Using Lua with YARP to write state machines

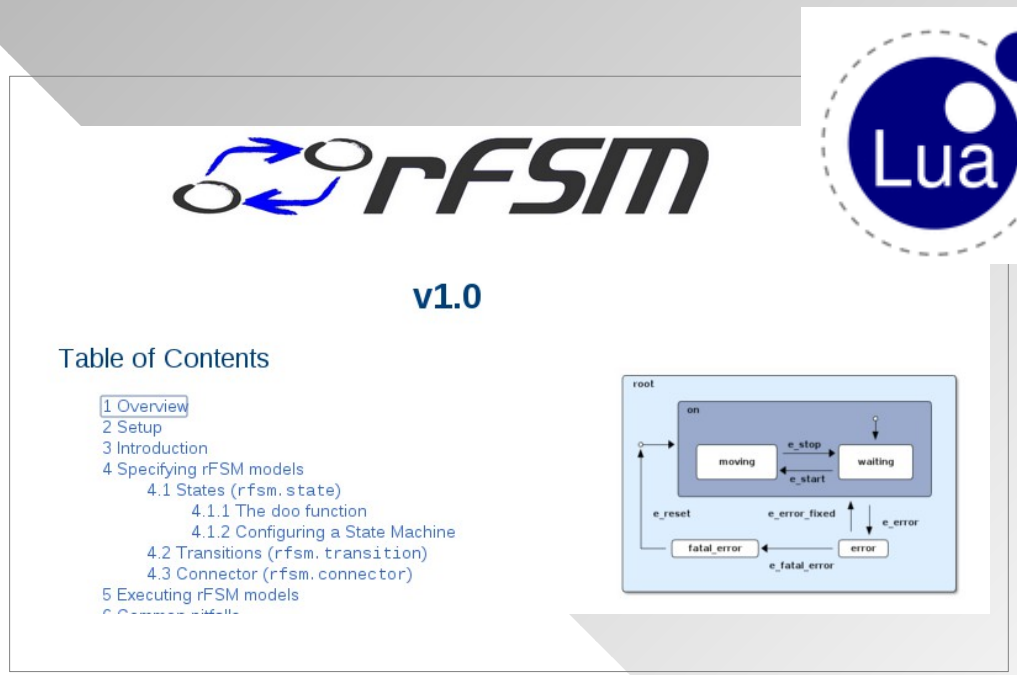
(A tutorial on using rFSM with YARP)

Ali Paikan (iCub Facility - Italian Institute of Technology)

VVV13, Veni Vidi Vici 2013, the iCub Summer School

July 22th 2013 – Sestri Levante

Why using rFSM?

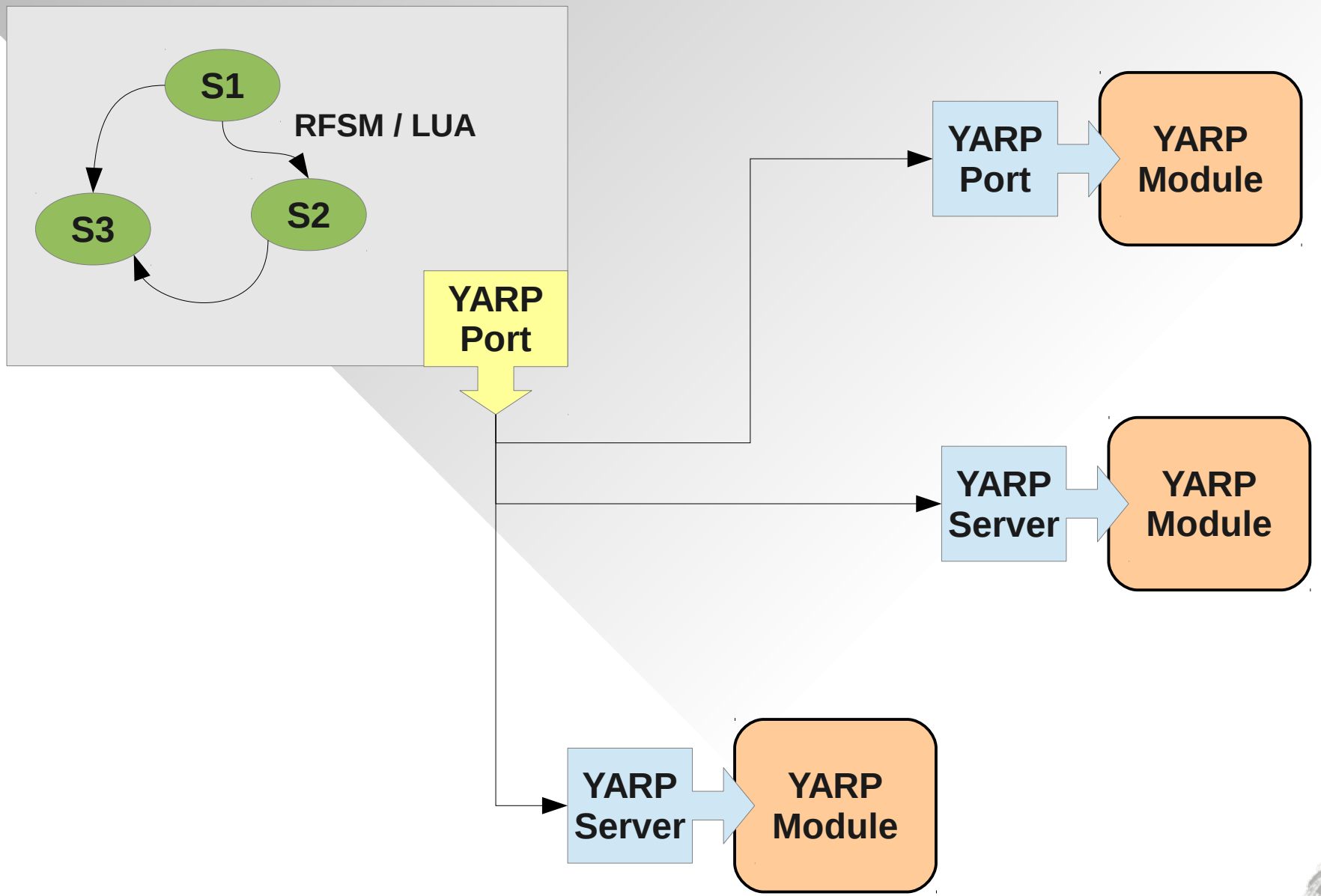


- Which implementation of state machine should I use? (e.g. based on C++, SCXML)
- rFSM is a small and powerful state machine
- rFSM is written in pure Lua
- Lua is a powerful, fast, lightweight, embeddable scripting language.
- Lua offers very simple procedural syntax

Documentation: http://people.mech.kuleuven.be/~mklotzbucher/rfsm/README.html

Download (git): https://github.com/kmarkus/rFSM

Using rFSM with YARP



Building Lua-YARP binding

```
$ cd $YARP_ROOT/bindings
$ mkdir build
$ cd build
$ ccmake ../
```

1

```
$ make
Building CXX object ...
```

3

```
CMAKE_BUILD_TYPE          Release
CMAKE_INSTALL_PREFIX      /usr/local
CREATE_ALLEGRO            OFF
CREATE_CHICKEN            OFF
CREATE_CSHARP             OFF
CREATE_JAVA               OFF
CREATE_LUA                ON
CREATE_PERL               OFF
CREATE_PYTHON             OFF
CREATE_RUBY               OFF
CREATE_TCL                OFF
SWIG_DIR                  /usr/share/swig2.0
SWIG_EXECUTABLE           /usr/bin/swig2.0
```

2

```
# add the Lua-YARP binding library to LUA_CPATH

export LUA_CPATH=";;;$YARP_ROOT/bindings/build/?.so"
```

4

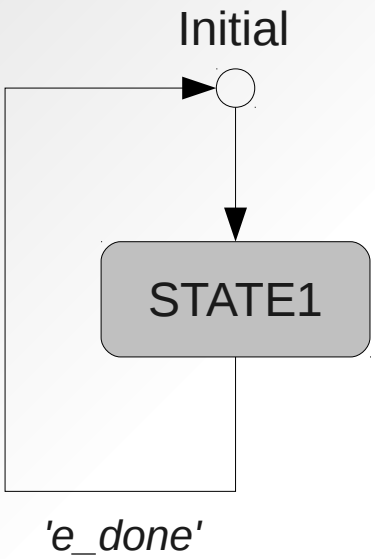
Testing Lua with Yarp

```
#!/usr/bin/lua

-- load YARP-Lua binding library (e.g. yarp.so, yarp.dll)
require("yarp")

-- initialize YARP network
yarp.Network_init()

-- create and connect the ports
sender = yarp.BufferedPortBottle()
sender:open("/sender")
yarp.NetworkBase_connect(sender:getName():c_str(), "/receiver")

-- prepare and send data
local wb = sender:prepare()
wb:addString("Hello Lua!")
sender:write()

-- finishing YARP network
sender:close()
yarp.Network_fini()
```

```
$ cd $YARP_ROOT/bindings
$ ./example2.lua
...
```

Simple rFSM structure

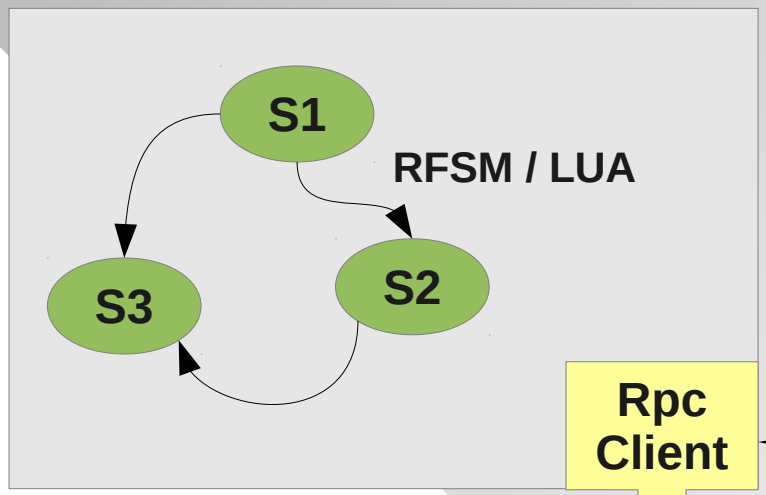
```
return rfsm.state {
    -- define a state and call it State1
    STATE1 = rfsm.state{

        entry = function()
            -- ...
        end,

        doo = function()
          while true do
            -- ...
             rfsm.yield()
          end
        end,

        exit = function()
            -- …
            rfsm.send_events(fsm, 'e_done')
        end
    },

    -- define transitions among states
    rfsm.transition { src='initial', tgt='STATE1' }
    rfsm.transition { src='STATE1', tgt='initial', events={'e_done'} }
}
```



Simple rFSM structure

```
#!/usr/bin/lua

-- load rFSM package
require("rfsm")

-- load state machine model
fsm_model = rfsm.load("./myfsm.lua")

-- initialize it
fsm = rfsm.init(fsm_model)

-- repeat and run fsm until specific condition holds
shouldExit = false

repeat
    rfsm.run(fsm)
    -- specify how fast fsm should handle the events
    -- yarp.Time_delay(0.1)
until shouldExit ~= false
```

State machine example



Rpc Client

(/fsm/commands)

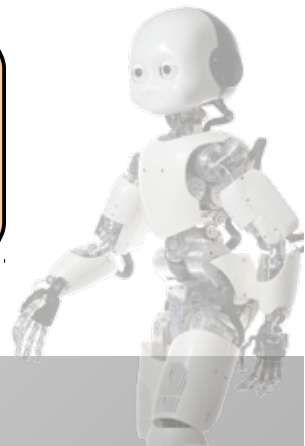
ack_done / ack_error

cmd_do

Rpc Server

YARP Module

(/Module1/event:rpc)



State machine example