State Machine Based Design for Embedded Systems

Katie Elliott

April 22, 2019

elliottke@gmail.com

Topics

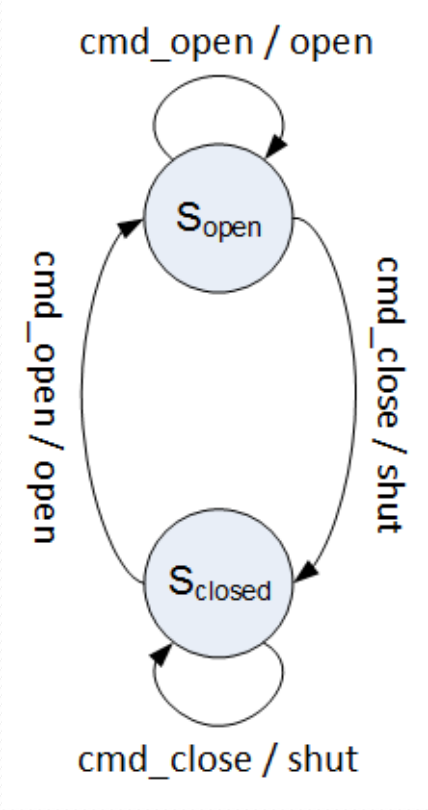
- What is a state machine?
  - Mealy, Moore, and UML state machines
- Unified Modeling Language (UML) state machine concepts
  - States
  - Events
  - Actions and transitions
  - Run-to-completion execution
- Creating clean designs with state machines
- Using the Quantum Platform (QP) with state machines
- Detailed design of a washing machine
- Break
- Demo/Exercise

Topics

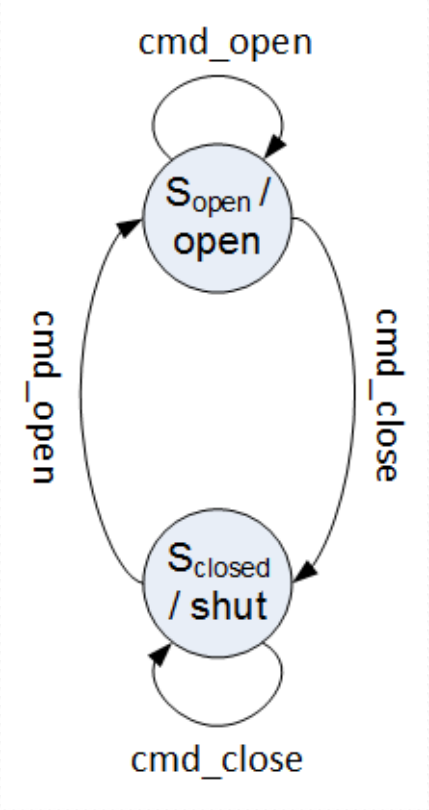
- What is a state machine?
  - Mealy, Moore, and UML state machines
- Unified Modeling Language (UML) state machine concepts
  - States
  - Events
  - Actions and transitions
  - Run-to-completion execution
- Creating clean designs with state machines
- Using the Quantum Platform (QP) with state machines
- Detailed design of a washing machine
- Break
- Demo/Exercise

Old-school state machines

Mealy state machine



Moore state machine

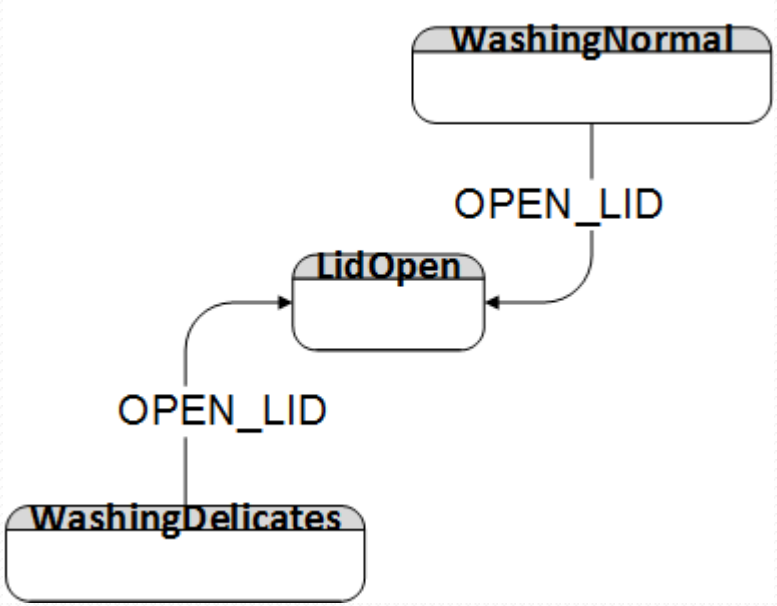


UML state machines

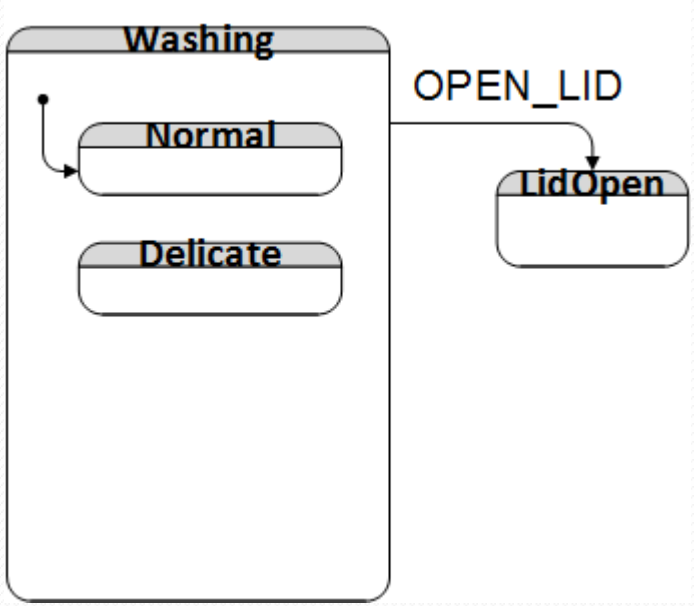
- Modified Harel statechart which incorporates aspects of both Moore and Mealy machines. [4]
- Support actions based on event as well as state entry and exit.
- Includes the concept of Hierarchical State Machines (HSMs).

UML state machines

Finite State Machine (FSM)

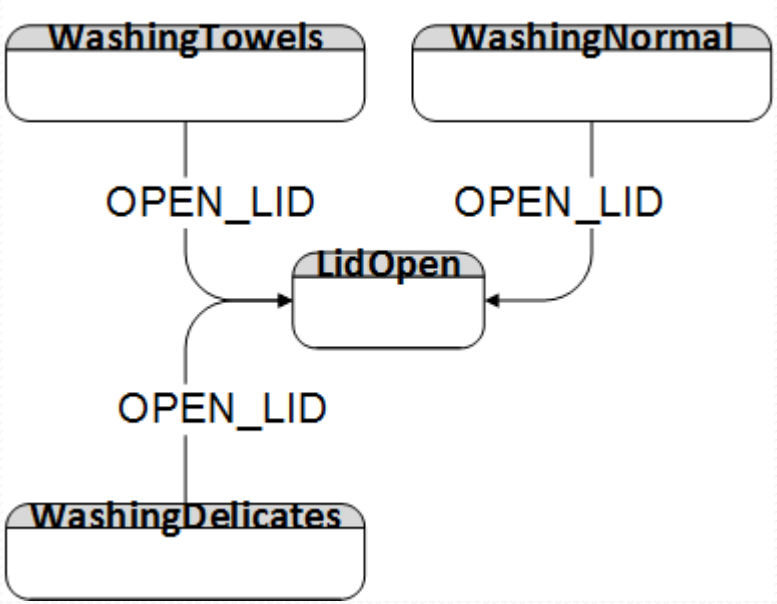


Hierarchical State Machine (HSM)

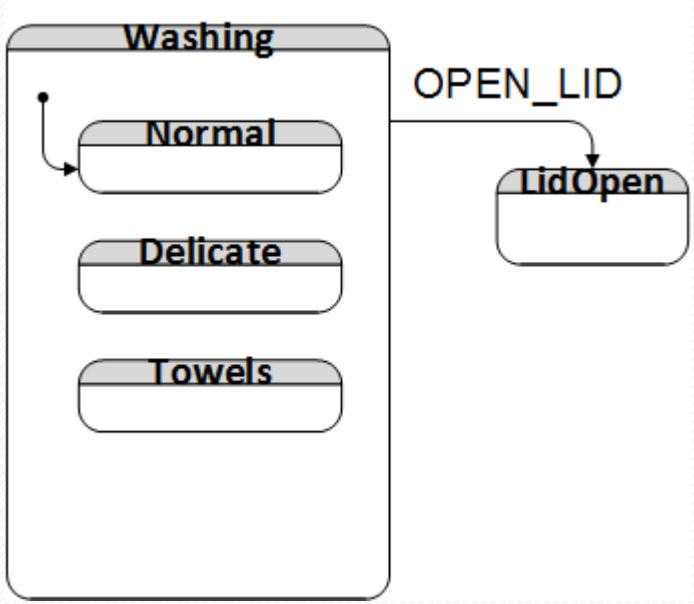


UML state machines

Finite State Machine (FSM)

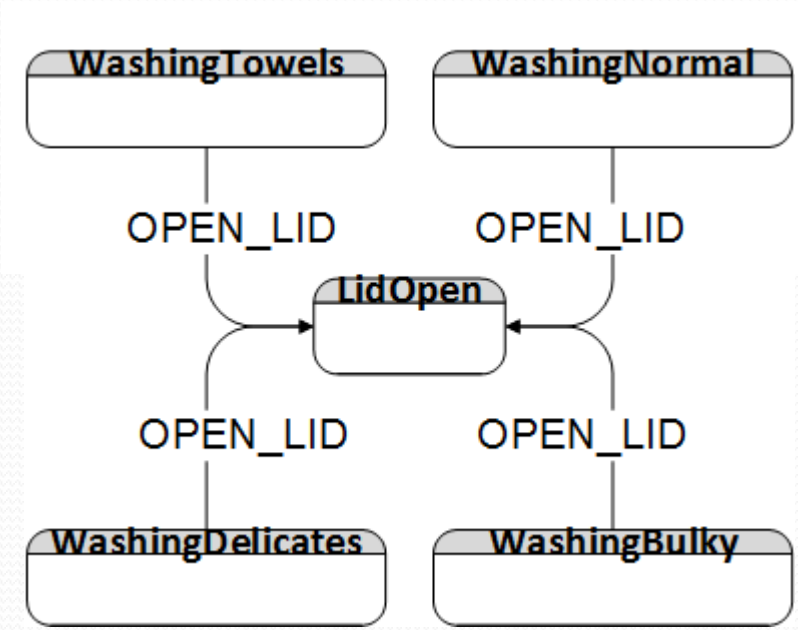


Hierarchical State Machine (HSM)

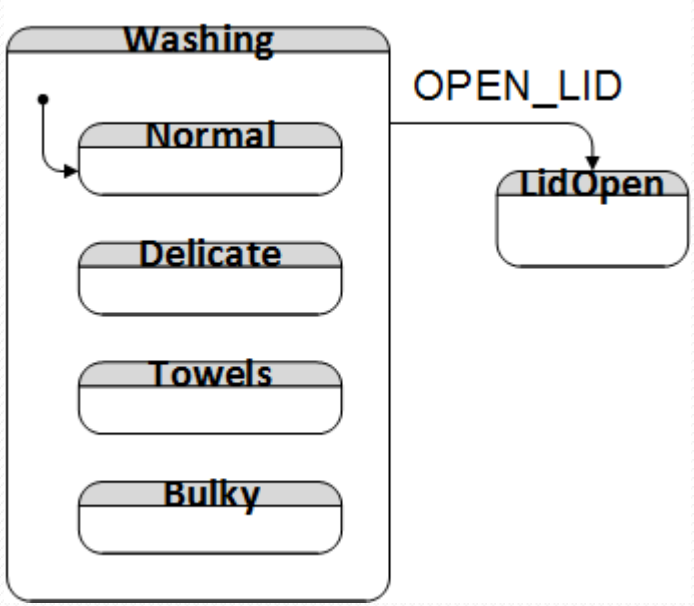


UML state machines

Finite State Machine (FSM)

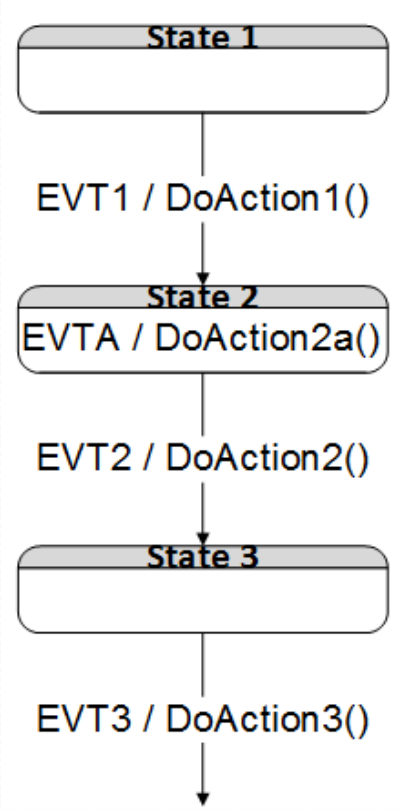


Hierarchical State Machine (HSM)

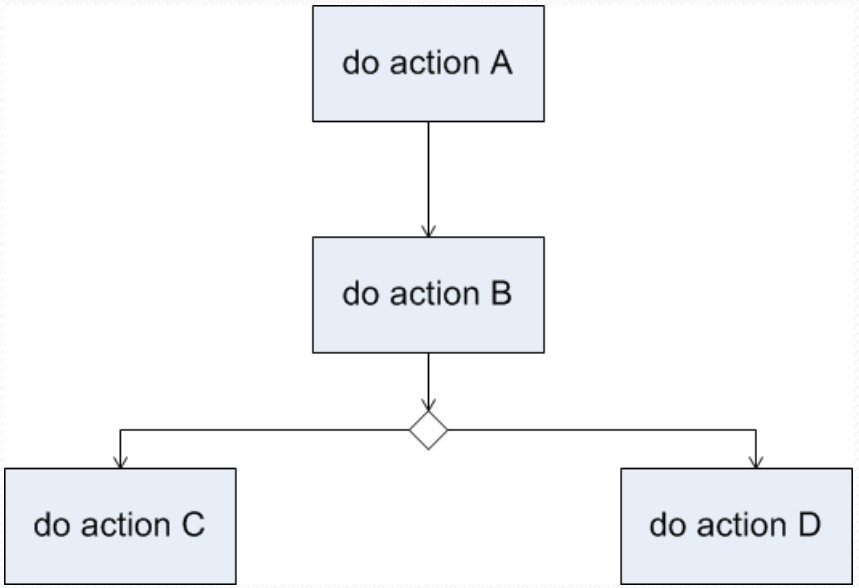


State Diagrams Versus Flow Charts

UML state chart



Flow chart



Topics

- What is a state machine?
  - Mealy, Moore, and UML state machines
- Unified Modeling Language (UML) state machine concepts
  - States
  - Events
  - Actions and transitions
  - Run-to-completion execution
- Creating clean designs with state machines
- Using the Quantum Platform (QP) with state machines
- Detailed design of a washing machine
- Break
- Demo/Exercise

UML statecharts

- A **state** (rounded boxes) represents the context and history of the system.

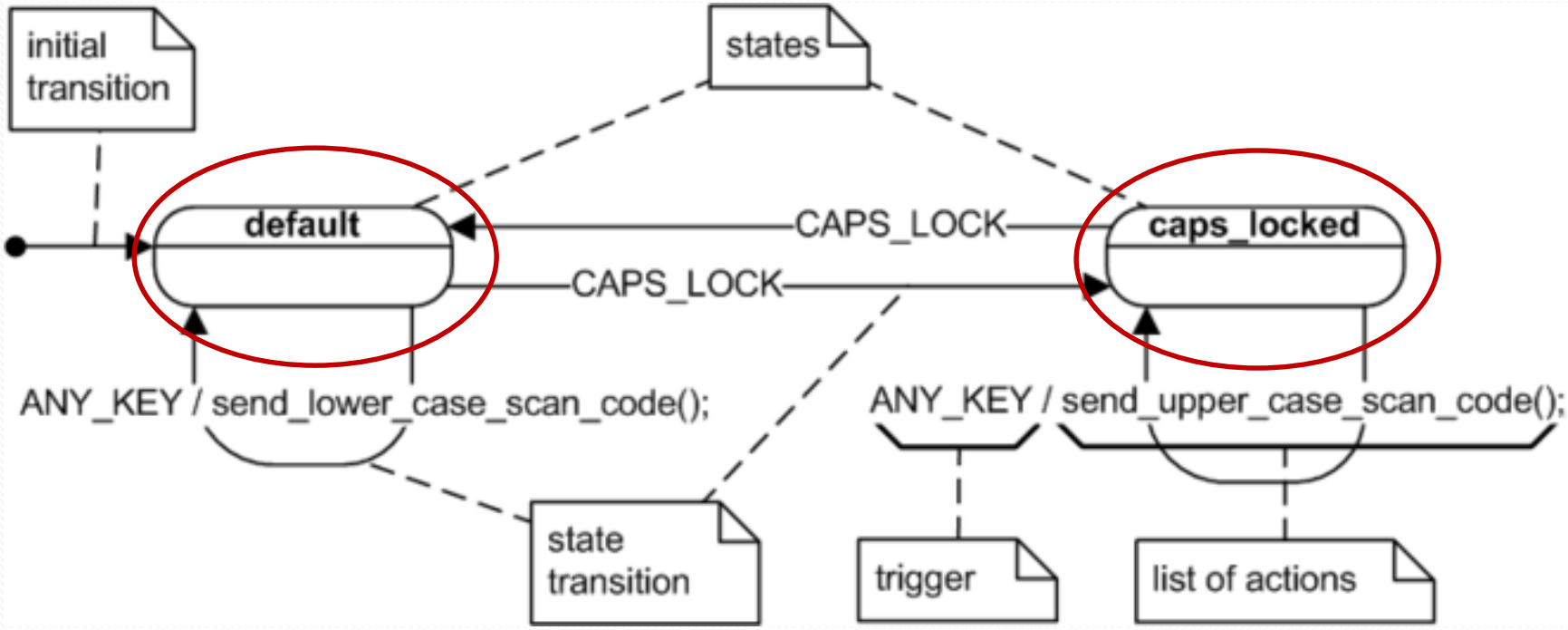


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

UML statecharts

- An **event** (all caps text) is something that happened in the system.

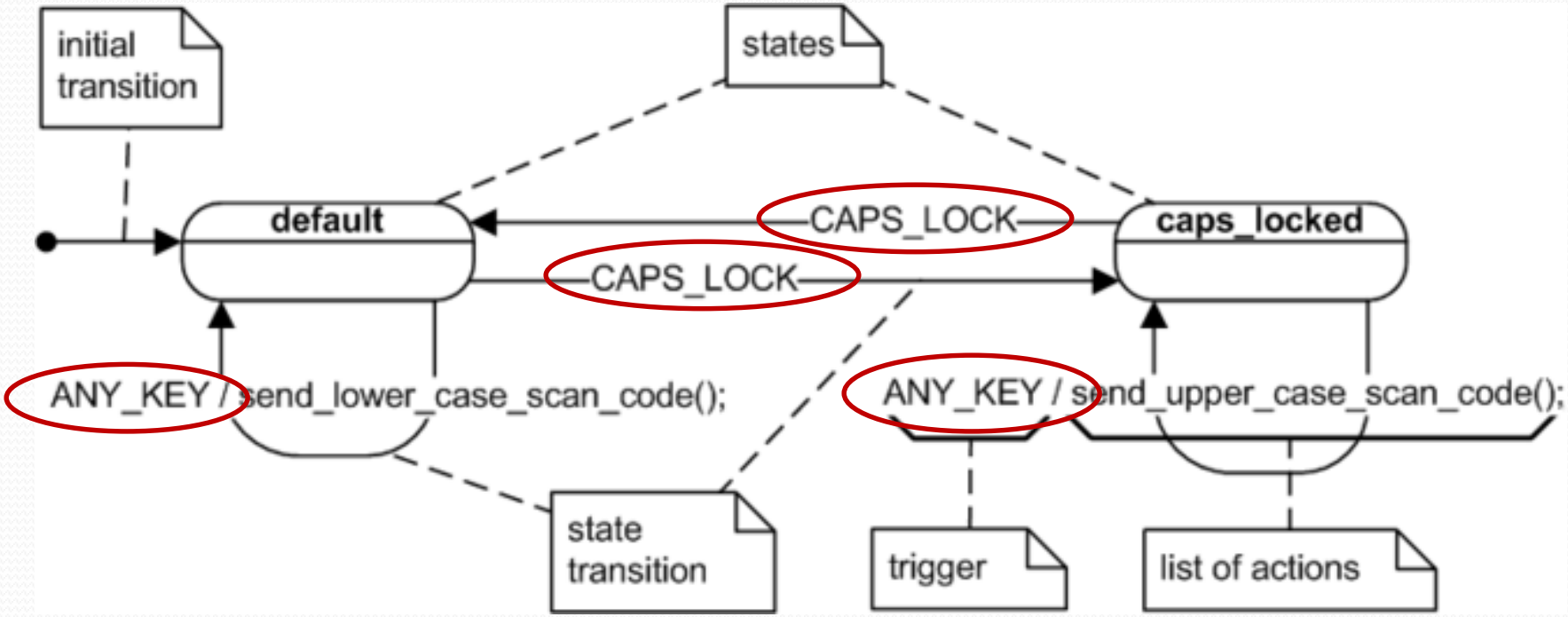


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

UML statecharts

- Event **actions** are optionally included after the event.

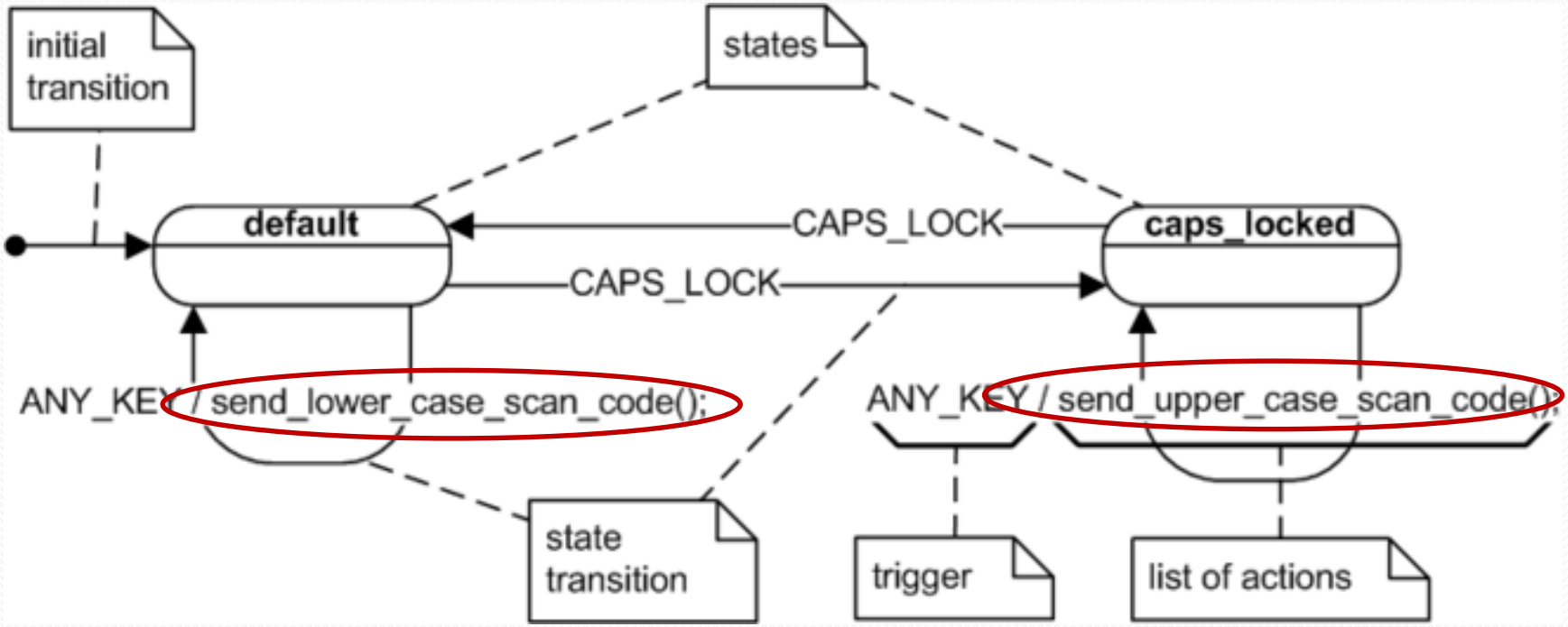


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

UML statecharts

- **Transitions** (arrows) are how we move between states.

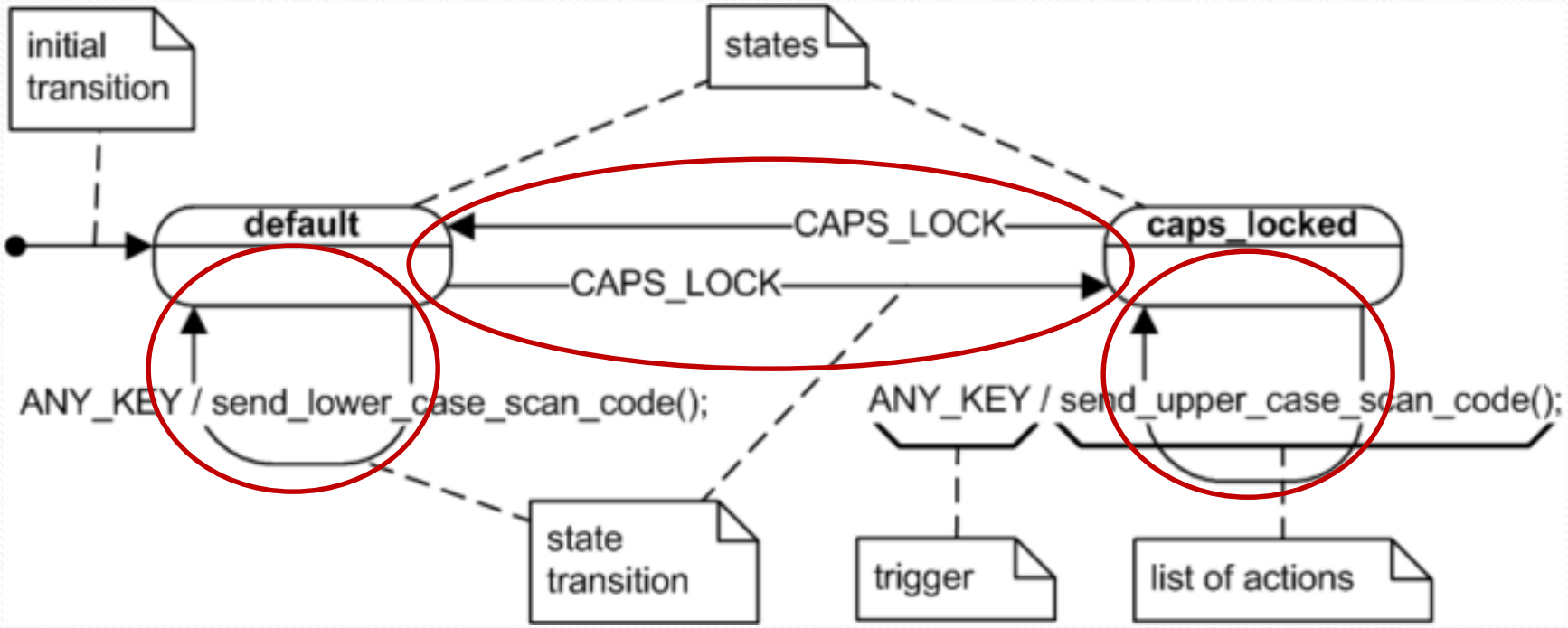


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

UML statecharts

- An **initial transition** (arrow with a dot) indicates which state is the starting point of the state machine.

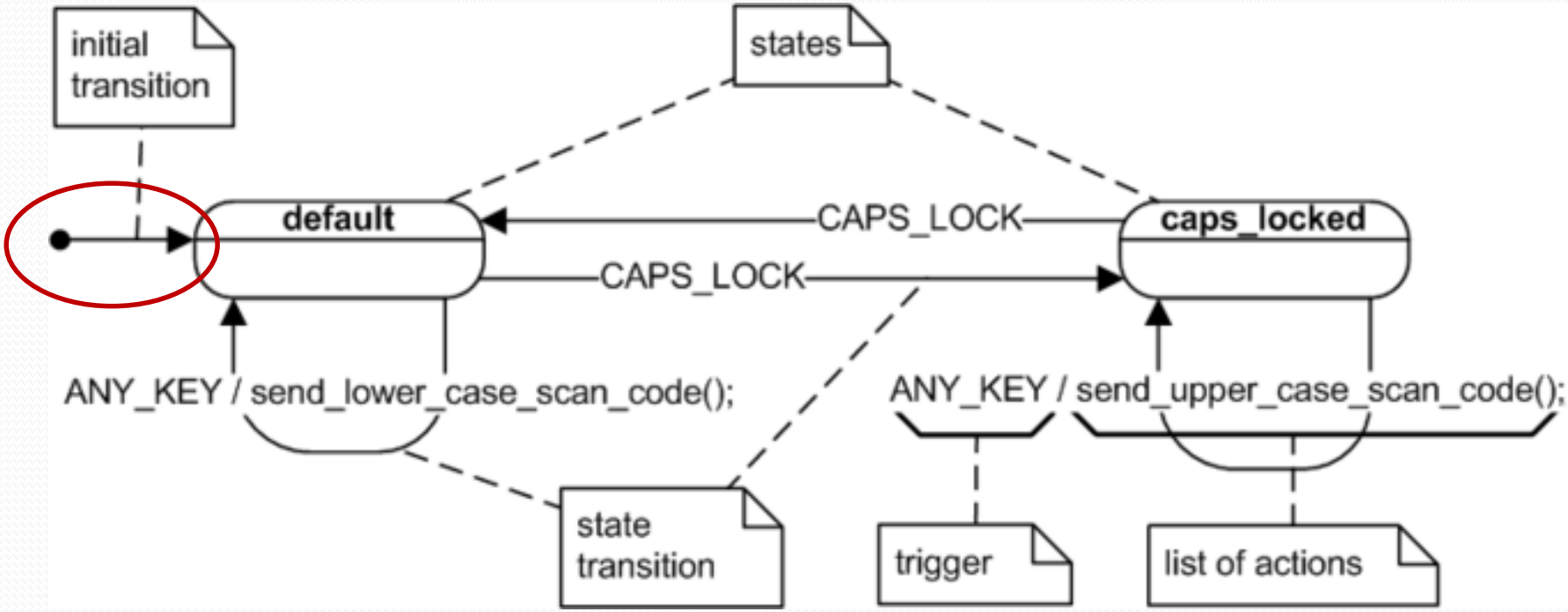
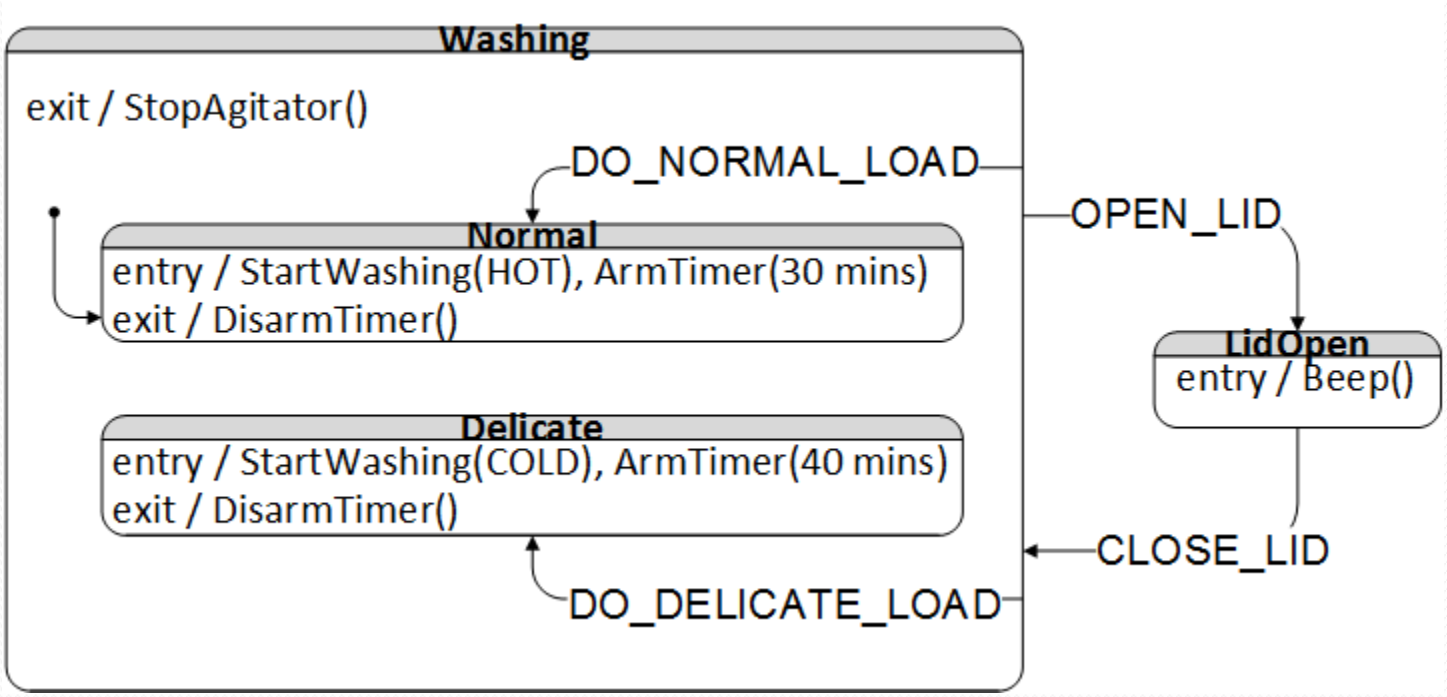


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

Entry and Exit Actions

- Associated with the entry and exit of the state, rather than an event
- Guaranteed initialization and cleanup



Internal Transitions

- Internal transitions are used for events that do not cause a transition to another state.

Internal Transitions

- Internal transitions are used for events that do not cause a transition to another state.

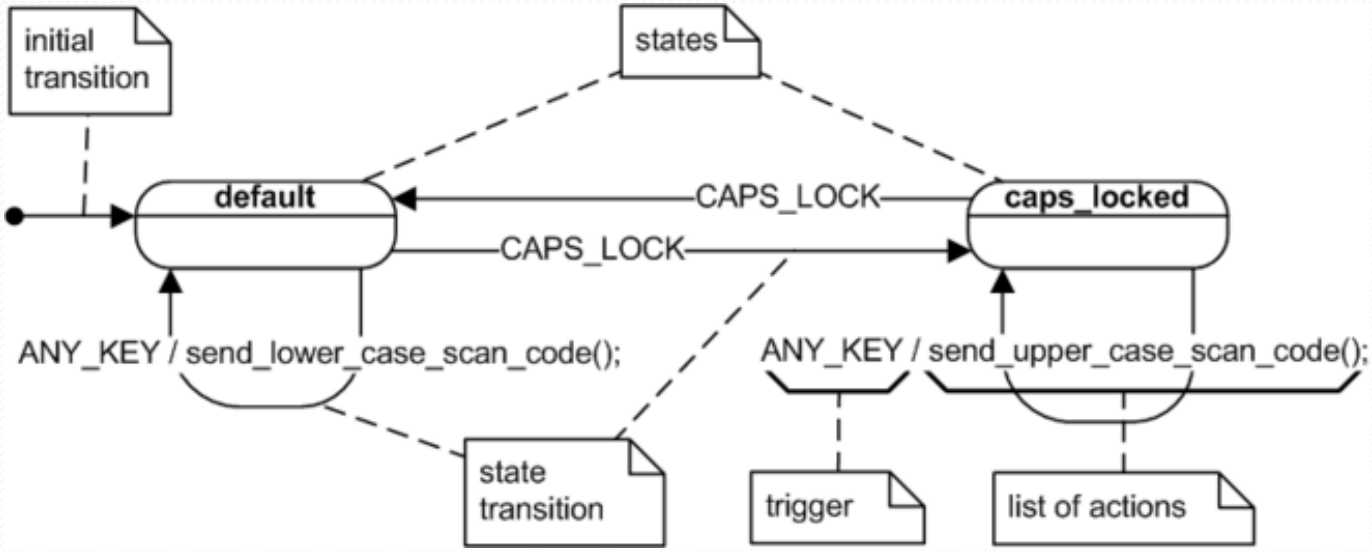


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

Internal Transitions

- Internal transitions are used for events that do not cause a transition to another state.

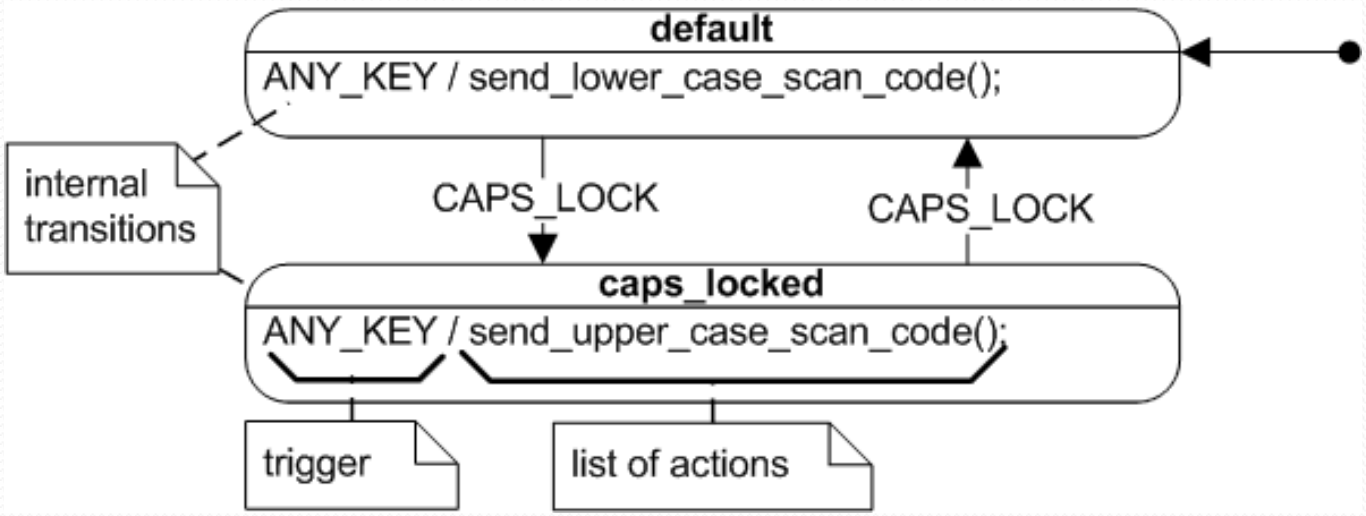


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

Run To Completion (RTC)

- Run-to-completion semantics mean that a state machine must always go from one stable configuration all the way to another stable configuration. [2]
  - Processing an event = 1 RTC step
  - Transitioning between states = 1 RTC step
- How do we deal with this?
  - Event queues
  - Events generally handled in FIFO manner

Drawbacks of UML state machines

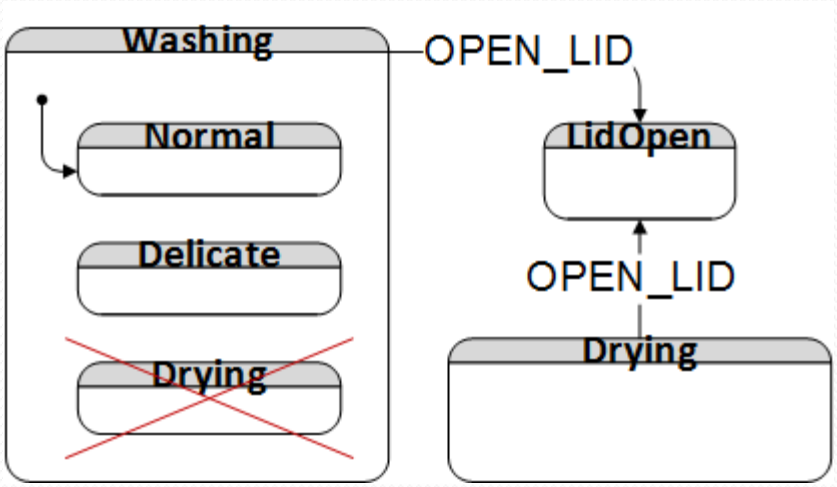
- Difficult to represent the sequence of processing
- Overhead associated with management of event queues, dispatching events
- Challenging to keep the state chart documentation and the code in sync

Topics

- What is a state machine?
  - Mealy, Moore, and UML state machines
- Unified Modeling Language (UML) state machine concepts
  - States
  - Events
  - Actions and transitions
  - Run-to-completion execution
- Creating clean designs with state machines
- Using the Quantum Platform (QP) with state machines
- Detailed design of a washing machine
- Break
- Demo/Exercise

Liskov Substitution Principle (LSP)

- The behavior of a substate should be consistent with its superstate.
- Allows for more efficient use of abstraction



Extended State Machines

- Supplement your state machine with variables to create *extended state machines*.
  - Qualitative aspects = state
  - Quantitative aspects = extended state variables
- A change in variable does not necessarily cause a change in state

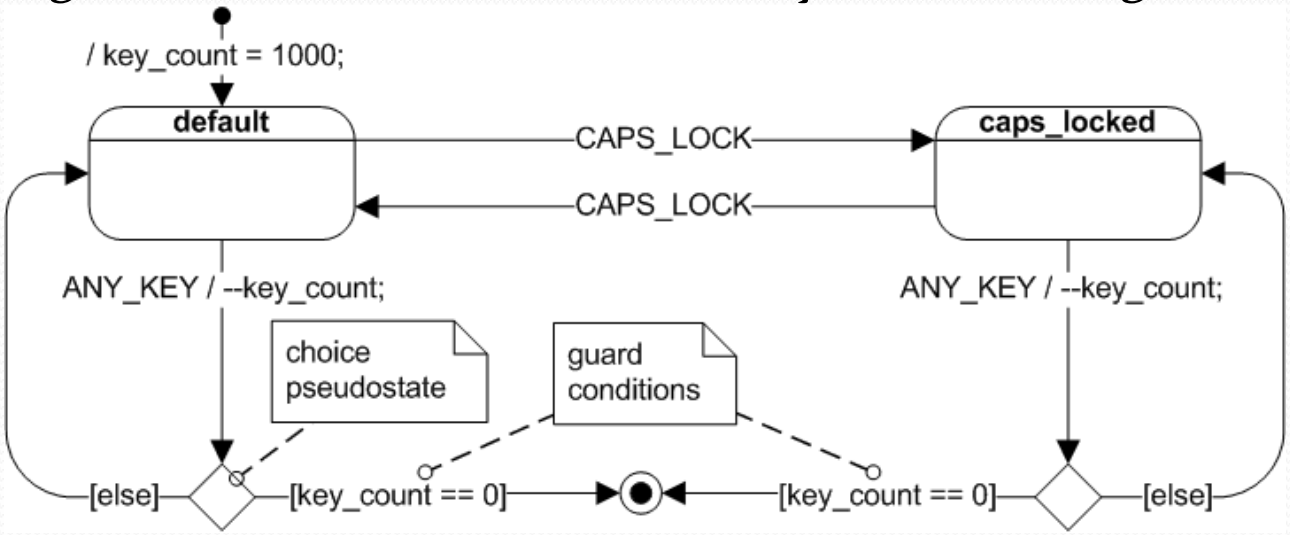
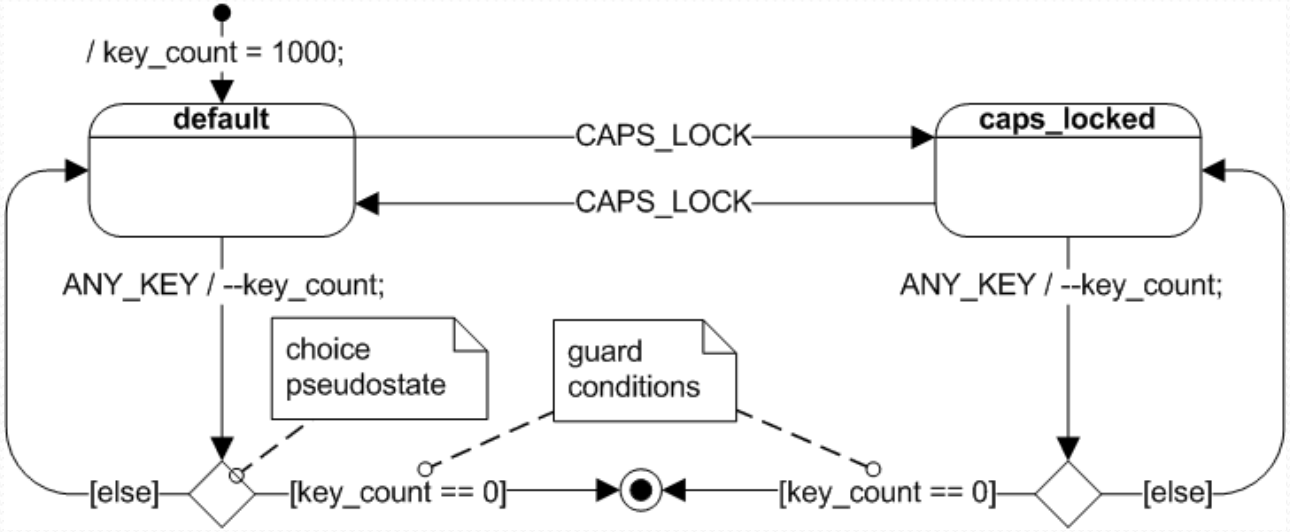


Figure:
{{cc-by-sa-3.0}} Miro Samek, Own work © Miro Samek / CC-BY-SA-3.0(http://creativecommons.org/licenses/by-sa/3.0/)

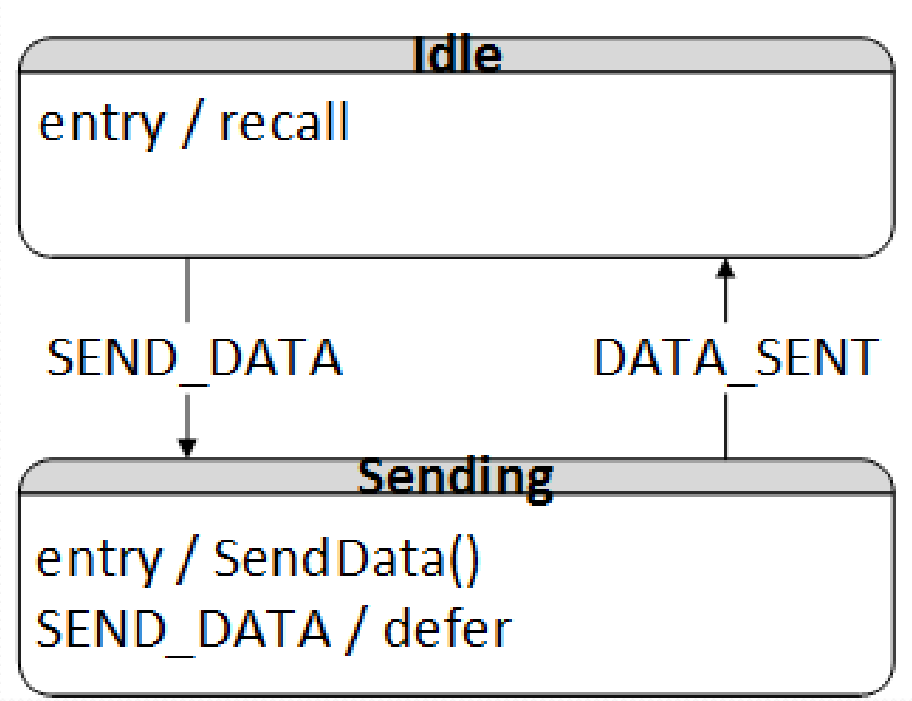
Extended State Machines

- UML indicates conditional behavior using guard conditions
  - DO_SOMETHING_IND[status==OK] / do something
  - DO_SOMETHING_IND[status!=OK] / do something else
- Avoid abuse of variables to avoid spaghetti



Event Deferral

- When you're too busy to handle an event, defer it.



Topics

- What is a state machine?
  - Mealy, Moore, and UML state machines
- Unified Modeling Language (UML) state machine concepts
  - States
  - Events
  - Actions and transitions
  - Run-to-completion execution
- Creating clean designs with state machines
- Using the Quantum Platform (QP) with state machines
- Detailed design of a washing machine
- Break
- Demo/Exercise

A Framework for UML State Machines

- Requirements:
  - Run-To-Completion semantics
  - Hierarchical states
  - Entry and exit actions
  - Events with custom parameters
  - Efficient and lightweight enough for embedded systems
- Quantum Platform (QP) satisfies the above requirements
  - C/C++ implementation
  - Open Source
  - Commercial licenses available

The QHsm Class

- Manages the movement between states
- Interface methods:
  - `Q_HANDLED` – used to indicate that an event was processed by the current state
  - `Q_SUPER` – passes the current event to the parent state for processing. Typically used when an event was NOT handled by the current state.
  - `Q_TRAN` – transitions to another state
- Reserved event signals:
  - `Q_ENTRY_SIG` – event signaling entry actions
  - `Q_EXIT_SIG` – event signaling exit actions
  - `Q_INIT_SIG` – event signaling nested initial transitions

Class interface and description are from [9].

Active Objects in QP

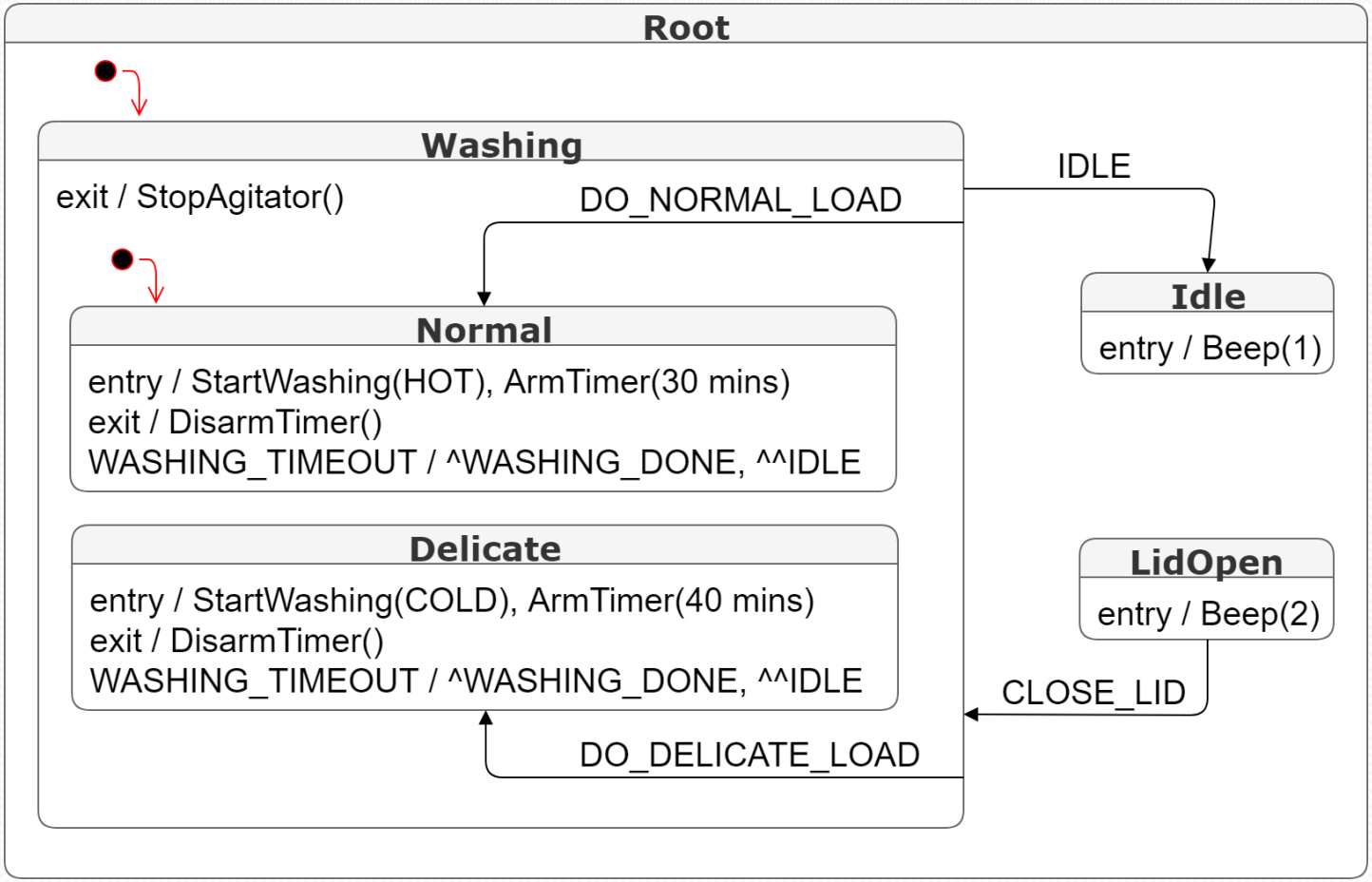
- Characteristics of Active Objects:
  - A state machine that has its own thread of execution
  - An object that encapsulates the behavior of the state machine
  - Contains its own event queue

The QActive Class

- Manages the event queues
- Derived from QHsm
- Intended as a base class for the Active Objects in your design
- Interface:
  - `Subscribe(sig)` – subscribes for delivery of signal, sig, to the active object
  - `post/postLIFO` – posts an event directly to an event queue, FIFO or LIFO
  - `defer` – defers an event to a separate queue
  - `recall` – recalls an event from a separate queue to the main event queue

Class interface and description are from [9].

Example: AOWashingMachine



Class Declaration

```
class AOWashingMachine : public QActive
{
public:
    AOWashingMachine();
    virtual ~AOWashingMachine();

private:
    QEvt const *m_mainEventQueue[MAIN_QUEUE_SIZE];
    QTimeEvt m_cycleTimer;

    // State functions
    static QState Idle(AOWashingMachine *me, QEvt const *pEvent);
    static QState LidOpen(AOWashingMachine *me, QEvt const *pEvent);
    static QState Washing(AOWashingMachine *me, QEvt const *pEvent);
        static QState Normal(AOWashingMachine *me, QEvt const *pEvent);
        static QState Delicate(AOWashingMachine *me, QEvt const *pEvent);

    // Helper functions
    void StartWashing(UINT time);
};
```

Class Declaration

```
class AOWashingMachine : public QActive
{
public:
    AOWashingMachine();
    virtual ~AOWashingMachine();

private:
    QEvt const *m_mainEventQueue[MAIN_QUEUE_SIZE];
    QTimeEvt m_cycleTimer;

    // State functions
    static QState Idle(AOWashingMachine *me, QEvt const *pEvent);
    static QState LidOpen(AOWashingMachine *me, QEvt const *pEvent);
    static QState Washing(AOWashingMachine *me, QEvt const *pEvent);
        static QState Normal(AOWashingMachine *me, QEvt const *pEvent);
        static QState Delicate(AOWashingMachine *me, QEvt const *pEvent);

    // Helper functions
    void StartWashing(UINT time);
};
```

Class Declaration

```
class AOWashingMachine : public QActive
{
public:
    AOWashingMachine();
    virtual ~AOWashingMachine();

private:
    QEvt const *m_mainEventQueue[MAIN_QUEUE_SIZE];
    QTimeEvt m_cycleTimer;

    // State functions
    static QState Idle(AOWashingMachine *me, QEvt const *pEvent);
    static QState LidOpen(AOWashingMachine *me, QEvt const *pEvent);
    static QState Washing(AOWashingMachine *me, QEvt const *pEvent);
        static QState Normal(AOWashingMachine *me, QEvt const *pEvent);
        static QState Delicate(AOWashingMachine *me, QEvt const *pEvent);

    // Helper functions
    void StartWashing(UINT time);
};
```

Class Declaration

```
class AOWashingMachine : public QActive
{
public:
    AOWashingMachine();
    virtual ~AOWashingMachine();

private:
    QEvt const *m_mainEventQueue[MAIN_QUEUE_SIZE];
    QTimeEvt m_cycleTimer;

    // State functions
    static QState Idle(AOWashingMachine *me, QEvt const *pEvent);
    static QState LidOpen(AOWashingMachine *me, QEvt const *pEvent);
    static QState Washing(AOWashingMachine *me, QEvt const *pEvent);
        static QState Normal(AOWashingMachine *me, QEvt const *pEvent);
        static QState Delicate(AOWashingMachine *me, QEvt const *pEvent);

    // Helper functions
    void StartWashing(UINT time);
};
```

Class Declaration

```
class AOWashingMachine : public QActive
{
public:
    AOWashingMachine();
    virtual ~AOWashingMachine();

private:
    QEvt const *m_mainEventQueue[MAIN_QUEUE_SIZE];
    QTimeEvt m_cycleTimer;

    // State functions
    static QState Idle(AOWashingMachine *me, QEvt const *pEvent);
    static QState LidOpen(AOWashingMachine *me, QEvt const *pEvent);
    static QState Washing(AOWashingMachine *me, QEvt const *pEvent);
        static QState Normal(AOWashingMachine *me, QEvt const *pEvent);
        static QState Delicate(AOWashingMachine *me, QEvt const *pEvent);

    // Helper functions
    void StartWashing(UINT time);
};
```

The QEvt Class

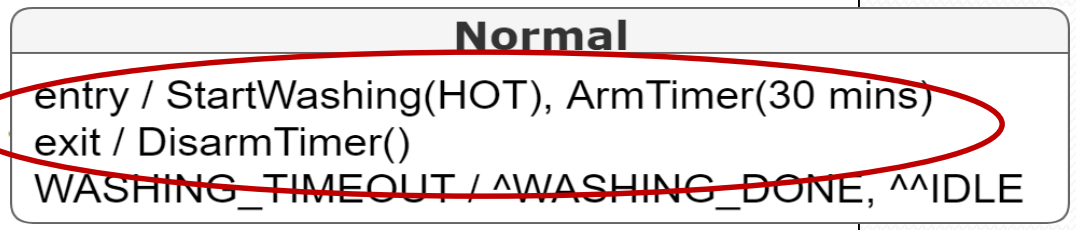
- Represents events (without parameters)
- Serves as the base class for derivation of events with parameters

```
struct EWashingDone : public QEvt
{
    int status;
};
```

Class interface and description are from [9].

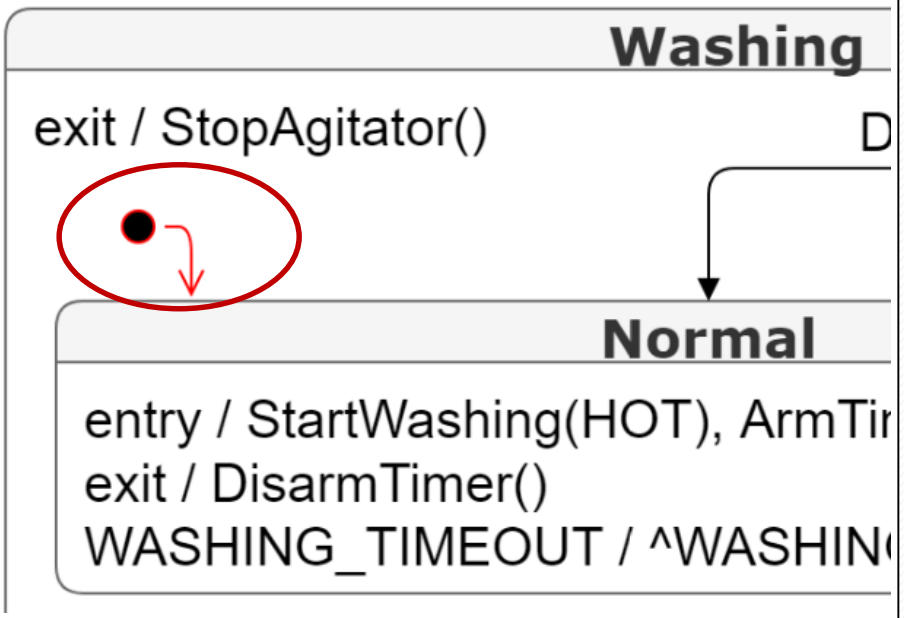
Entry and Exit Handlers

```
QState AOWashingMachine::Normal(AOWashingMachine *me, QEvt const *pEvent)
{
    switch (pEvent->sig)
    {
        //.....................................................
        case Q_ENTRY_SIG:
        {
            me->StartWashing(HOT);
            me->ArmTimer(30);
            return Q_HANDLED();
        }
        //.....................
        case Q_EXIT_SIG:
        {
            me->DisarmTimer();
            return Q_HANDLED();
        }
        //.....................................................
            . . .
    }
    return Q_SUPER(&AOWashingMachine::Washing);
}
```



Initial transition

```
QState AOWashingMachine::Washing(AOWashingMachine *me, QEvent const *pEvent)
{
    switch (pEvent->sig)
    {
        //.....................
        case Q_ENTRY_SIG:
        {
            return Q_HANDLED();
        }
        //.....................
        case Q_EXIT_SIG:
        {
            me->StopAgitator();
            return Q_HANDLED();
        }
        //.....................
        case Q_INIT_SIG:
        {
            return Q_TRAN(&AOWashingMachine::Normal);
        }
```



The QF Class

- Combines framework services
- Static members only
- Not to be instantiated
- Maintains a list of registered active objects
- Interface:
  - `onIdle` – idle mode callback function
  - `publish` – publishes an event to ALL active objects

Class interface and description are from [9].

Publishing Events

```
QState AOWashingMachine::Normal(AOWashingMachine *me, QEvt const *pEvent)
{
    switch (pEvent->sig)
    {
        //.....
            . . .
        //.....................................................
        case WASHING_TIMEOUT:
        {
            QF::publish(Q_NEW(QEvt, WASHING_DONE));
            me->postLIFO(Q_NEW(QEvt, IDLE));
            return Q_HANDLED();
        }
    }
    return Q_SUPER(&AOWashingMachine::Washing);
}
```

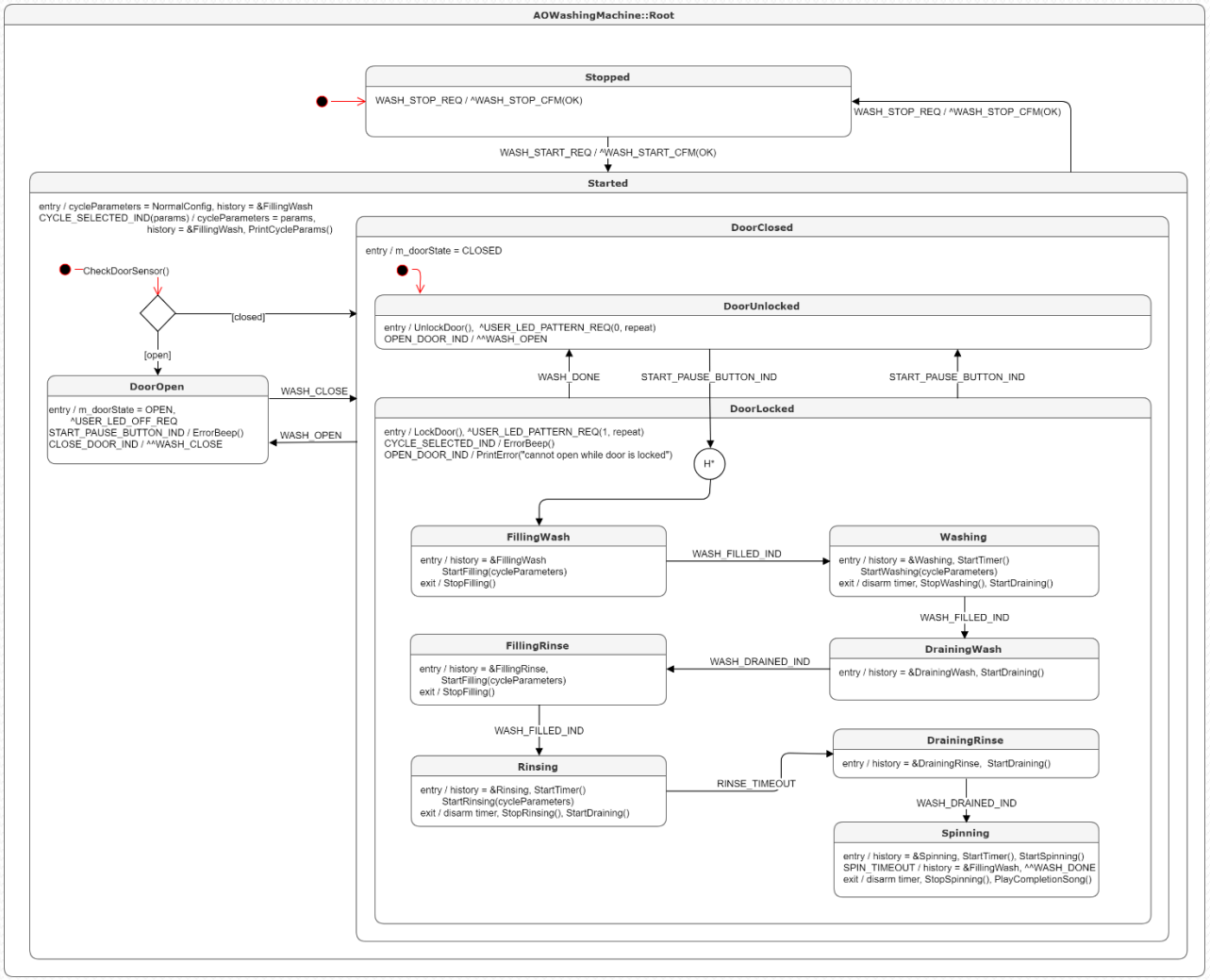
Normal

entry / StartWashing(HOT), ArmTimer(30 mins)
exit / DisarmTimer()
WASHING_TIMEOUT / ^WASHING_DONE; ^^IDLE

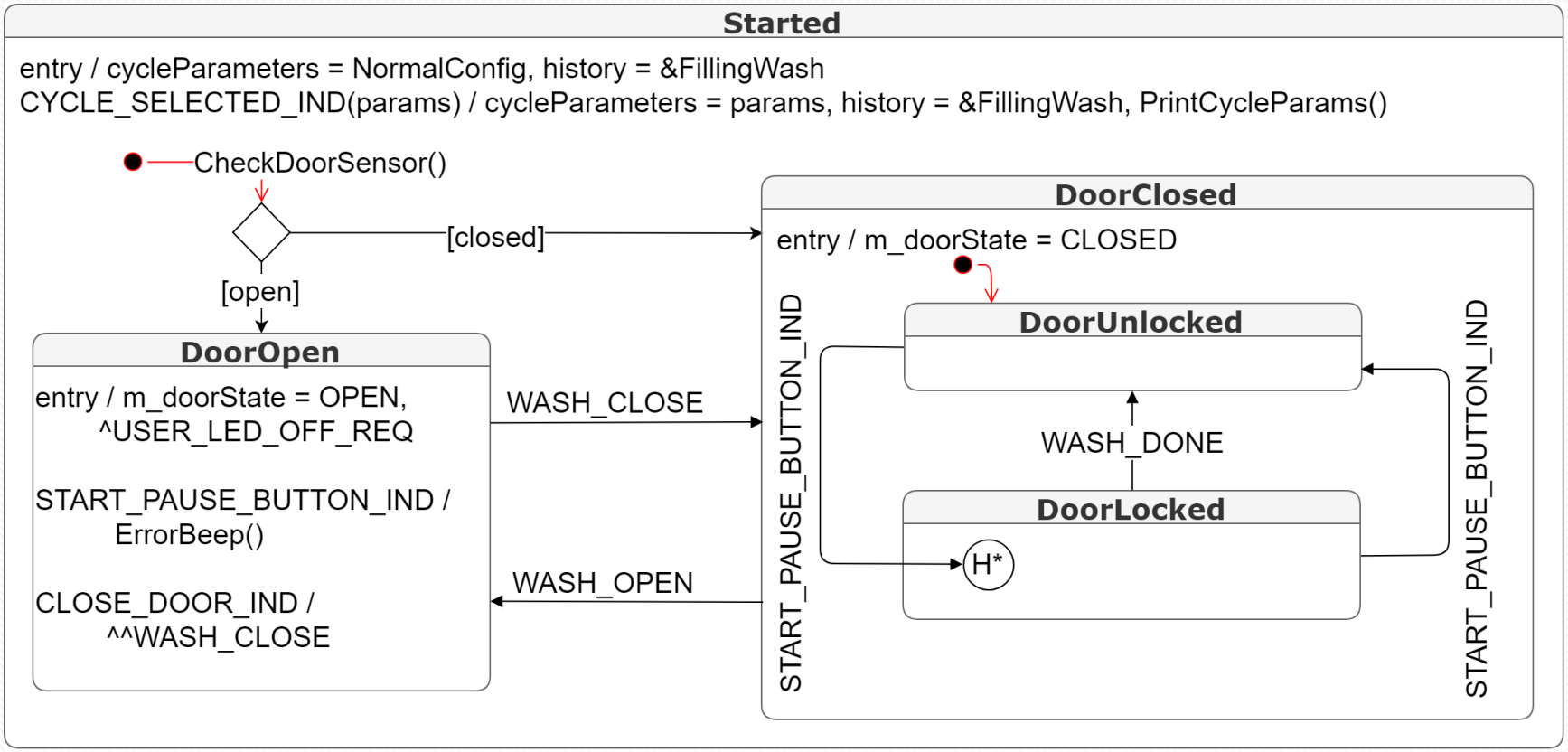
Topics

- What is a state machine?
  - Mealy, Moore, and UML state machines
- Unified Modeling Language (UML) state machine concepts
  - States
  - Events
  - Actions and transitions
  - Run-to-completion execution
- Creating clean designs with state machines
- Using the Quantum Platform (QP) with state machines
- Detailed design of a washing machine
- Break
- Demo/Exercise

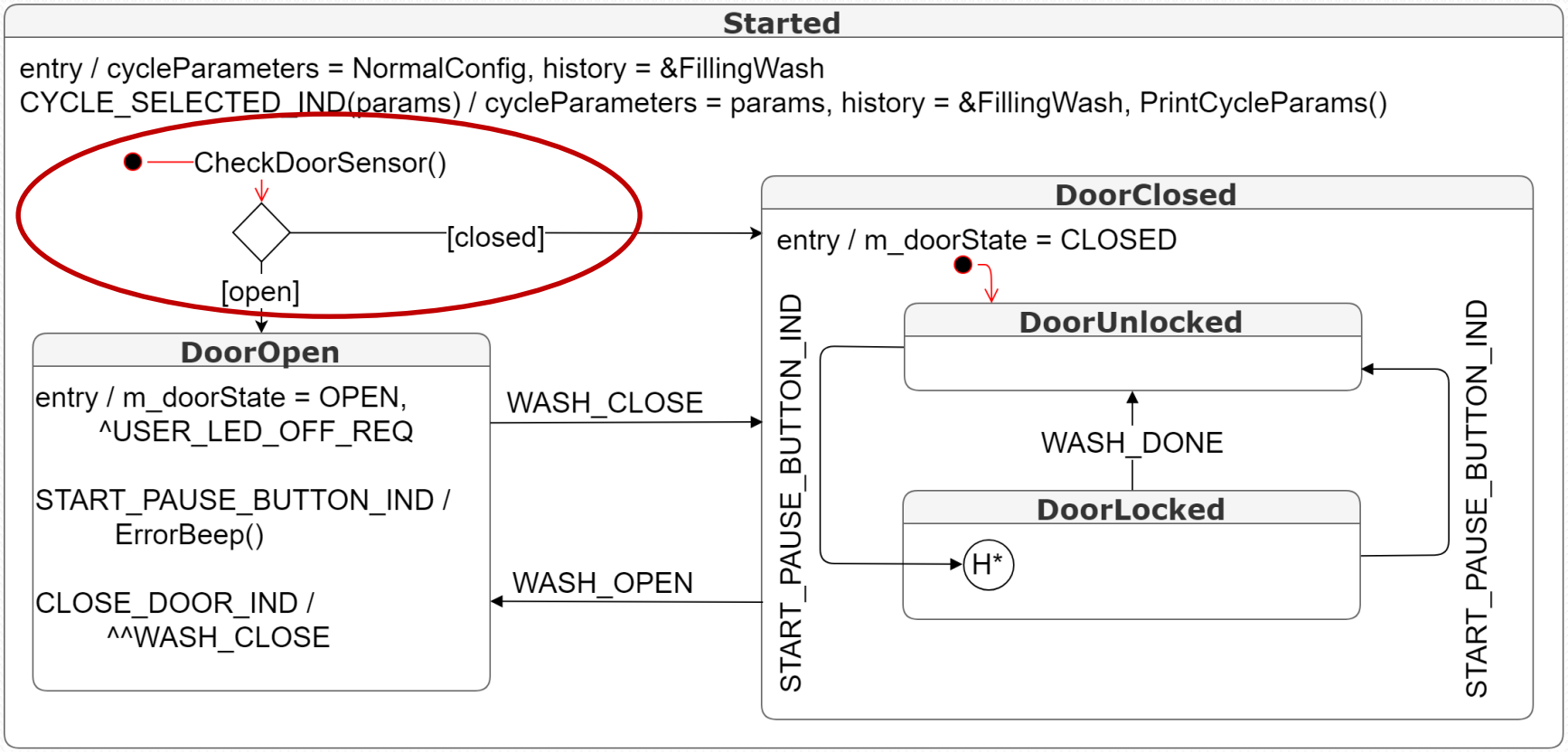
Enhanced AOWashingMachine



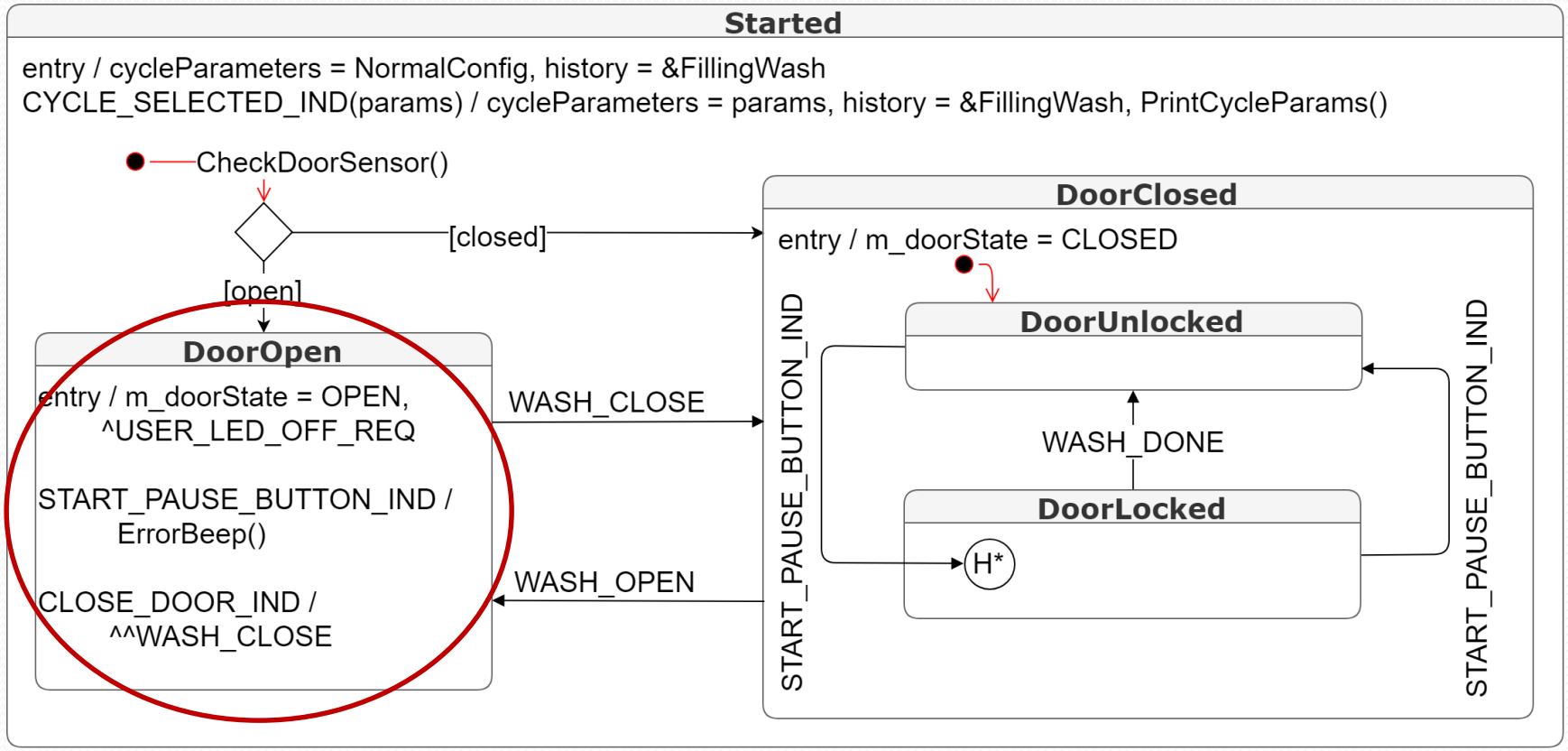
Enhanced AOWashingMachine



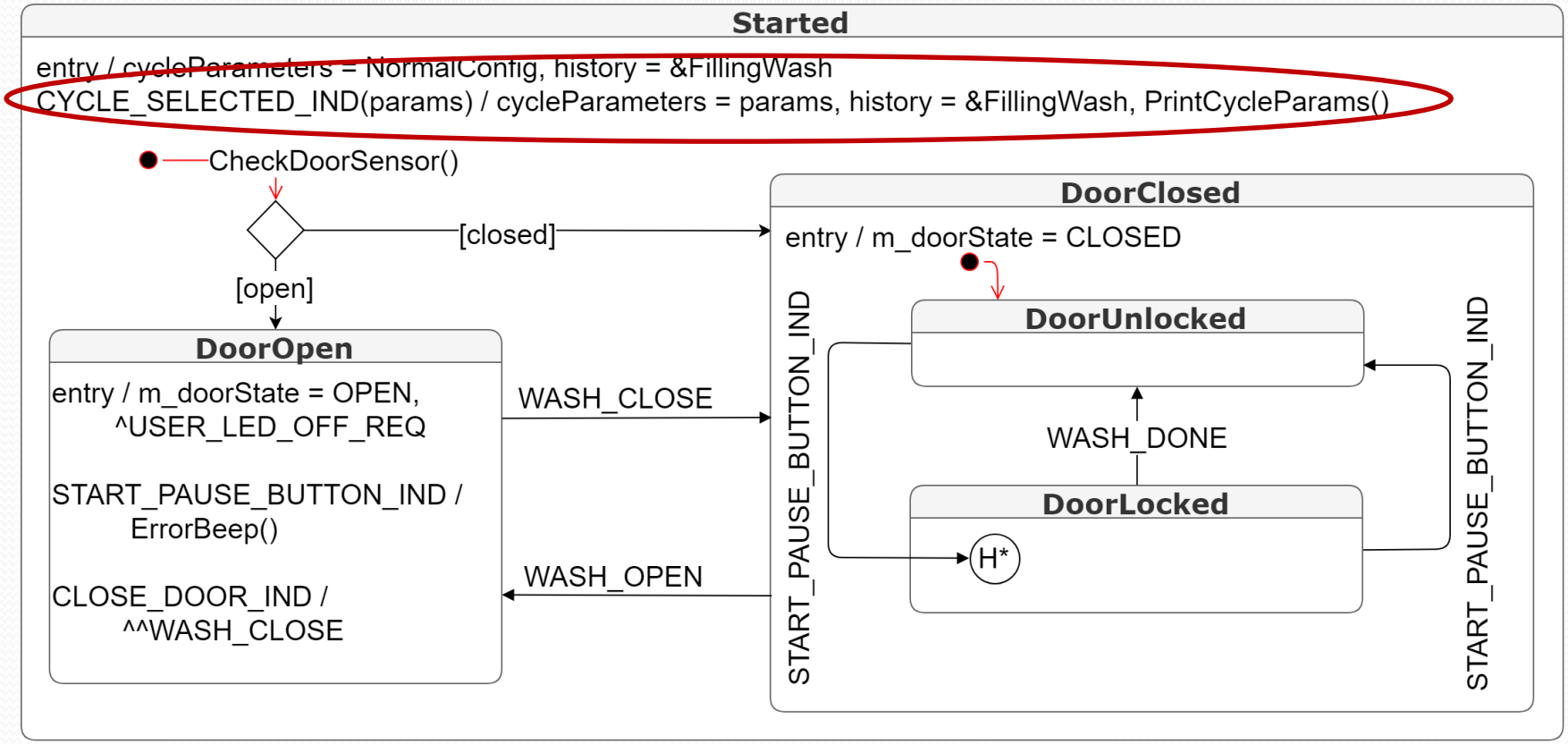
Enhanced AOWashingMachine



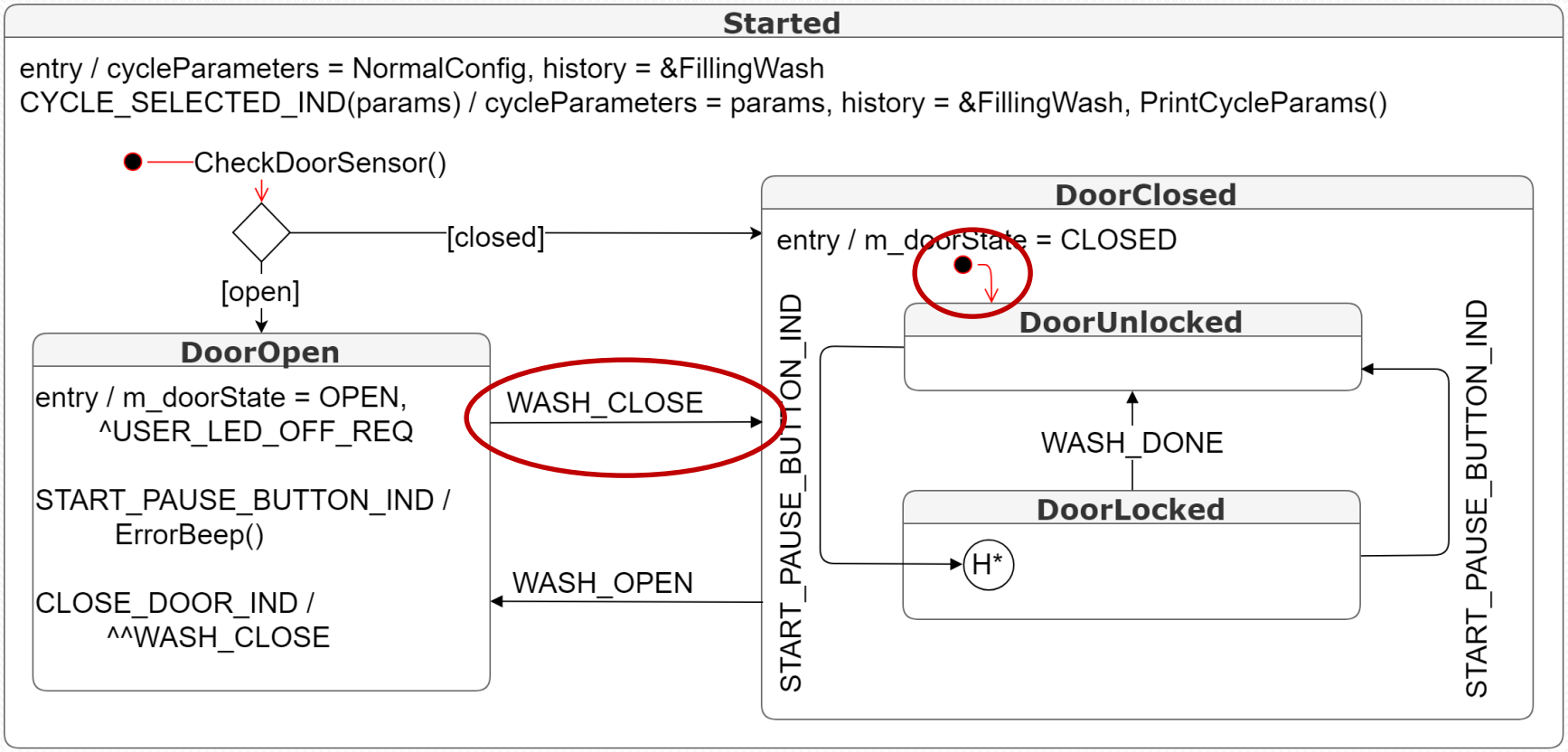
Enhanced AOWashingMachine



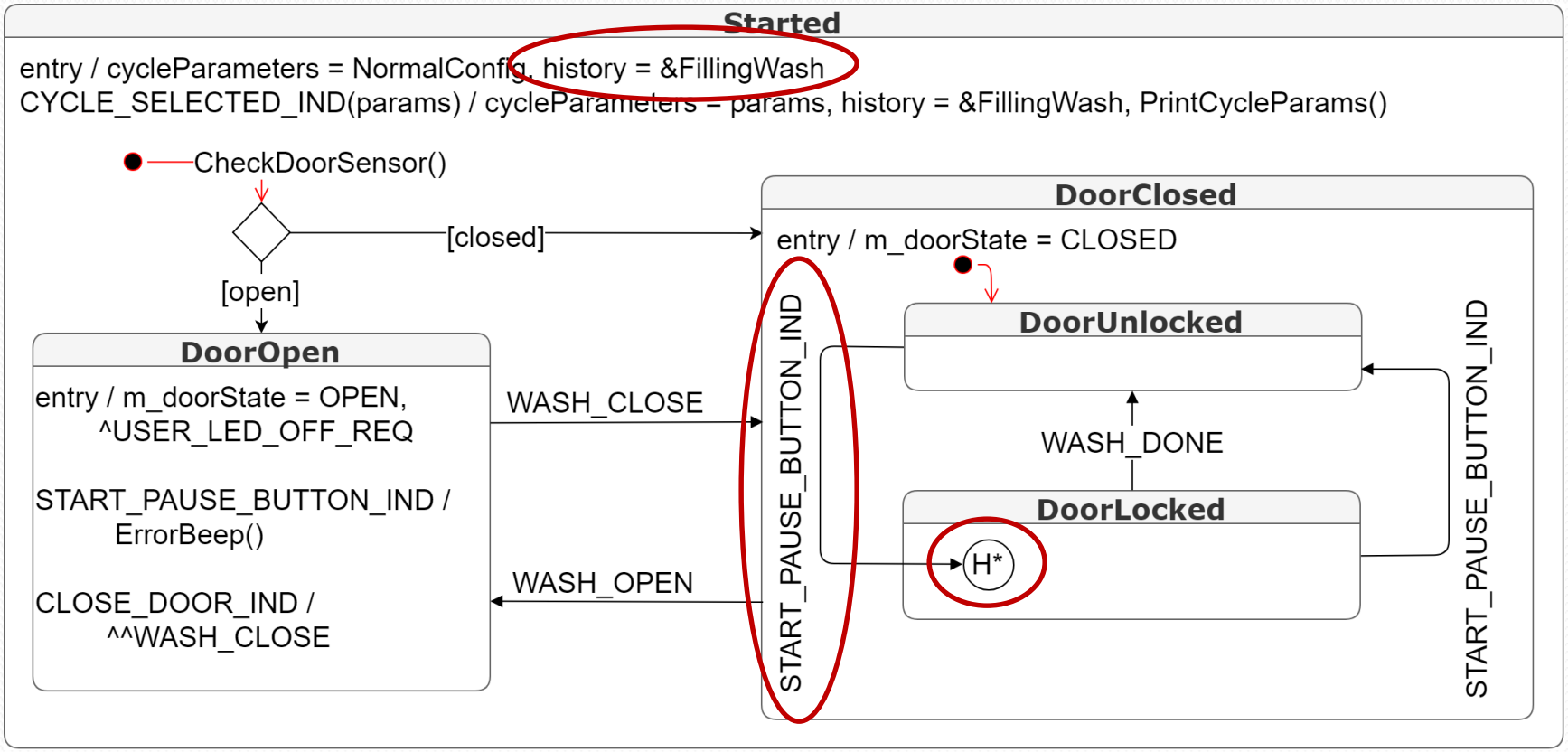
Enhanced AOWashingMachine



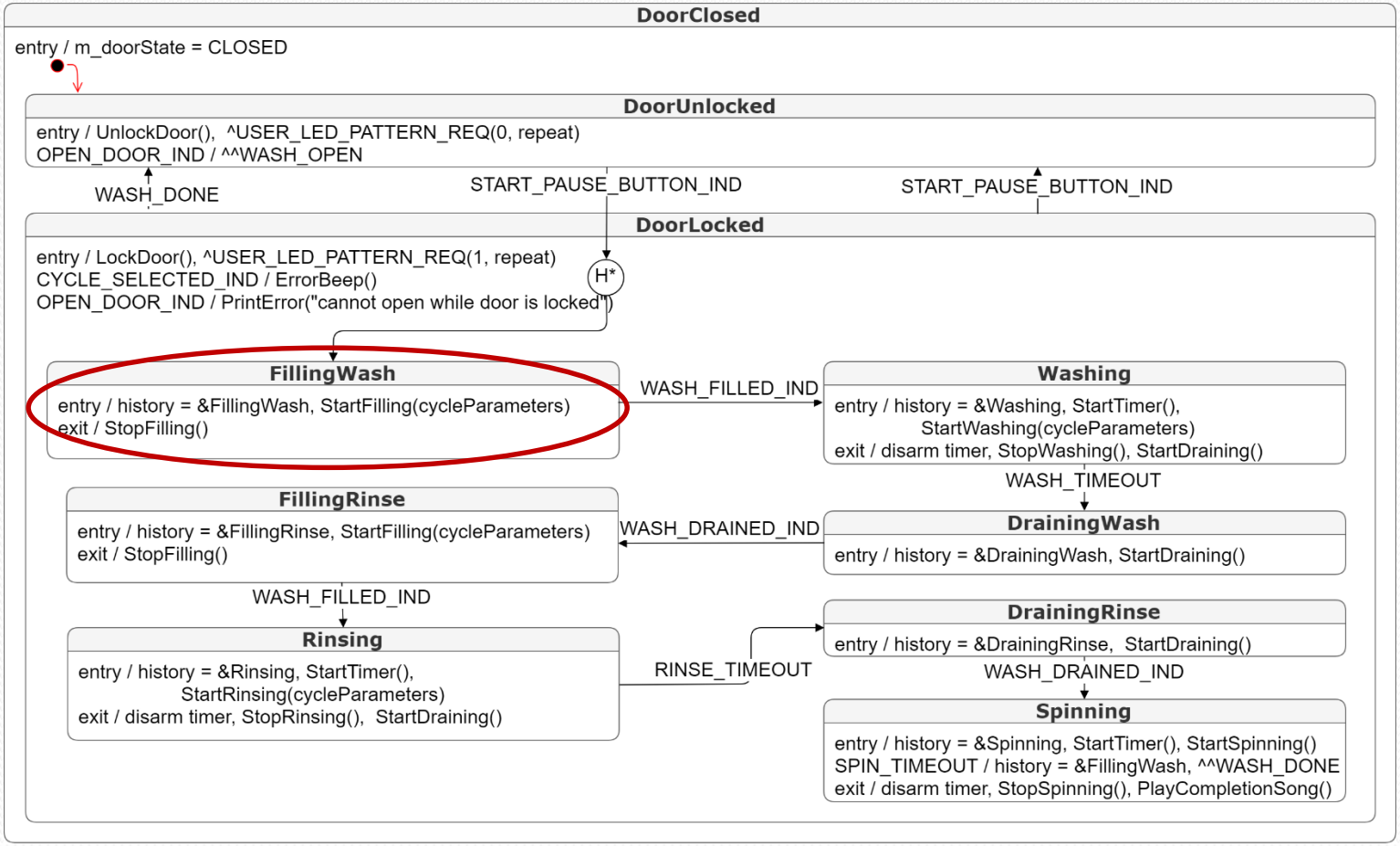
Enhanced AOWashingMachine



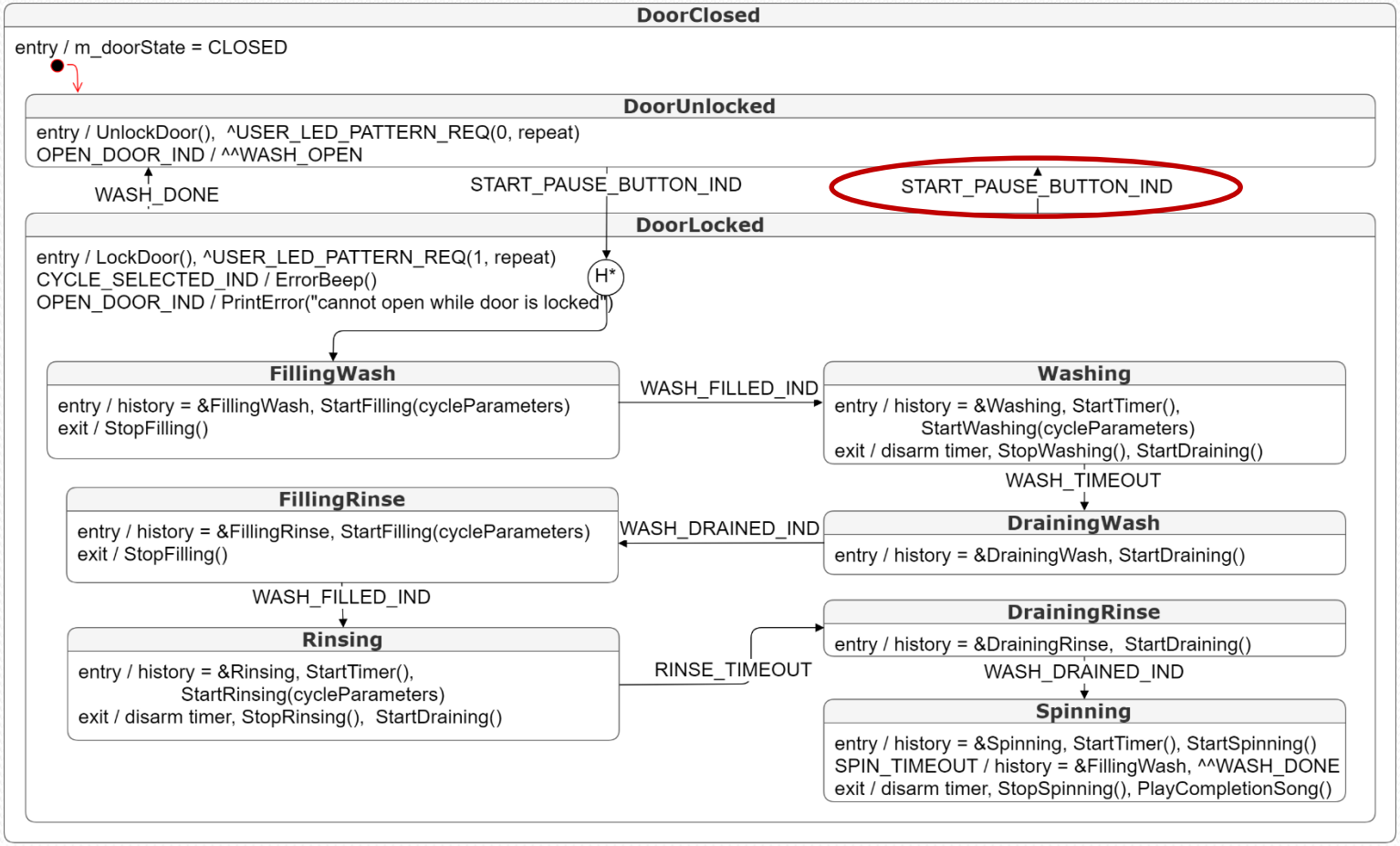
Enhanced AOWashingMachine



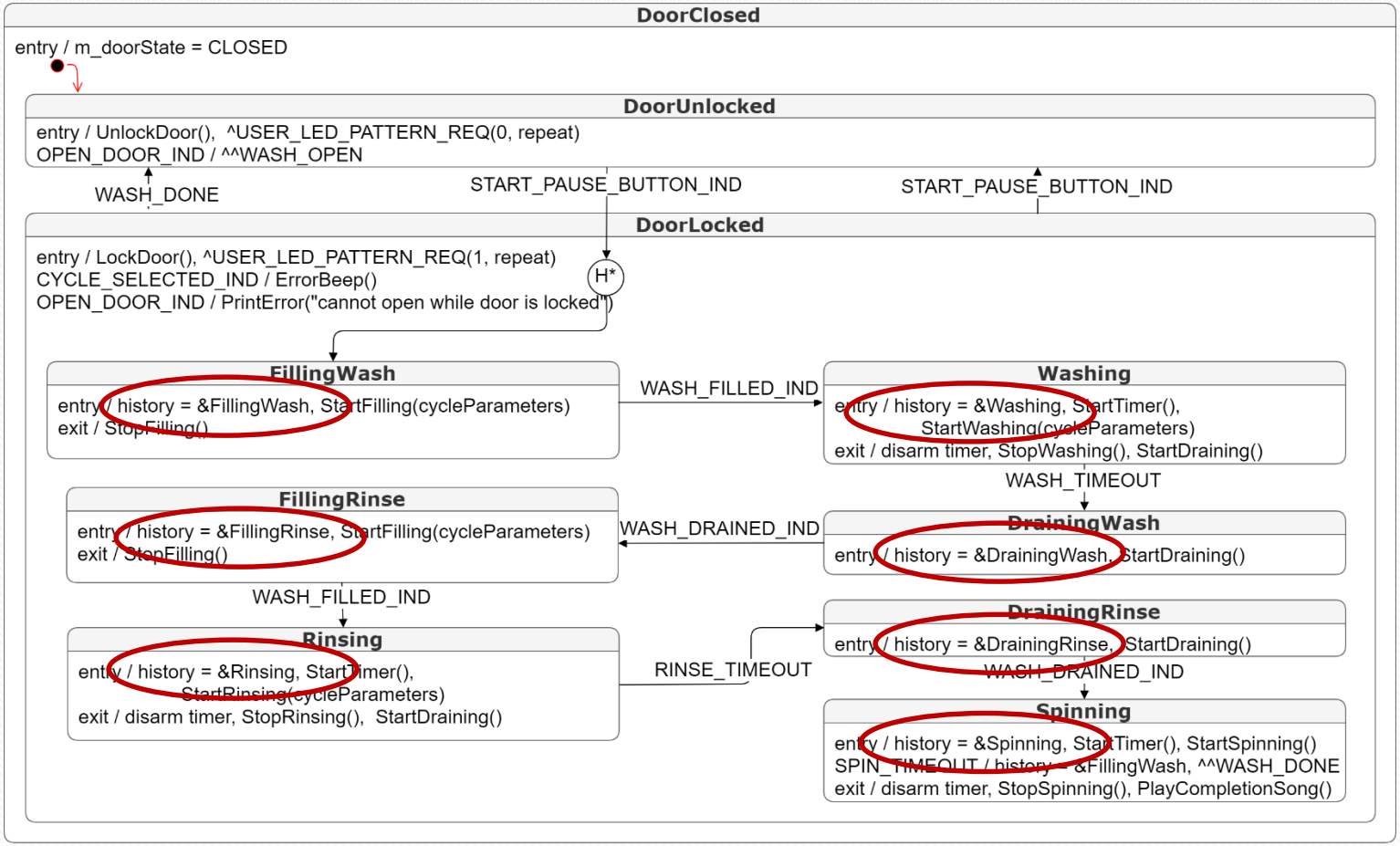
Enhanced AOWashingMachine



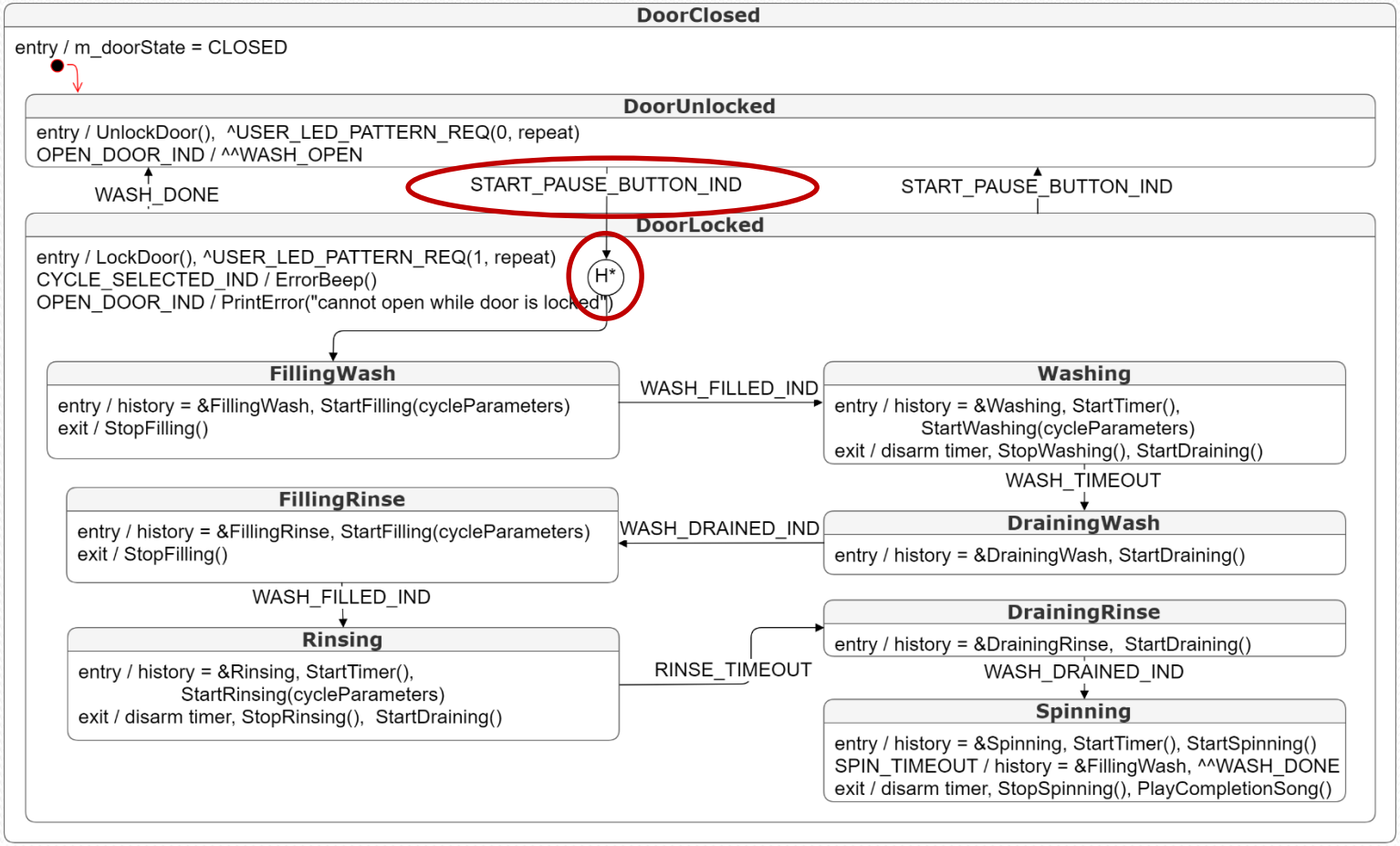
Enhanced AOWashingMachine



Enhanced AOWashingMachine



Enhanced AOWashingMachine



State Design Summary

- UML state machines combine elements of both Moore and Mealy machines.
- Use UML state machines for behavioral systems.
  - Hierarchically combine states with shared characteristics to allow for code reuse and simpler designs.
  - Design patterns: LSP, event deferral, extended state machines, history states
- QP provides a framework for UML state machine implementation.
  - The design of the statecharts is the hard part. Using QP, coding statecharts is easy.

References & Resources

- [1] Anand, Madhukar. *Hierarchical State Machines – a Fundamentally Important Way of Design* [PDF document]. Retrieved from: http://www.cis.upenn.edu/~lee/06cse480/lec-HSM.pdf
- [2] Samek, Miro. *Practical UML Statecharts in C/C++, Second Edition: Event-Driven Programming for Embedded Systems*, CRC Press, 2009, ISBN: 978-0750687065.
- [3] "finite state machine." *American Heritage® Dictionary of the English Language, Fifth Edition*. 2011. Houghton Mifflin Harcourt Publishing Company 19 Apr. 2015. Retrieved from: http://www.thefreedictionary.com/finite+state+machine
- [4] Wikipedia contributors. "UML state machine." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 25 Feb. 2015. Web. 19 Apr. 2015.
- [5] Wikipedia contributors. "Mealy machine." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 16 Apr. 2015. Web. 19 Apr. 2015.
- [6] Wikipedia contributors. "Moore machine." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 16 Apr. 2015. Web. 19 Apr. 2015.
- [7] Wikipedia contributors. "Finite-state machine." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 8 Apr. 2015. Web. 19 Apr. 2015
- [8] Wikipedia contributors. "Liskov substitution principle." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 9 Mar. 2015. Web. 26 Apr. 2015.
- [9] Samek, Miro. Quantum Leaps documentation 5.3.1, [HTML document]. Retrieved from http://www.state-machine.com/qp/qpcpp/index.html
- QP: http://www.state-machine.com/