pysmlib

A Python Finite State Machine Library for EPICS

Dr. Marcato Davide

INFN – Legnaro National Laboratories

davide.marcato@lnl.infn.it

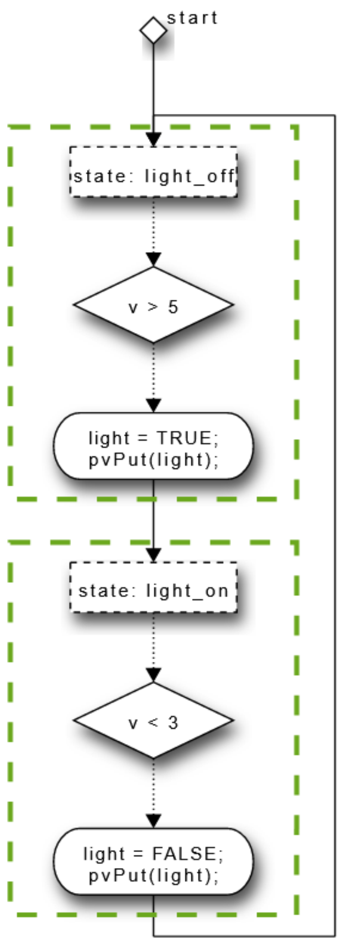
18th International Conference on Accelerator and Large Experimental Physics Control Systems

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

INFN

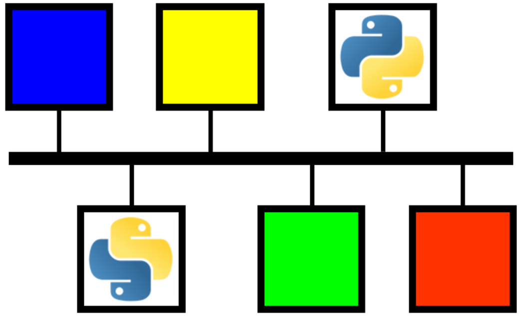
The EPICS Sequencer

- A tool to define procedures and sequences of operations in EPICS
- State Notation Language
  - To describe Finite State Machines (FSM) states and transitions
  - C-like language, transcompiled to C
- Standard tool in the EPICS community
  - First proposed on the original EPICS paper
  - Good performance and reliability
  - Flexible programming model
- Low level language
  - Unfamiliar to new users
  - Limited expandability



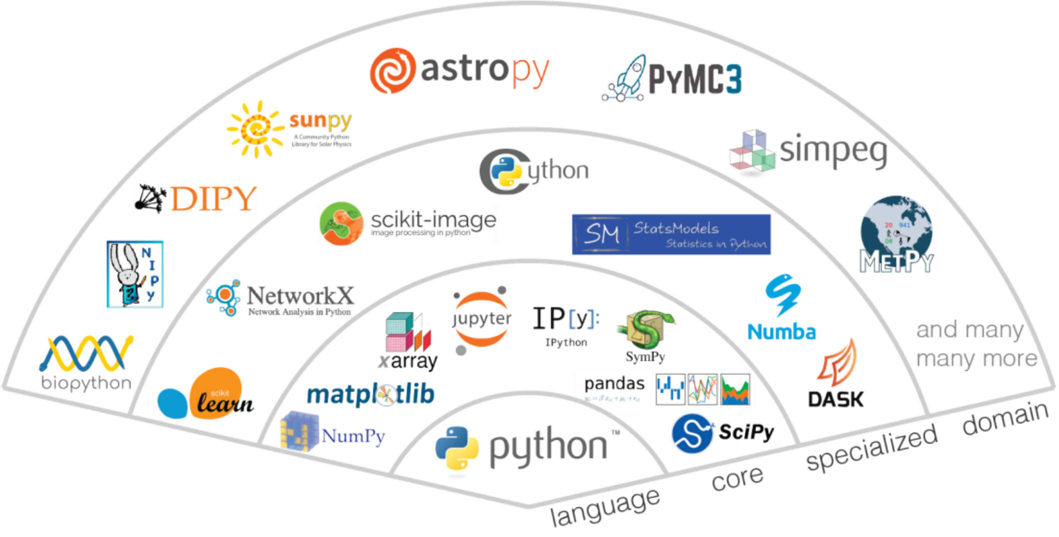
Alternatives

- Vanilla PyEpics scripts
  - Easy, fast to prototype
  - Basic functionality
- Bluesky project
  - Complete suite of tools for data acquisition, experiment specification and orchestration.
  - Advanced functionalities
  - Requires a big investment into their design model
- Facility or experiment-specific tools
  - Not available to smaller labs/experiments



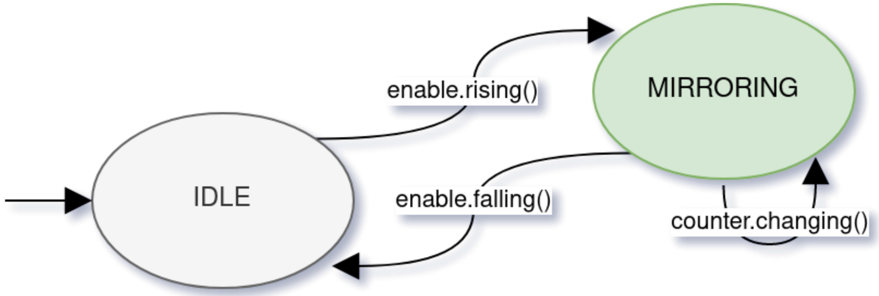
Pysmlib

- A simpler alternative to the EPICS sequencer
  - High level description of FSMs
  - Leave implementation details to the library
- Python language
  - High level language
  - Rich scientific and engineering ecosystem
  - Familiar to many new users



Example FSM

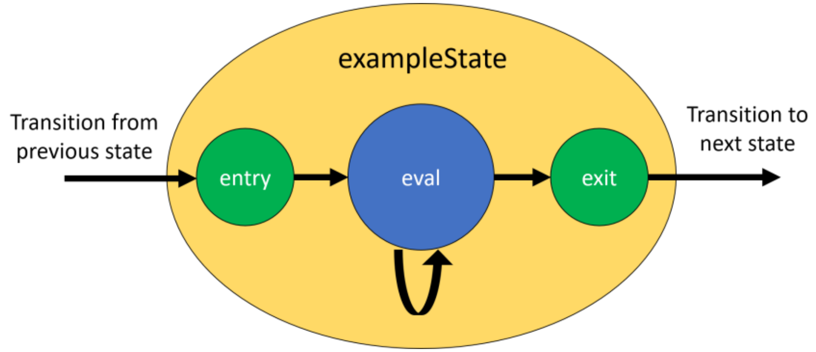
- Subclass fsmBase
  - Connect to the PVs on the constructor
- Idle state
  - Wait for enable
- Mirroring state
  - Copy the value of the *counter* PV to the *mirror* PV




```python
#! /usr/bin/python
from smlib import fsmBase, loader

# FSM definition
class exampleFsm(fsmBase):
    def __init__(self, name, *args, **kwargs):
        super(exampleFsm, self).__init__(name, **kwargs)

        self.counter = self.connect("testcounter")
        self.mirror = self.connect("testmirror")
        self.enable = self.connect("testenable")

        self.gotoState('idle')

    # idle state
    def idle_eval(self):
        if self.enable.rising():
            self.gotoState("mirroring")

    # mirroring state
    def mirroring_eval(self):
        if self.enable.falling():
            self.gotoState("idle")
        elif self.counter.changing():
            readValue = self.counter.val()
            self.mirror.put(readValue)

# Main
if __name__ == '__main__':
    # load the fsm
    l = loader()
    l.load(exampleFsm, "myFirstFsm")

    # start execution
    l.start()
```

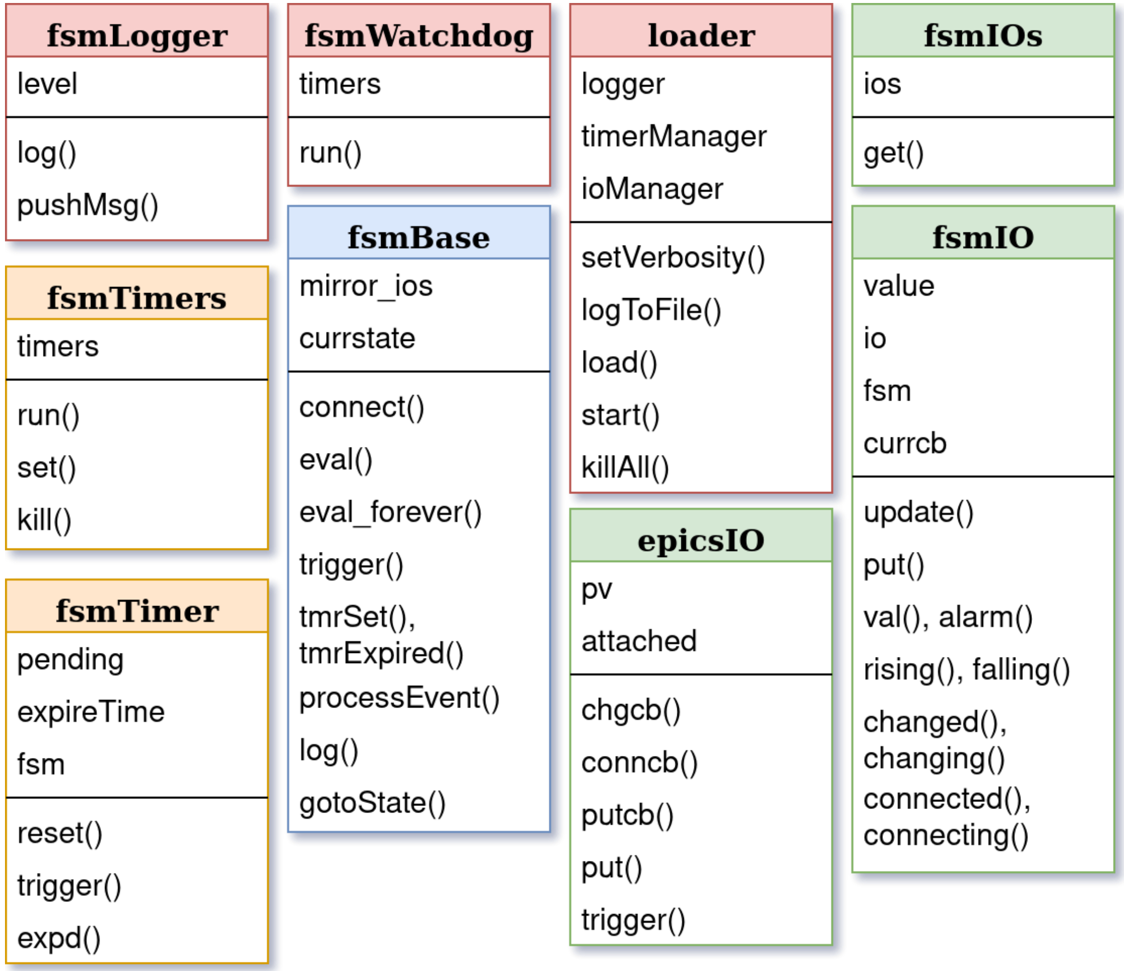
Design

- Event driven FSM
- Daemon-like execution flow
  - Concurrent execution of multiple FSMs
- Network efficiency
  - Share the Channel Access PV connections across FSMs
- Inputs should not change during the state execution
  - Each input event triggers one state execution
- Execute actions on state transitions
  - *entry*, *eval*, *exit* methods



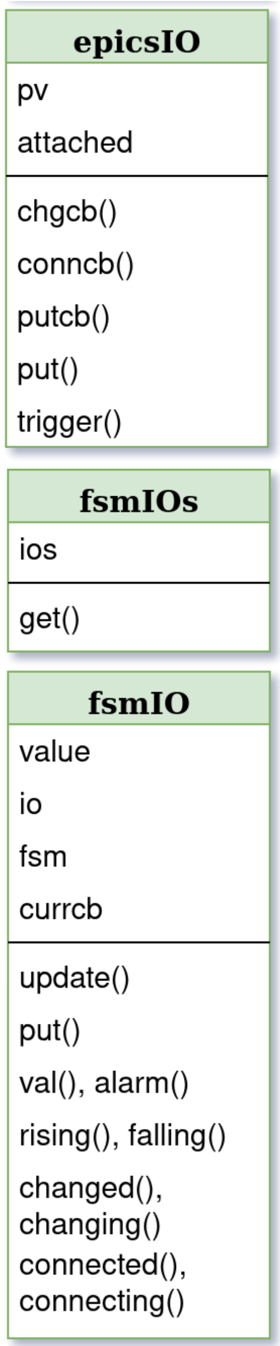
Architecture

- 4 main subsystems
  - Input management
  - FSM execution
  - Timers
  - Utilities



Input Management

- 3 event types from Channel Access
  - *change, connection, put_complete*
- One PV emits an event
  - The event data is placed on thread-safe queues
- All the FSM connected to the corresponding input are executed
  - Each one is a different thread
- Each FSM keeps a local proxy of all its inputs
  - fsmIO class
  - Updated with the data retrieved from the queue
- The current state is executed
  - The triggering event type is used to check edge conditions

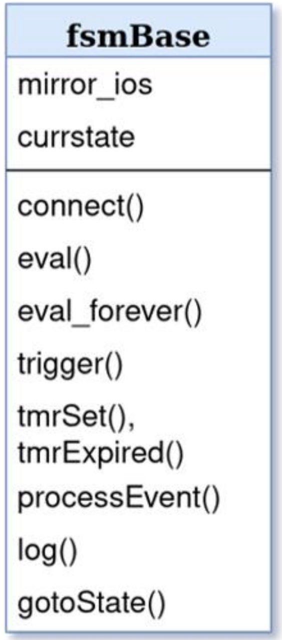


Execution Flow

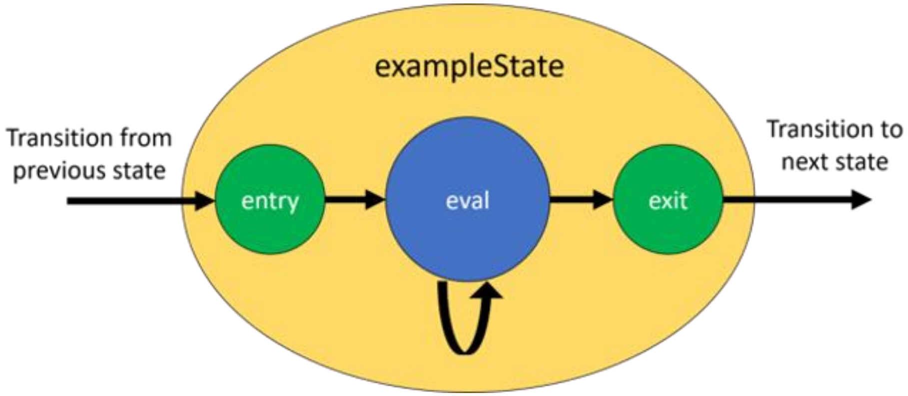
1. Perform a state transition if required. In this case it also executes the **_entry()** method of the new state, if it's defined.

2. Execute the **_eval()** method of the current state.

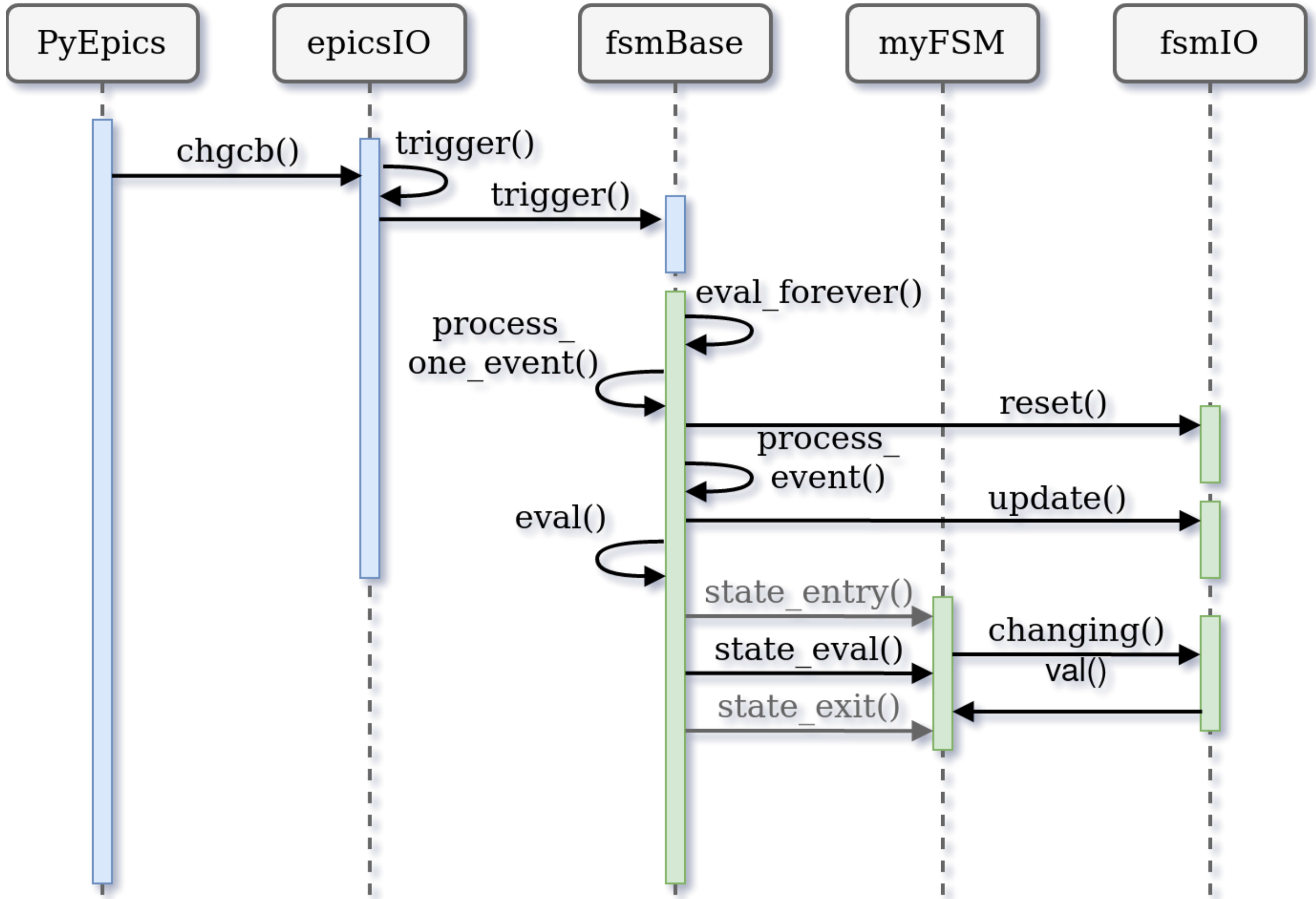
3. If the user requested a state transition, the **_exit()** method of the current state is executed. In this case go back to step 1 without processing a new event.



gotoState() automatically finds the right methods based on the state name



Change event example



Pysmlib

19 Oct 2021

Timers

- Trigger FSM execution after a fixed time delay
  - To check timeouts, perform periodic actions, wait before an action…
- Internal event of type *timer_expired*
  - A thread manages all the timers and queues events

```python
def move_entry(self):
    self.motor.put(100)                   # move the motor
    self.tmrSet('moveTimeout', 10)        # Set a timer of 10s

def move_eval(self):
    if self.doneMoving.rising():          # If the motor movement completed
        self.gotoState("nextState")       # continue to next state
    elif self.tmrExpiring("moveTimeout"): # Timer expired event
        self.gotoState("error")           # go to an error state
```

Utilities

Logger

- Unified interface to log to different backends

Loader

- Load multiple FSM on a single executable
- Share resources

Watchdog

- Specify PV as watchdog
- A thread periodically writes a value
- The PV goes into alarm if no writes occur after a delay

User Experience

- First concept in 2016 for RF control system @ LNL
- Used for many other subsystems
  - Diagnostic, ion beam sources, vacuum
- Simulators
  - Replace real devices by simulating their actions on PVs
- Alarm handling
  - Example: send notification via Telegram
- Beam Optimization Procedures
  - BOLINA
- Useful when asynchronous interaction is expected
  - Eg: user input, non-constant delays
  - Trigger on the rising or falling edges of conditions

Publishing

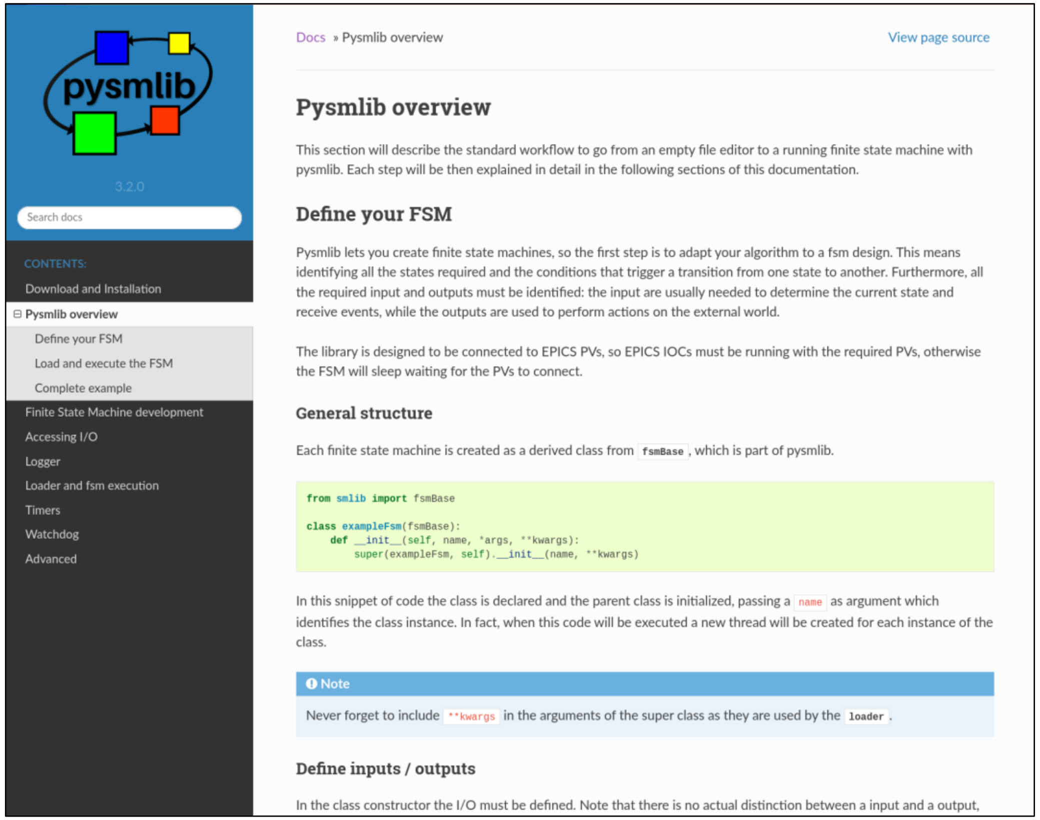
https://github.com/darcato/pysmlib

pipeline passed | coverage 63.00%

GPL-3.0 License

https://pypi.org/project/pysmlib/

https://darcato.github.io/pysmlib



DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Conclusion

- Pysmlib: A library to develop EPICS Finite State Machines
  - Focus on simplicity
  - Great expandability with Python libraries
  - Useful features for common use-cases
- Available to the whole EPICS community
  - Makes no assumption
  - Tested and running in production
- Future improvements
  - Add support for different input types (pvAccess?)
  - Contributions are welcome

Thank you

Davide Marcato