FSM API

Arturo Laurenzi , Luca Muratore, Giuseppe Rigano

FSM API Features

- #include **<XBotInterface/StateMachine.h>**
- Package **XBot::FSM**
- States are c++ classes **inheriting** from a Base class that you have to define
- The Base class must **extend** the class State<BaseClass, SharedData>
- SharedData can be a definition of a data structure (e.g. struct) that can be used to share data between states
- Possibility to create **custom Event** and **Message** by inheriting Event and Message classes
- Instance of the class StateMachine<BaseClass, SharedData> provides a way to **register states** and **interact** between them

Message

- Allows the fsm to initialize a state
- A basic implementation of an event is provided by the Message class
- It allows you to specify just a string
- A custom Message can be created by **extending** the Message class
- Used by the **init(e)** method method to trigger a specific **entry message** in a state
- It will be triggered at **every transition** towards the state

Event

- Allows the fsm to react to a specific event
- A basic implementation of an event is provided by the Event class
- It allows you to specify just a string
- A custom Event can be created by **extending** the Event class
- Used by the **send_event(e)** method to trigger a specific **react event** in a state

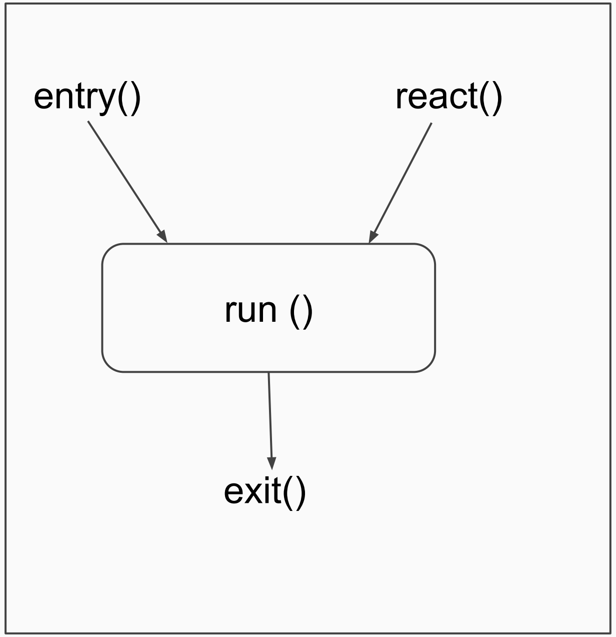
BaseState class

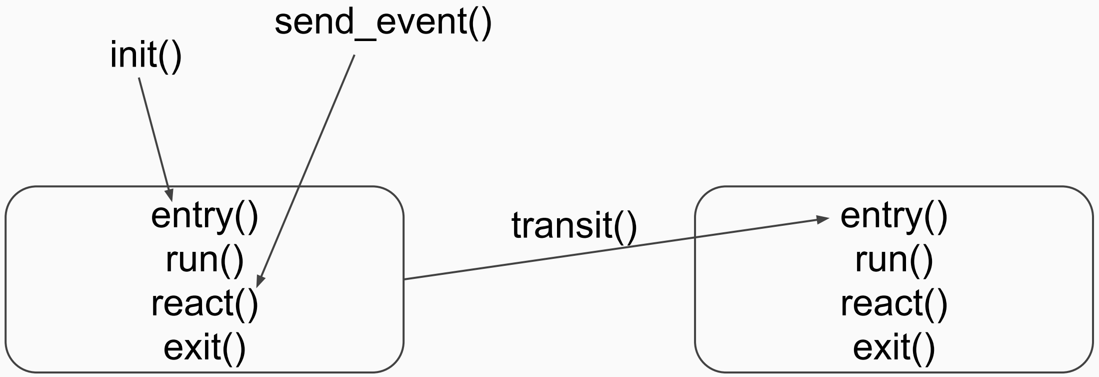
- It **extend** the class State<BaseClass, SharedData>
- It define a list of **virtual react event** and **virtual entry message** used by the states (classes inheriting from the BaseClass you define)
- By default the **generate fsm script skeleton** provides the implementation of an **entry** and **react** method using the default Message and Event class.
- The **react and entry** method usually will be empty because **the implementation will be provided in the specific state.**

State

- Each state is a class **inheriting** from BaseClass
- It provides the following methods you need to implement
  - **get_name** , used to identify the state
  - **run** , it will contains the to be executed in the state
  - **entry**, it will be triggered at each transition toward the state
  - **react**, it will triggered when a specific event is sent
  - **exit**, it will be triggered when the state is left
- Is it possible to **add multiple entry and react methods** in order to respond to **custom events** and **messages**
- Transition towards a state is possible by calling the **transit method** inside **run** or **react**
- **Transit** method is **overloaded** in order to accept either just the next state or also the message

State


entry()
react()
run ()
exit()


send_event()
init()
entry()
run()
react()
exit()
transit()
entry()
run()
react()
exit()

StateMachine steps

1. **Create** an instance fsm of the StateMachine class
   - XBot::FSM::StateMachine< myfsm::MacroState , myfsm::SharedData > fsm;
2. **Register** every state to the fsm instance
   - fsm.register_state(std::make_shared<myfsm::state1>());
3. **Init** the fsm with a state
   - fsm.init("state1");
4. **Run** the fsm in the control_loop
   - fsm.run(time, 0.01);
5. Possibility to **send_event** to the current state
   - fsm.send_event(myfsm::Event(1));

Starting from Skeleton: Steps (1)

1. Run the script: **generate_XBot_PluginFSM.sh PluginName state1 stateN**
    - It will create a RT plugin ready to use skeleton with a FSM
    - To keep things modular, the fsm implementation is done in a **different file** with respect to the plugin file
    - it will create a MacroState class as a BaseClass, and all the states will be **inherithed** from MacroState
    - It will register all the state and initialize the first one
    - SharedData struct contains a **reference to the _robot variable**
2. Create **custom events and messages** if you need them and remember to insert **react and entry** methods inside the MacroState class and provide their implementation in the specific state

Starting from Skeleton: Steps (2)

3. Provide the implementation for the following methods in each state
    - **run()**
    - **entry()**
    - **react()**
    - **exit()**
    - **get_name()**