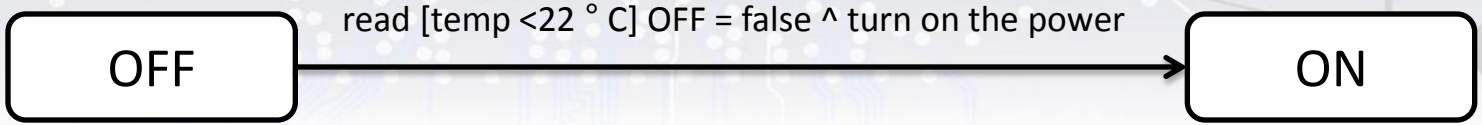
Transition

Transitions connect states together. They define events and conditions which must be met so that an object can change its state.

Notation:

trigger [guard condition] / action ^ event

Example:



State machine diagram

State diagram (states+transitions)

Transition

trigger [guard condition] / action ^ event

- **trigger –** event which may cause a transition, as long as conditions are met
- **guard condition** – conditions which must be met for a transition to occur
- **action –** operations carried out undivided during a transition (refers to the diagram – e.g. "Disable state")
- **event –** event which is sent during a transition (refers to the system – e.g. “Turn off heater")

State machine diagram

Complex states

In the simplest case the possibility of existence of substates is ignored. However, in practice, an object may often appear in a number of substates, meaning it has its own state machine.

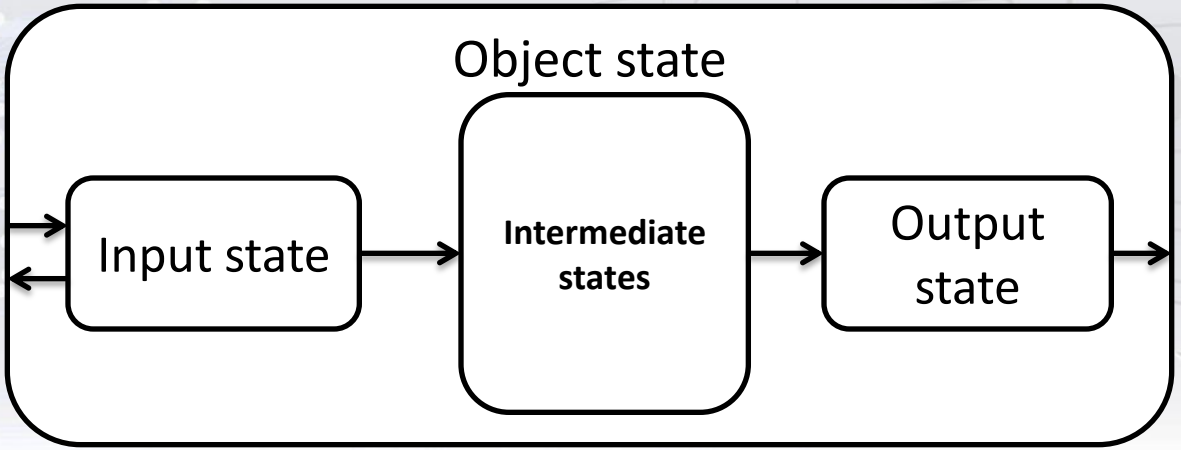
It is the case, for example, in case of objects from databases, state of which is modified from the outside by a number of customers and the objects themselves change their internal state with each such operation.

State machine diagram

Complex states

Input state is the initial state

Output state is the final state

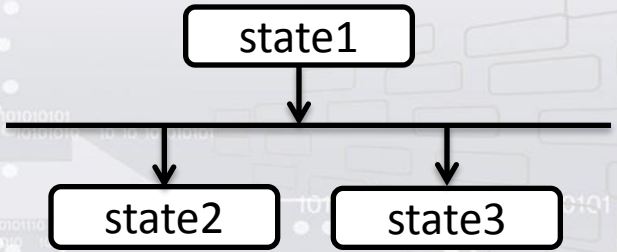


State machine diagram

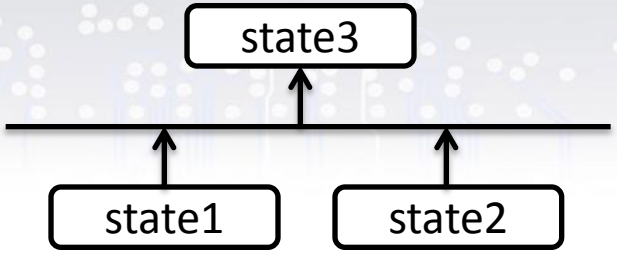
Pseudo states

Have their source in the complexity of transitions, which in general do not need to be *"one state to one state"* but:

- *fork*:



- *join*:

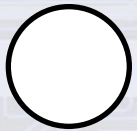


State machine diagram

Pseudo states

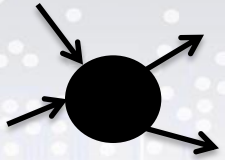
- are abstract entities in the state machine diagram
- are more than transitions but not states

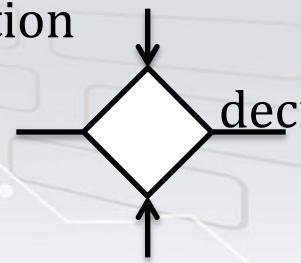
Examples:


initial state


end state


point of destruction

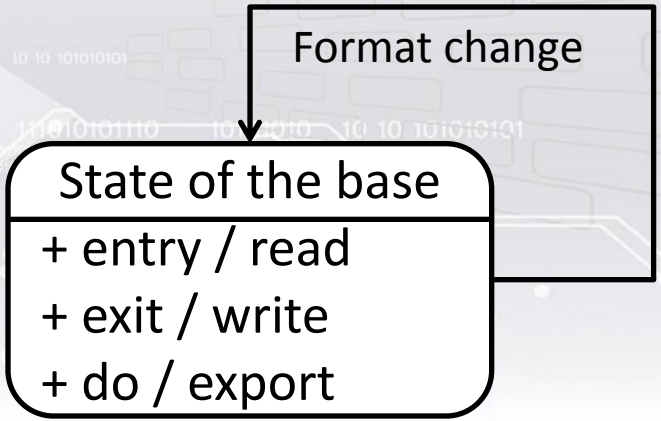

node



State machine diagram

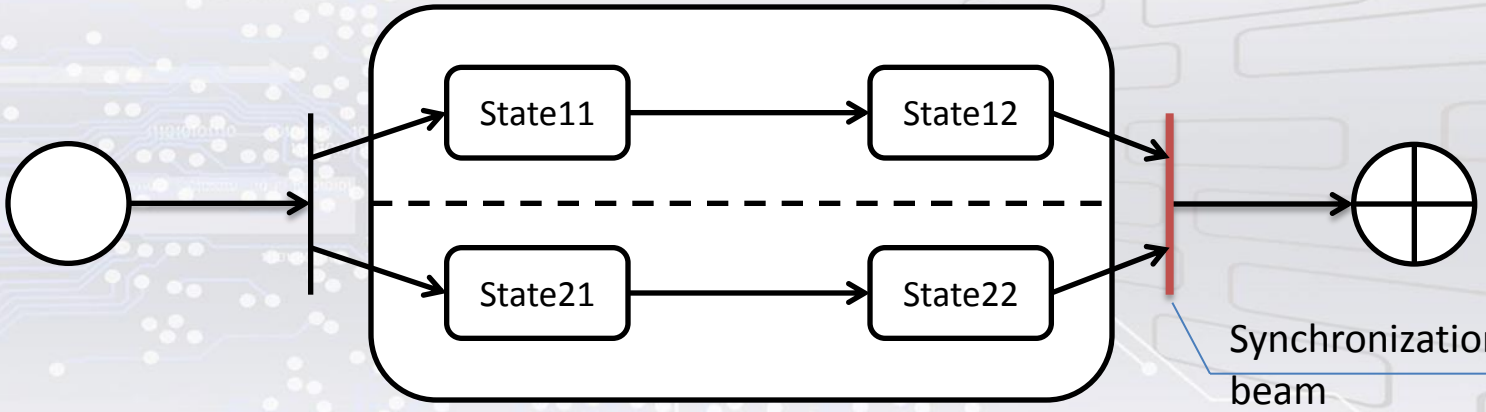
Internal transition compartment

- State does not change
- It is not a reverse transition (entry and exit operations are not performed)



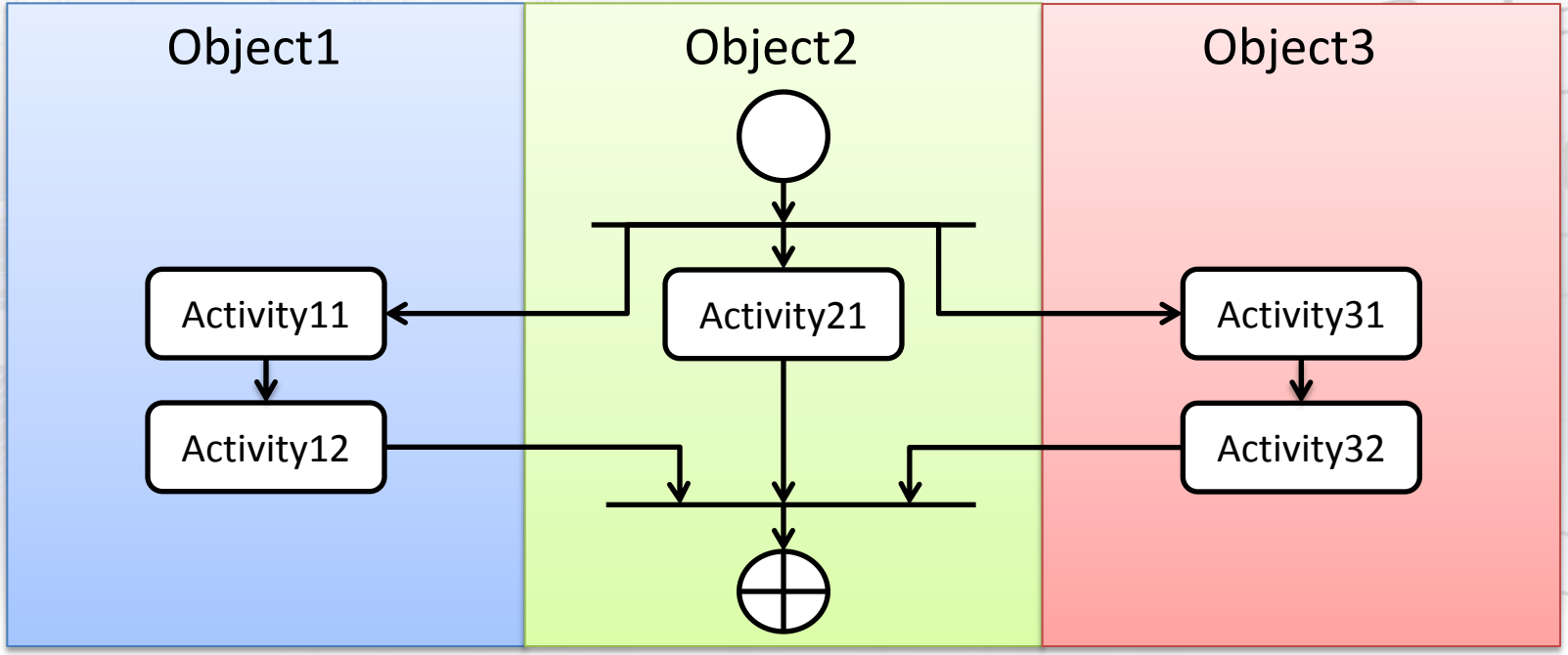
State machine diagram

Parallel states



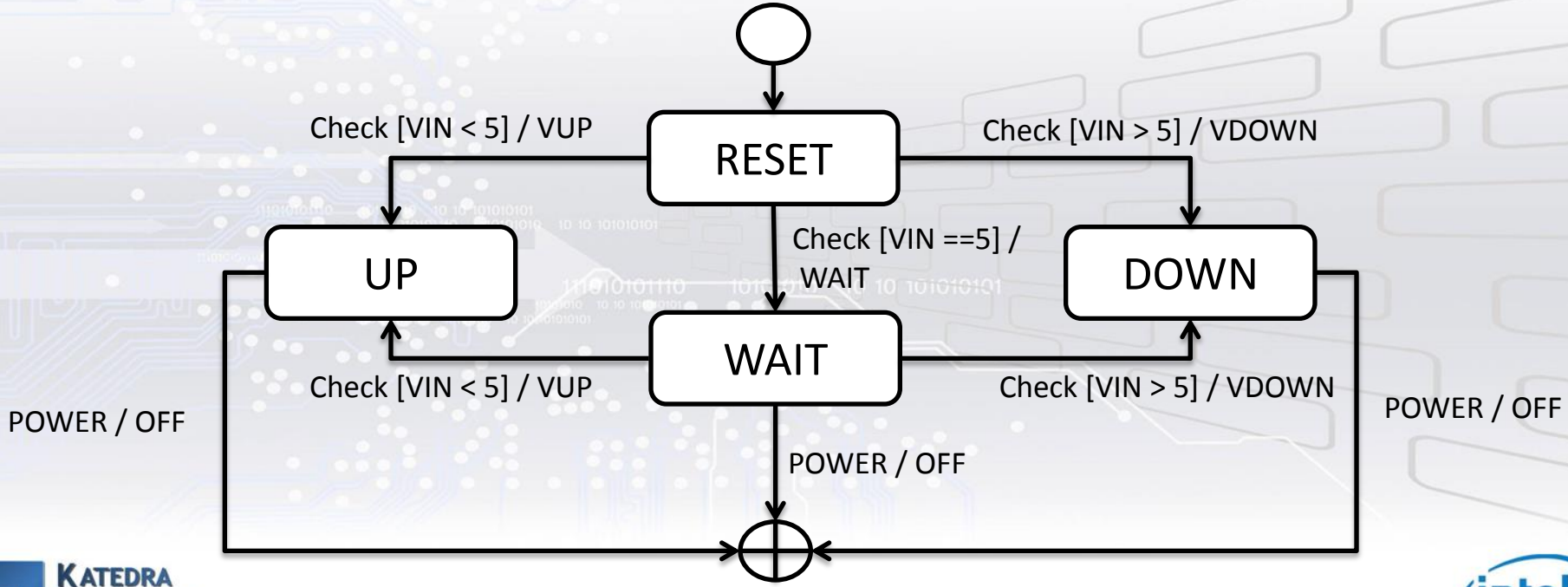
State machine diagram

Concurrent activities



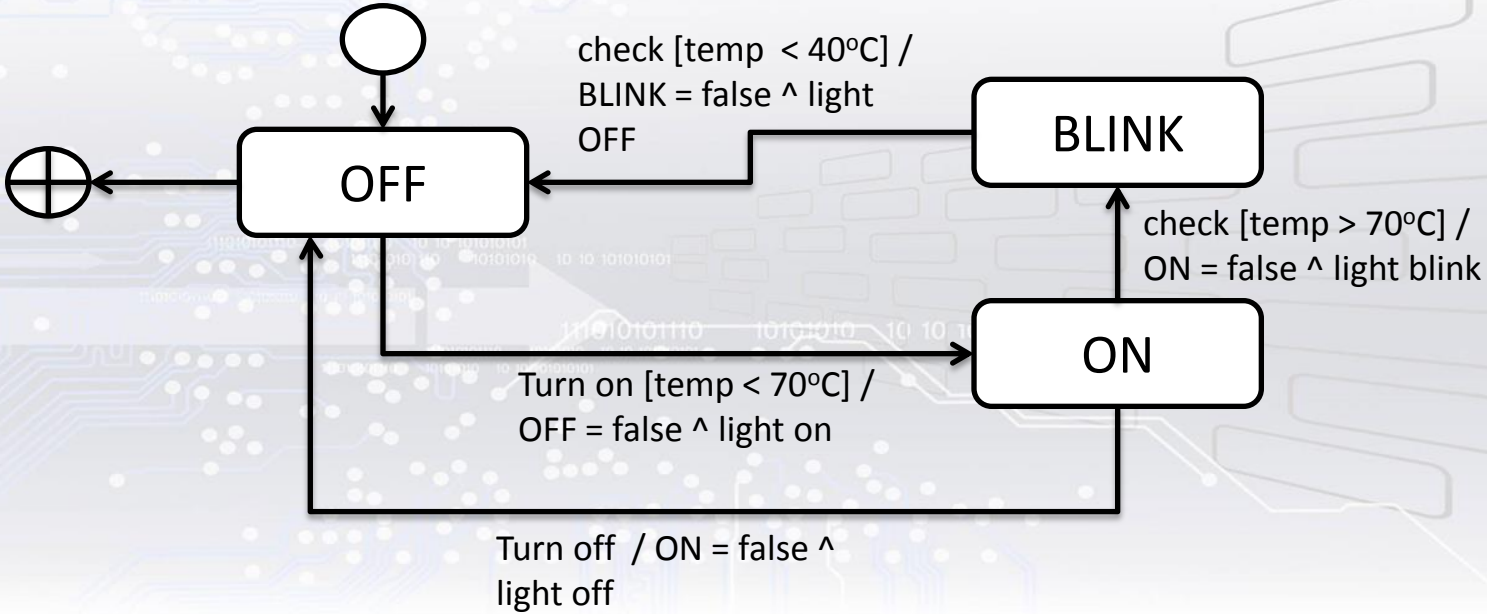
State machine diagram

Example state machine (voltage stabilizer)



State machine diagram

Example state machine (vacuum cleaner with a thermostat)



State machine diagram

How to start creating a state machine:

- Choosing a state machine (protocol or behavior)
- Defining states
- Ordering states and substates according to a specific hierarchy
- Connecting states with transitions
- Defining pseudo states based on transitions

Bibliography

[1] Grady Booch, James Rumbaugh, Ivar Jacobson, *Unified Modeling User Guide* (book, pdf)

[2] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-klas%2c1%2c11.html

[3] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-pakietow%2c1%2c18.html

[4] http://brasil.cel.agh.edu.pl/~09sbfraczek/diagram-maszyny-stanowej%2c1%2c19.html

[5] Jarosław Kuchta, *State transistion modeling*, Embedded Systems Software Engineering, 2015