State Machine

Document #:	P2284R0
Date:	2021-11-13
Project:	Programming Language C++
Audience:	LEWGI
	Israel NB

Reply-to:	Ran Regev
	<regev.ran@gmail.com>

Contents

1 Motivation and Scope

State Machines are fundamental aspect of computer sience and are wildy used in the industry. There are many aspects to consider when implementing a state machine framwork, and a good standard library can ease the burden for developers.

2 Terminology

State Machine means different things to different people. This sectoin sets the terminology for the rest of the paper.

2.1 FSM

FSM is a Finite State Machine. It encapsulates everything this proposal suggests. The facility this paper proposed is called fsm.

```
std::fsm my_fsm;
```

A FSM has the following responsibilities: * Holds its states * Holds a common context for all states * Transit between states.

2.2 State

A State is a position in the state machine. The state machine itself can rest, at a given time, only in a single state.

2.3 Transition

A Transition is the move of the state machine from one state to another.

2.4 Context

The context of a state machine is the common data or functionality that is accesible to all states.

```
    fsm::state wait_for_input("wait-for-input");
    fsm::state open("open");
    fsm::state locked("locked");

    int retries = 3;

    fsm::transition(
        wait_for_input, // when in this state
        // do the following when the input is std::string
        [](const std::string password) -> std::fsm::state&
        {
            if (fsm::context_v > 3) // quering the context
                {return locked;}
            if (password_ok(password)) // cheking the input
               {return open;}
            ++fsm::context_v;   // changing the context
            return wait_for_input;
        }
    // constrcu an fsm with int as the common context
    std::fsm<int> myfsm(retries);
```

2.5 Inputs

When the state machine is constructed and active, the inputs it handles are *events* that the user *fires* on it. From the user's point of view the fsm is a "black box" in the sense that the user doesn't and should't know wich state is about to handle the fired event.

2.6 Outputs

There are many ways a state machine can produce outputs. - The final state of the fsm. - Update of the context to be exmine externaly. - Calling external facilities with results. - Other There is no standard way to generate outputs.

3 UML Standard

[UML] defines a set of facilities that define a state machine. When implementing a state machine we should choose what facilities are in the standard implementation:

— MANDATORY: must have.

— OPTIONAL: may have.
— EXTENSION: may have either in future releases or by means of having users the ability to implement.
— OTIOSE: adding it will degragate from the quality of the implementation.

```
@startuml

!pragma layout smetana

[*] --> Idle
Idle --> [*]
Idle : do nothing wait for issues

Idle -> Development : issue
Development -> Idle
Development -> [*] : resign


@enduml
```

4 Open Issues

— Completeness: should all states handle all inputs? ** Yes: it is safer, it covers all possible situations. ** No: it is just too much effort from the developer point of view to force declarig handling of impossible combinations of state/input.
— Explicit and enforced states transitions vs. implicit and non enforced state transition ADD EXAMPLE

5 Sources

The code for the diagrams in this paper are witten in PlantUML and can be used to regenrate the drawing with uml-generator like https://www.planttext.com

6 Acknowledgements

Michael Park mcpark@gmail.com (for github.com/mpark/wg21)

7 References

[UML] Version 2.2 formal/2009-02-02. State Machine Specification in OMG.
https://www.omg.org/spec/UML/2.2/Superstructure