Toribio

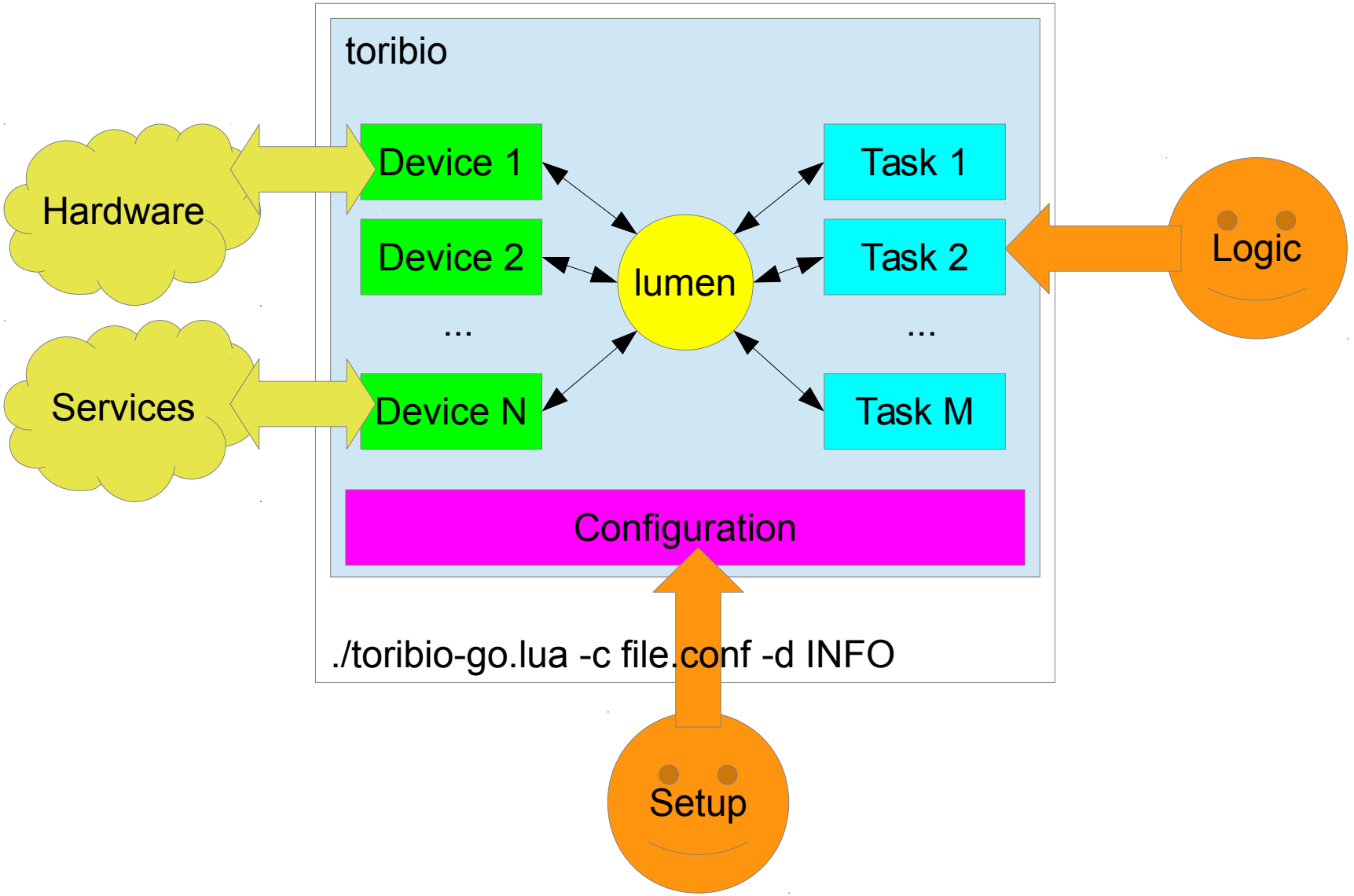
Embedded Robotics Platform

jvisca@fing.edu.uy - grupo mina / fing / udelar - 2015

Toribio

- Library for writing high level logic for robots and other control applications.
- Geared towards embedded applications.
- Written in/for Lua, a fast a small VM based language.
- Based on Lumen, a cooperative multitasking environment.
- Support POSIX platforms (Linux, BSD, etc.).
- Only external dependency is Nixio (a I/O library).
- MIT license.

Toribio


toribio
Hardware
Services
Device 1
Device 2
...
Device N
lumen
Task 1
Task 2
...
Task M
Logic
Configuration
./toribio-go.lua -c file.conf -d INFO
Setup

Toribio components

- Lumen: multitasking environment
- Devices: represent hardware&services
  - Instantiated by “Deviceloaders” tasks
- Tasks: business logic
- Configuration: deployment parameters, used to start Devices and Tasks
- Selector task: file and socket I/O (from Lumen)
- Log: logging module (from Lumen)

```
local toribio = require 'toribio'
```

Toribio Tasks

- Tasks implement business logic
- Tasks can exchange messages using the scheduler
- Compatible with Lumen tasks
- Placed in *toribio/tasks* folder.

In configuration file:

```
tasks.taskname.load=true
tasks.taskname.someparameter='text'
tasks.taskname.anotherparameter=0
```

In *toribio/task/taskname.lua*:

```
local M = {}
local sched = require 'sched'
return {
    init = function(conf)
        -- initialize stuff
        print(conf.someparameter)
        -- do something
    end
end
```

Toribio Deviceloaders

- Just like Tasks, but with the purpose of instantiating “Device” objects.
- Several already provided: dynamixel, gpsd, openmoko, mouse, accelerometer, bobot...
  - More can be created
  - Special one: filedev. Monitors the filesystem and manages devices linked to a file (e.g. /dev/tty)
- Placed in toribio/deviceloaders folder.
- In configuration file:

```
deviceloaders.dynamixel.load = true
deviceloaders.dynamixel.filename = '/dev/ttyUSB0'
deviceloaders.dynamixel.serialtimeout = 0.05
```

Toribio Devices

- Object representing a piece of hardware or service
- Devices provide an API (methods and signals)
  - *device.name*: unique name of the device
  - *device.module*: type of device (e.g. deviceloader)
  - *device.task*: may contain a tasks (to pause the device?)
  - *device.signals*: the signals emitted by this device `sched.wait({d.signals.something_happened})`
  - *device.filename*: if depends on a file, it goes here (e.g. /dev/ttyUSB0)
  - *device:register_callback*('something_happened', f)
  - More calls and attributes, implementing device specific functionality

Toribio Devices

- Devices are published in a table
- Browse all devices:

```
for name,device in pairs(toribio.devices)do
  print(name, device.module)
end
```

- Request by name

```
local om = toribio.wait_for_device('openmoko')
```

- Request by filter

```
local motor = toribio.wait_for_device({
  module = 'ax',
  filename = '/dev/ttyUSB1'
})
```

Configuration file

```
log.level.default = 'INFO'
log.level.SCHED = 'NONE'
```

Global configuration

```
deviceloaders.dynamixel.load = true
deviceloaders.dynamixel.filename = '/dev/ttyUSB0'
deviceloaders.dynamixel.serialtimeout = 0.05
```

1 deviceloader

```
tasks.rc_bot.load = true
tasks.rc_bot.ip = '*'
tasks.rc_bot.port = 9999
tasks.rc_bot.motor_left = 'ax12:3'
tasks.rc_bot.motor_right = 'ax12:12'

tasks.shell.load = true
tasks.shell.ip = '*'
tasks.shell.port = 2012
```

2 tasks

```
./toribio-go.lua -c file.conf -d NONE
```

Tips

- Think ahead: emit signals for everything useful.
- Lumen is not a RTOS.
  - Do not swamp the scheduler with signals (under 10Hz is ok)
- If a task is CPU bound, consider placing it outside and feed results to Toribio trough a socket.
- Use the logging module
- When writing a deviceloader, fill in all attributes of the devices (module and name always, events, task and filename when appropiate).
- Use the Selector module for I/O
  - Everything Is a File, right?

Tutorial, step 1

Grab events from mouse and print them

```
$ cat tutorial.conf
deviceloaders.mice.load = true
tasks.tutorial.load = true
```

```
$ cat tasks/tutorial.lua
local M = {}
local toribio = require 'toribio'
M.init = function(conf)
   local mice = toribio.wait_for_device({module='mice'})

   toribio.register_callback(mice, 'leftbutton', function(v)
          print('left:',v)
   end)
   toribio.register_callback(mice, 'move', function(x, y)
      print('move:',x,y)
   end)
end
return M
```

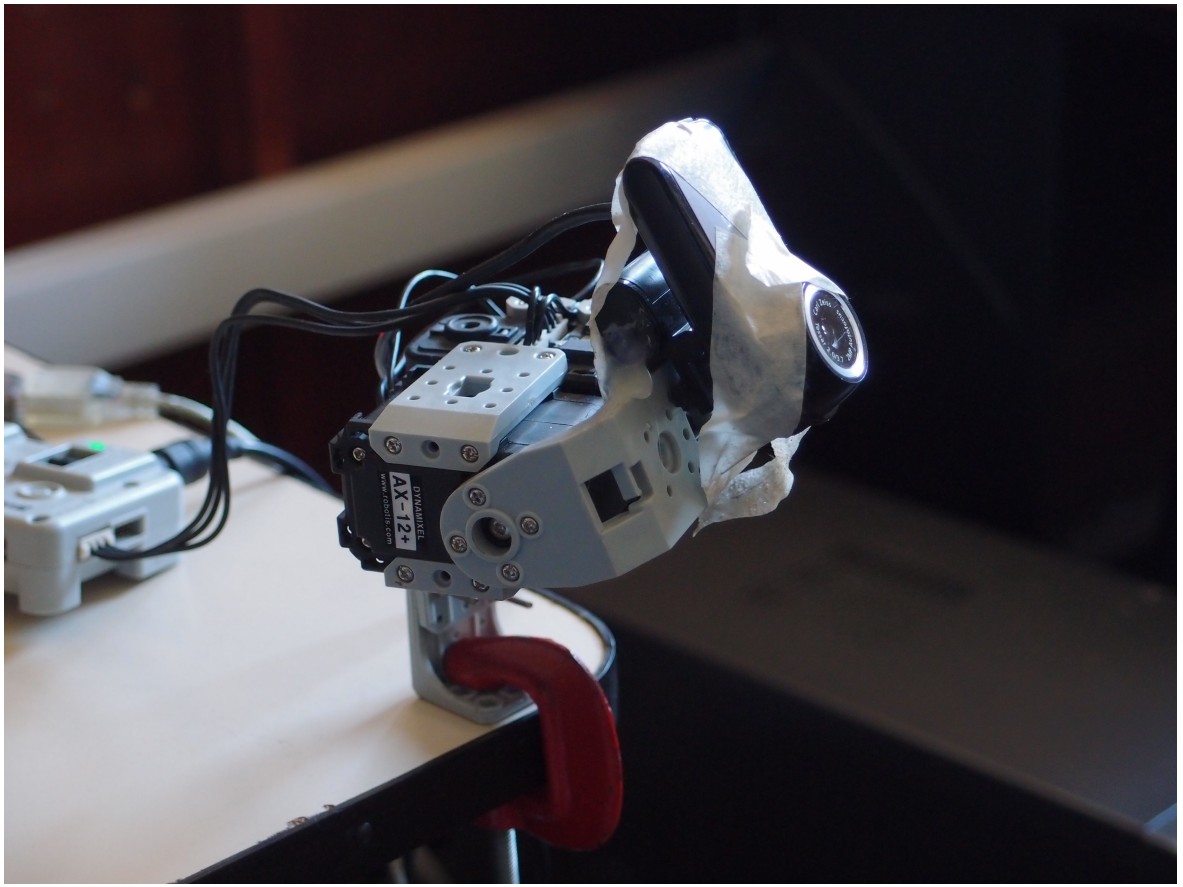
Tutorial, step 1

Grab events from mouse and print them

```
$ sudo lua toribio-go.lua -c tutorial.conf
move:  0  1
move:  0  2
move:  0  3
move:  0  4
move:  -1 4
move:  -1 5
move:  -2 5
move:  -2 6
move:  -2 7
left:  true
move:  -2 8
move:  -2 9
left:  false
move:  -2 10
move:  -3 10
$ sudo lua toribio-go.lua -c tutorial.conf -d INFO
```

Tutorial, step 2

Move motors on input events



Tutorial, step 2

Move motors on input events

```
$ cat tutorial.conf
deviceloaders.mice.load = true
deviceloaders.dynamixel.load = true
deviceloaders.dynamixel.filename = '/dev/ttyUSB0'
tasks.tutorial.load = true
tasks.tutorial.motor_pan = 'ax12:9'
tasks.tutorial.motor_tilt = 'ax12:8'
```

```
   -- somewhere inside the init() of tasks/tutorial.lua
   local mice = toribio.wait_for_device({module='mice'})
   local motor_pan = toribio.wait_for_device(conf.motor_pan)
   local motor_tilt = toribio.wait_for_device(conf.motor_tilt)

   toribio.register_callback(mice, 'move', function(x, y)
      motor_pan.rotate_to_angle( x/5 )
      motor_tilt.rotate_to_angle ( y/5 )
   end)
```

Tutorial, step 3

Have some concurrent tasks

```
-- somewhere inside the init() of tasks/tutorial.lua
local axbus = toribio.wait_for_device({module='dynamixel'})
local bcaster = axbus.get_broadcaster()

sched.run(function()
    while true do
        bcaster.set.led(true)
        sched.sleep(0.5)
        bcaster.set.led(false)
        sched.sleep(1)
    end
end)
```

Tutorial, step 3.5

Have some concurrent tasks

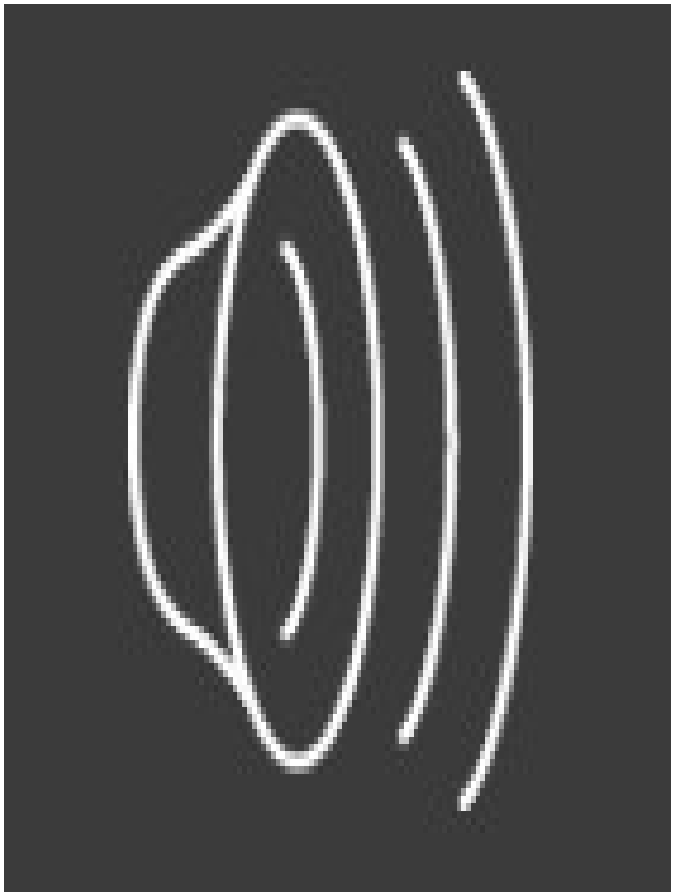
```
-- somewhere inside the init() of tasks/tutorial.lua
local axbus = toribio.wait_for_device({module='dynamixel'})
local bcaster = axbus.get_broadcaster()

sched.sigrun(
   {'blink!'},
   function(_,_,v)
      bcaster.set.led(true)
      sched.sleep(v or 0.5)
      bcaster.set.led(false)
   end
end)

--anybody can do this
sched.signal('blink!')
```

Tutorial, step 4

Grab events from several sources



Tutorial, step 5

Query bobot modules...

```
-- in config file, start bobot device
deviceloaders.bobot.load = true
deviceloaders.bobot.path = '/PATH/butia-code/bobot'
deviceloaders.bobot.comms = {"usb"}
deviceloaders.bobot.timeout_refresh = 10 --nil disables
```

```
    -- in task file, use the bobot device
    local distanc = toribio.wait_for_device('bb-distanc:0')
    sched.run(function()
        while true do
            local dist = distanc.getValue()
            if dist < 100 then
                sched.signal('avoid!', dist)
            end
            sched.sleep(0.1)
        end
    end)
```

References

- Toribio sources
  https://github.com/xopxe/Toribio
- Toribio tutorial
  https://github.com/xopxe/Toribio/blob/master/docs/1-Tutorial.md
- Lumen API
  http://xopxe.github.com/Lumen/
- Nixio:
  https://github.com/Neopallium/nixio

End