HSM Notes

HSM (Statecharts): *Advanced formalism for specifying state machines which extend the traditional automata theory.*

- *Support entry and exit actions for states*
    - *Helps guarantee the idea of initialization and cleanup*

Entry Actions: *are always executed starting with the outermost state being similar to class constructed form the most general class*

Exit Actions: *Are always executed in exact reverse order; Entry and exit actions combined with state nesting complicate transition sequence*

The Importance of An Event Driven Framework

HSM need Framework:

Minimum Requires:

- *RTC (Run To Completion)*
- *Queuing of events*
- *Event based timing service*

Without the event driven infrastructure like Real time Embedded Framework: *State Machines need infrastructure !!!!!!!!*

State Machines (https://www.state-machine.com/qpc/srs_sm.html)

State: *History of past histories of a/the system that is equivalent to the sense that the future behavior of the system given any of these past histories will become identical*

Transitions: *change from one State to another during the lifetime of a system*

- *Caused only by events*
    - *Triggering factor == **Triggering Event or Trigger of the Transitions***

State Machines: *set of all States (with equivalent classes of relevant histories) + all transitions (rules for changing States)*

- ***Entry and Exit actions:***
    - *When in states define entry actions that are executable when the state becomes active; Entry actions are typically used to initialize state specific variables or perform setup tasks*
- ***Transition conditions:***
    - *Each transition between states in an HSM can be associated with a condition; Transitions only occur when their associated conditions evaluated to true*

    - *Entering in a nested state, you also enter its parent state, which can be a way to trigger entry actions in both the nested and parent states*
- ***Hierarchical Structure:***
    - *HSMS allow for the nesting of states within states; this hierarchical structure helps in organizing and modeling complex systems more intuitively*
    - *When entering this nested state, you also enter its parent state which can be a way to trigger entry actions in both the nested and parent states*
- ***Orthogonal Regions:***
    - *HSMS can have orthogonal regions which are essentially multiple independent state machines running parallel == * important can be used to model concurrent behavior*
- ***History States:***
    - *HSM often include history states, which allow a state machine to remember its previous state *useful for returning to previous state especially in complex hierarchies*
- ***Event Driven:***
    - *Transitions in HSMS are typically event driven , Events can be triggered by user interactions, timers, or other sources, ad they can cause transitions between states or regions*
- ***Superstates and Substates**:*
    - *HSM's can have superstate (parent states) and substates (nested states). Superstates can have transitions and actions of their own, and they can contain substates*
    - *Events occurring in substrates can be handled by the substates or bubbled up to the superstates depending on the design*
- ***Default Transitions:***
    - *HSM's can define transitions for cases when none of the explicit transitions have conditions that evaluate to true. These default transitions provide a fallback behavior*
- ***Guards:***
    - *guards are conditions associated with transitions. They provide an additional level of control by allowing or blocking transitions based on specific criteria*
- ***Self Transitions:***
    - *A state transition to itself, allowing you to trigger exit and reentry actions without leaving the state*

Hierarchical State Machine: *is an advanced formalism which extends traditional state machines in several ways. Important innovations of UML state machines over classical states machines is the introduction of hierarchically nested machines*

Hierarchically Nested States: *refers to a feature in state machine modeling, particularly in the context of Hierarchical State Machines (HSMs). In HSMs, states can be organized in a hierarchical structure, where states can contain other states, creating a nesting of states within states. This hierarchical nesting is used to represent complex systems and their behavior in a more organized and modular manner.*

Implementation Strategy For State Machines:

- *efficiency in time (CPU cycles)*
- *efficiency in data space (RAM footprint)*
- *efficiency in code space (ROM footprint)*
- *monolithic vs. partitioned with various levels of granularity*
- *maintainability (with manual coding)*
- *maintainability (via automatic code generation)*
- *traceability from design (e.g., state diagram) to code*
- *traceability from code back to design*
- *other, quality attributes (non-functional requirements)*

None could be optimal for all circumstances and the QP framework shall support multiple and interchangeable strategies.

QP (Quantum Platform): *framework is a software framework for developing event-driven, real-time, and embedded systems. It is particularly focused on applications that require precise control and are often found in safety-critical and mission-critical domains such as aerospace, automotive, industrial automation, and medical devices. QP is designed to facilitate the development of such systems by providing a structured and efficient way to manage state machines, hierarchical state machines, and finite state machines (FSMs).*

- *Important for dealing with complex state logic and real time requirements*

Dispatching Events to a State Machine in QP Framework

Dispatching:

- *an event to the state machine, and it requires interaction between the QP framework and the QP application*

State Machine Specification:

- *state machine code(when the state machine is coded manually) or a state machine model (when the state machine is specified in a modeling tool like QM*

State Machine Processing:

- *Decides which elements of the "state machines specification" to call*
- *Executes some action and returns back to the SMP (State Machine Processor) QM Framework with the status info as to what has happened*
- *Passive software component that needs to be explicitly called from some control threat t dispatch each event to the given state machine object*

Restrictions: *the dispatch operation must necessarily run to completion before another event can be dispatched to the same state machine object*

The provided requirements are part of a detailed specification for the QP (Quantum Platform) framework, which is a software framework for developing event-driven, real-time, and embedded systems. Each requirement outlines a specific feature or behavior that the QP framework should support. Here's a summary of each requirement:

Requirements

****REQ-QP-02_00: Support for State Machines****

- *The QP framework should support state machines for both Active Objects and passive event-driven objects in the application.*
- *This support includes hierarchical state machines (HSMs) and rules for "State Machine Specifications" and "State Machine Processor" to handle events according to HSM semantics.*

****REQ-QP-02_10: Support for Multiple State Machine Implementation Strategies****

- *The QP framework should support multiple and interchangeable state machine implementation strategies.*
- *Applications can choose a strategy based on the type of a state machine object, and QP should resolve the matching "State Machine Processor" at runtime.*
- *Applications can add their own implementation strategies, and QP should resolve the matching dispatch method based on the type of the state machine object.*

****REQ-QP-02_20: State Machine Implementation Strategy for Manual Coding****

- *The QP framework should provide a state machine implementation strategy optimized for "manual coding."*
- *Changes in the state machine design should only require changing a single matching element in the implementation.*

****REQ-QP-02_21: State Machine Implementation Strategy for Automatic Code Generation****

- *The QP framework should provide a state machine implementation strategy optimized for "automatic code generation."*
- *This strategy may contain some redundant information to improve efficiency and support more advanced state machine features than manual coding.*

****REQ-QP-02_22: Bidirectional Traceability****

- *All state machine implementation strategies provided by QP should be bidirectionally traceable, ensuring that each state machine element in the design corresponds to an element in the implementation and vice versa.*

****REQ-QP-02_23: Access to the Current Event****

- *All state machine implementation strategies should allow applications to easily access the current event with its signal and parameters during runtime processing.*

****REQ-QP-02_24: Access to Instance Variables****

- *All state machine implementation strategies should allow applications to easily access instance variables associated with a given state machine while maintaining encapsulation.*

****REQ-QP-02_25: Check if a State Machine is in a Given State****

- *State machine implementation strategies might supply a method for checking if a state machine is in a given state. The "is-in" state operation returns 'true' if the current state is equal to or a substate of the given state.*

****REQ-QP-02_30: Support for Hierarchical State Machines****

- *All state machine implementation strategies should support hierarchical state machines with features specified in sub-requirements 02_(3x).*

****REQ-QP-02_31: Support for Nested Substates****

- *All state machine implementation strategies should support states capable of holding hierarchically nested substates, allowing for easy addition, removal, or reconfiguration of substates within a state.*

****REQ-QP-02_32: Entry Actions to States****

- *All state machine implementation strategies should support entry actions to states, allowing for optional actions to be executed when a state is entered.*

****REQ-QP-02_33: Exit Actions from States****

- *All state machine implementation strategies should support exit actions from states, allowing for optional actions to be executed when a state is exited.*

****REQ-QP-02_34: Nested Initial Transitions in Composite States****

- *All state machine implementation strategies should support nested initial transitions in composite states, specifying the execution sequence of these transitions.*

****REQ-QP-02_35: Transitions Between States at Any Level of Nesting****

- *All state machine implementation strategies should support transitions between states at any level of nesting, providing clear semantics for transitions with a trigger and specifying the execution sequence.*

****REQ-QP-02_36: Support for Internal Transitions****

- *All state machine implementation strategies should support internal transitions in states, which are transitions that execute associated actions but do not change the current state.*

****REQ-QP-02_37: Support for Guard Conditions****

- *All state machine implementation strategies should support guard conditions attached to regular and internal transitions, allowing transitions to be enabled or disabled based on the evaluation of these conditions.*

****REQ-QP-02_38: Top-Most Initial Transition****

- *All state machine implementation strategies should support a top-most initial transition that can be explicitly triggered independently from the instantiation of the state machine object.*

****REQ-QP-02_39: Support for Transitions to History****

- *All state machine implementation strategies should support transitions to history, both shallow and deep histories, which allow the state machine to return to the most recently active substate upon reentry.*

****REQ-QP-02_40: Optional Support for the Top-State****

- *State machine implementation strategies may supply the concept of a top-state, which represents the ultimate root of the state hierarchy and silently ignores all events.*

****REQ-QP-02_50: Submachines for Reuse of Behavior****

- *State Machine Implementation Strategy "optimized for automatic code generation" should support the reuse of behavior via submachines, allowing for the packaging of composite states as units for reuse.*

****REQ-QP-02_51: Multiple Submachines in a Hierarchical State Machine****

- *Submachines can be added to a hierarchical state machine, and a submachine can have its own entry and exit actions and an initial transition.*

****REQ-QP-02_52: Support for Entry Points in Submachines****

- *Submachines should support entry points, which target substates of the submachine and may have associated actions.*

****REQ-QP-02_53: Support for Exit Points in Submachines****

- *Submachines should support exit points, which serve as termination points for all state transitions exiting the submachine.*

****REQ-QP-02_54: Support for History Segments in Submachines****

- *Submachines should support history segments, both deep and shallow histories, to remember and transition to the most recently active substate.*

****REQ-QP-02_55: *Support for Submachine-States*****

- *State Machine Implementation Strategy "optimized for automatic code generation" should support submachine-states, which are instances of submachines placed in the context of the host hierarchical state machine.*

****REQ-QP-02_56: *Support for Exit Segments in Submachines*****

- *Submachines should support exit segments, which are transitions originating from exit points in a submachine state. These segments do not have a trigger but can have associated actions*.