Based on CREATE material from Itemis.

Extended from a version by Arjan Mooij

Finite-State Machines (FSM)

Software Systems (Computer & Embedded Systems Engineering)

Rosilde Corvino

January 2024 (week 8)

TU Delft Delft University of Technology

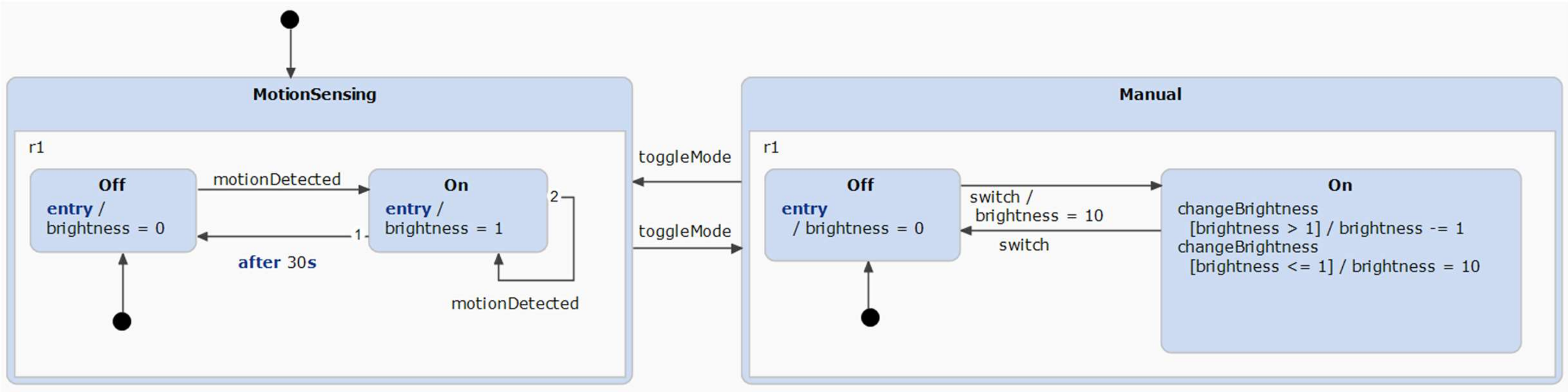
ESI

An initiative of industry, academia and TNO

Notation and simulation

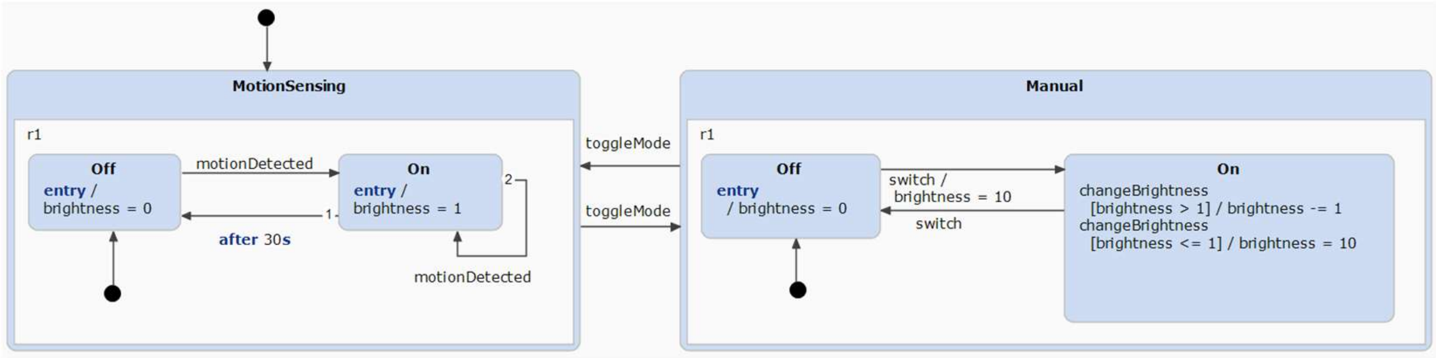
Finite-State Machines (FSM)

What would the elements in this Finite-State Machine mean?



Think → Pair → Share

Composite states

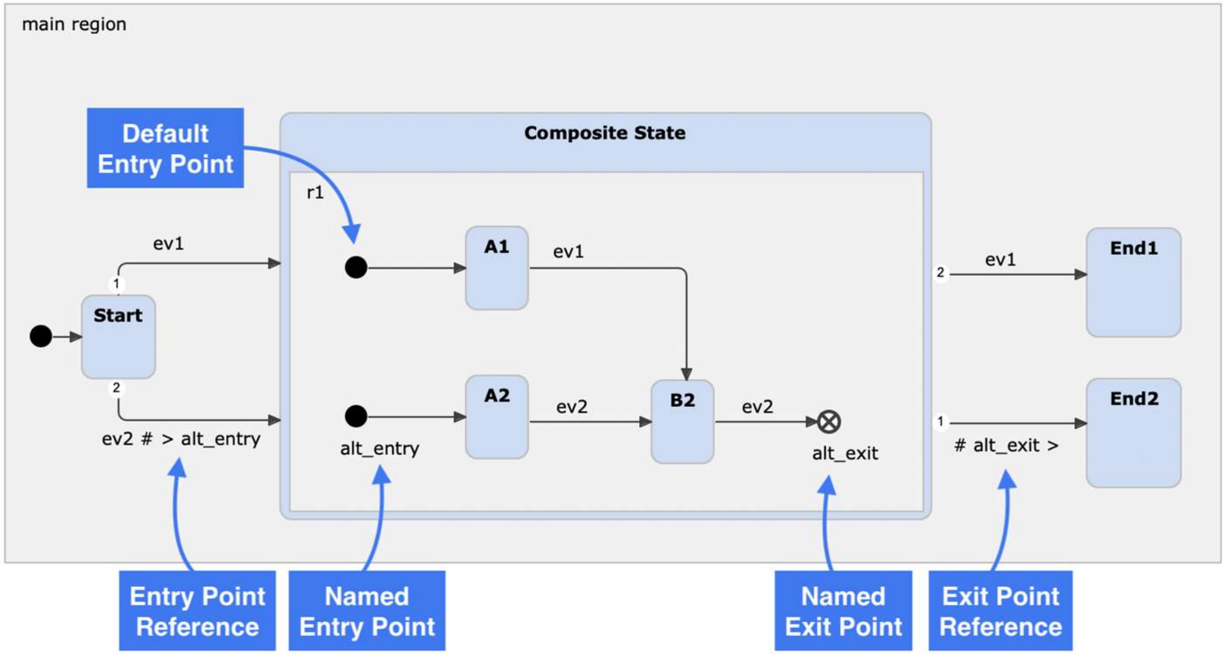


- **Composite state**
  - State that contains one or more substates
  - Outgoing transitions from a state apply to all its substates
  - Transitions can point to either a state or a substate
  - Whenever entering a composite state, the entry node is activated

Simulation

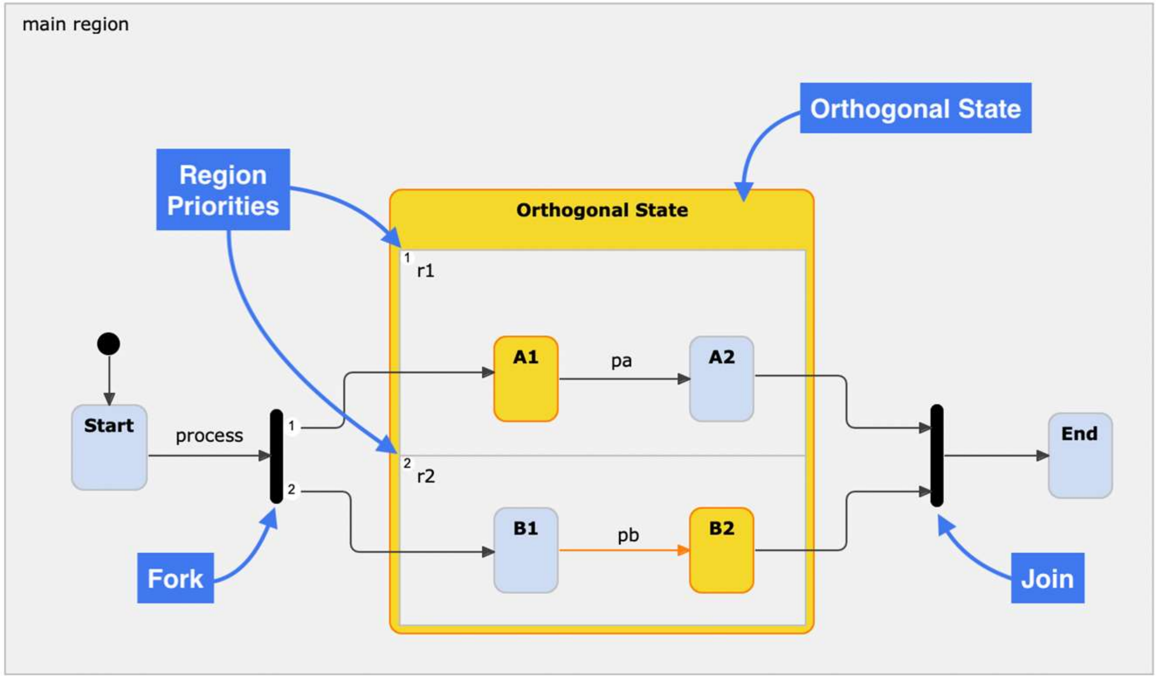
Live demo

Multiple entries and History nodes



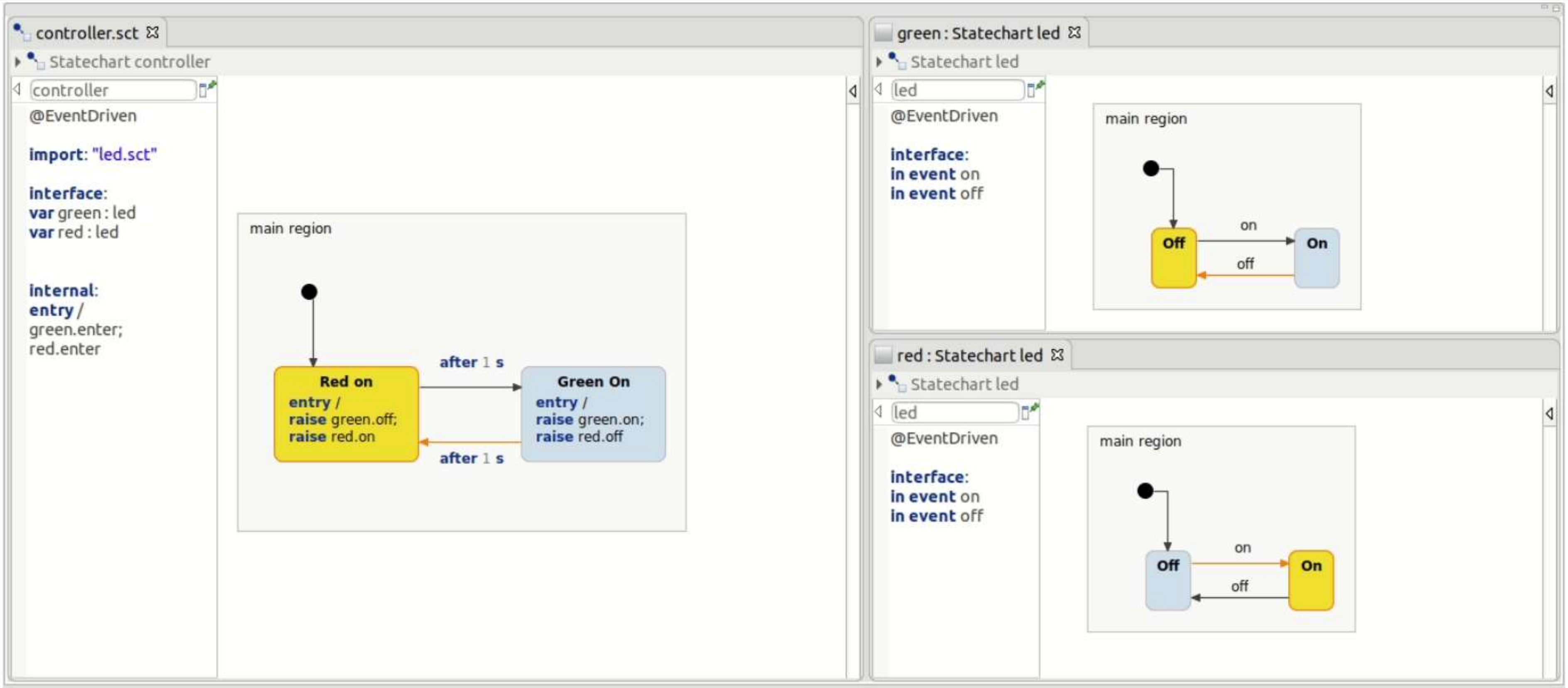
- **Composite state:**
  - Possibly multiple entry nodes (with unique names)
  - Possibly multiple exit nodes
- **History nodes:**
  - (Default: don't remember the state that was active when the composite state was left)
  - Shallow: remember the state that was active when the composite state was left
  - Deep: remember all nested states when the composite state was left

Orthogonal states



- **Orthogonal state**
  - Surrounded by Fork and Join nodes
  - System can be in multiple states simultaneously
    - Transitions are executed sequentially
    - Orthogonal regions have priorities
  - Orthogonal regions can communicate via internal events

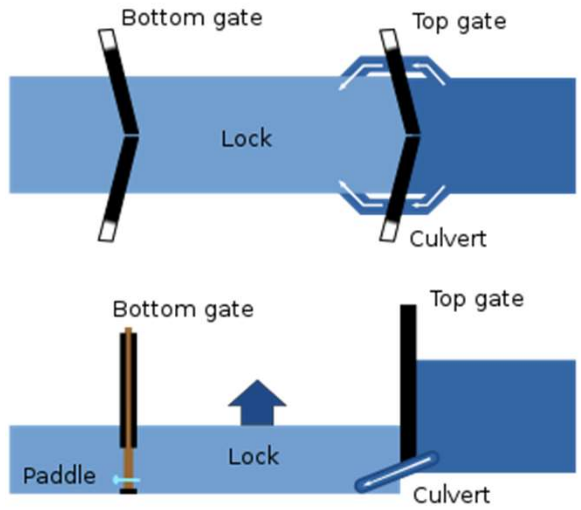
Multi state-machine modelling



Modeling skills

Finite-State Machines (FSM)

Creation of an FSM



https://en.wikipedia.org/wiki/Lock_(water_navigation)

Ship lock:

- Two gates: bottom and top
  - Each with two traffic lights: inside and outside the lock
- Two valves: bottom (paddle) and top (culvert)

Extra:

- Split the OpenGate state in two phases: leaving and entering
- Only operate the lock when needed (so boats need to announce themselves)
- Before exiting the lock, boats need to pay

Let's model the boat's behavior and its relation to the lock!

Creation of an FSM

1. **Model the behavior of the boat:** **Arrive; (Announce; Enter || Pay); Exit; Depart**
2. **Combine the model of the boat and the lock**
   - Internal events: Announce, Enter, Pay, Exit

Think → Pair → Share

Alternative

- **Another way of modeling**
  - Obtain state machine by projection from sequence diagrams
  - Include and discuss non-local choice (and verification)
    - E.g., spontaneous error situation that leads to a race condition

Application areas

Finite-State Machines (FSM)

How can we exploit Finite-State Machine models?

Now imagine:

- You are working on a Software System…
- And you have some Finite-State Machine models…
- And you notice the important information that these models contain…

How could we possibly use Finite-State Machine models to benefit even more from them?

Think → Pair → Share

How can we exploit Finite-State Machine models?

- **Communication**
  - Customers — Model of the specification/requirements
  - Developers — Model of the implementation
- **Generation of implementation code in various languages**
- **Validation and verification of the model**
  - Model simulation — explore a specification model
  - Model testing — check whether the model contains particular execution traces
  - Formal verification — check model properties (e.g., safety, liveness, fairness)
    - E.g., mCRL2 (https://www.mcrl2.org/) and UPPAAL (https://uppaal.org/)
- **Validation and verification of an implementation with respect to the specification**
  - Model-based testing — deriving tests from the model and some test requirements
    - E.g., TorXakis (https://torxakis.org/) and Axini (https://www.axini.com/)
  - Run-time monitoring — observing the system during any execution

How to model interfaces?

- So far we have modeled the internal behavior of components…
- But we could also apply modeling to external interfaces of components...

Why would this be useful in a system consisting of multiple components?

What would you like to model of a software interface?

Think → Pair → Share

Modeling component interfaces

The integration of components is known to be challenging

- Also after “minor” changes in components
- Concurrency, asynchronous execution

Aspects of interfaces:

- Signature: methods, messages, parameter types, etc.
- Behavior: order of allowed/expected messages or method calls
- Timing: expected response times, supported frequency
- Data format: how to encode the data, valid data ranges

Some approaches:

- Formal verification: Verum ASD/Dezyne https://verum.com/
  - Gives formal guarantees that certain properties are established
- Runtime monitoring: Eclipse CommaSuite https://www.eclipse.org/comma/
  - Helps to detect and diagnose issues at run-time

Closing remarks

Finite-State Machines (FSM)

Objectives

At the end of the course, you should be able to:

- Explain the purpose of Finite-State Machines, including several application areas
- Explain the concepts and notations of Finite-State Machines
- Create basic Finite-State Machines to model software-intensive systems

Assessment:

- Modeling assignment using Finite-State Machines (in groups of 2 students)
- Reflection document on Model-Based Development (individual)

Closing remarks

- **Lab session work on "Notation and simulation" before the next lecture**
  - Download and install the Itemis CREATE
  - Eclipse → Help → Help Contents → Itemis CREATE documentation
    - Tutorials → Comprehensive tutorial
- **Optional (if you are interested in test-driven development):**
  - Eclipse → Help → Help Contents → Itemis CREATE documentation
    - User guide → Testing state machines
      - **NOTE: Up to section 1.3 only! So stop before "1.4 The SCTUnit language"**
- **Lab session work:**
  - Modeling assignment
- **Think about the relation between FSM and UML use case diagram**