Soletta

Closing the IoT Development Gap

OpenIoT & ELC Europe 2016

ProFUSION
embedded systems

Agenda

- Who am I?
- IoT Development Gaps
- How to close IoT Development Gaps
- Soletta Overview
- Key Subsystems
- Flow Based Programming
- Developer Tools
- Future Plans

Who am I?

Gustavo Sverzut Barbieri
Computer Engineer
ProFUSION embedded systems

- Brazilian
- Software Developer since 9yo
- Working with Embedded since 2005
- Software development services
- Passionate about efficiency
- Soletta Architect & Lead Developer

IoT Development Gaps

ProFUSION
embedded systems

IoT Development Gaps

- IoT differences to traditional embedded systems
- Solutions are focused on a single subset (just hardware, just network…)
- Solutions are platform specific, no scalable solutions
- Nothing is integrated

Hard to reuse your knowledge

IoT Development Gaps: needs

- Fast development cycles
- Cover product families (MCU, gateways, multi core CPUs)
- Allow small engineering teams
- Ease choices

How to close IoT Development Gaps?

- Uniform API abstracting the multiple platforms
- 3 mains areas
  - I/O
  - Comms
  - OS services
- Easy to use API
- Scalable

Soletta Overview

ProFUSION
embedded systems

Soletta Overview

- Open Source License: Apache 2 (static linkage for small systems)
- Real Open Source Development Model @ GitHub
- Portable code: multiple OSes from day-0

  Linux | Linux Micro (PID1) | Contiki | RIoT | Zephyr

- Many supported boards & easily extensible to add more

  Intel Edison | Intel Galileo | Intel MinnowBoard MAX | Arduino 101 | Atmel SAMR21 XPro | Raspberry Pi

- Scalable yet easy to use: Object-Oriented C code
- Event-Driven Programming: abstracts OS specifics from user
- Modular: use only what you need

Soletta Subsystems

I/O | Comms | OS Services | Persistence | Machine Learning

Logging | Main Loop | Parsers | Data Types | Worker Threads | Crypto

ProFUSION embedded systems

Soletta Input/Output Subsystem

- Low-level: GPIO, AIO, I2C, SPI, UART, PWM
- High level: Sensors and Actuators
  - Linux uses IIO (Industrial Input/Output)
  - Zephyr will use sensor subsystem (TODO)
- OS specifics are abstracted via main loop - no ISR or threads are exposed

Mantra “implement drivers where they belong: IN THE KERNEL”

Soletta Communications Subsystem

- MQTT
- HTTP server & client
- LWM2M
- OIC/OCF
- CoAP
- BLE

Mantra “choose wisely & integrate well”

Similar APIs should feel the same

APIs should be implementable everywhere

Soletta OS Services Subsystem

- Software Update (check, fetch, apply)
- Start, Stop & Monitor services (ie: bluetooth)
- Power supply enumeration & monitoring
- Poweroff, Reboot, Suspend, Enter Rescue mode…
- Network Connection Manager

Soletta Other Subsystems

- Data Types: list, array, buffers and slices
- Logging: with domains, thread-safe and can be compiled-out
- Parsers: JSON based on string slices (no memory allocation)
- Persistence: File, EFIVars and EEPROM with compile-time defined structure
- Worker Threads: low priority preemptible threads
- Crypto: Certificates, Message Digest and Encryption (TODO)
- Machine Learning (SML): Fuzzy & Neural Network made easy to use

How to close IoT Development Gaps?

checklist

- ✓ Uniform API abstracting the multiple platforms
- ✓ 3 mains areas
  - I/O
  - Comms
  - OS services
- ? Easy to use API
- ✓ Scalable

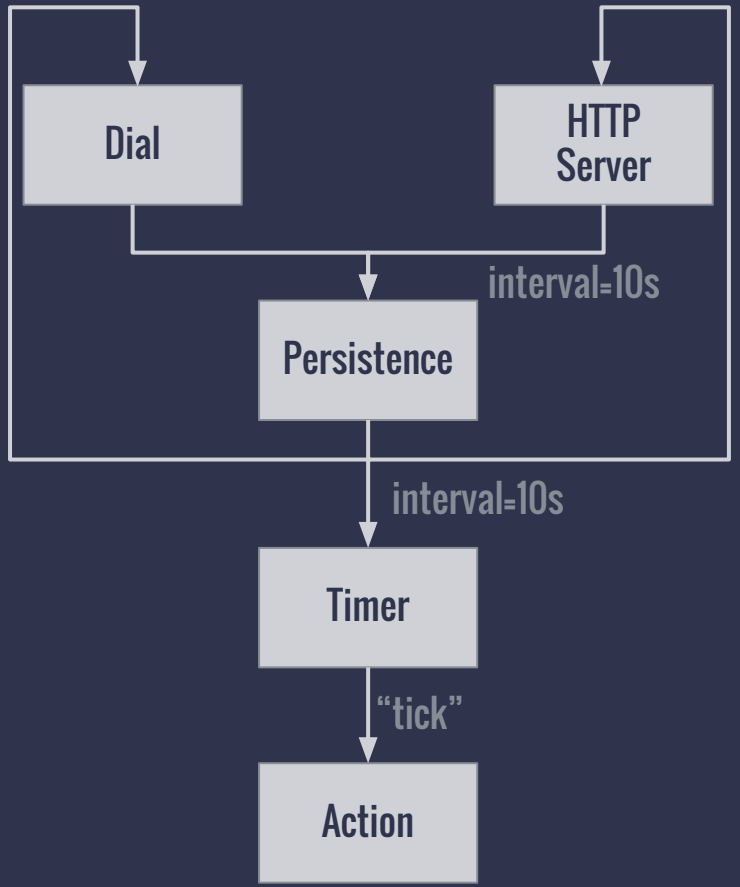
most users don’t get callbacks

Leaks & SEGV

boring pattern “on event, get data”

Flow Based Programming
FBP

or how did we avoid
callbacks and memory management
for our users
making their lives easier



FBP

- Invented by J. Paul Morrison in the early 1970s http://www.jpaulmorrison.com/fbp
- Components are Black Boxes with well defined interfaces (Ports)
- Focus on Information Packets (IP)
- Started to gain traction in Web:
  - NoFlo
  - Facebook Flux
  - Google TensorFlow
  - Microsoft Azure Event Hubs
- Also on Embedded Systems:
  - ROS
  - MicroFlo
  - NodeRED
- Also on Multimedia:
  - V4L
  - Gstreamer
  - Apple Quartz

FBP: Nodes as Black Boxes

- Simple interface
- Low (no?!) coupling, allows replacing components
- Easy to optimize code size by removing unused ports
- Parallelization
- Isolation (including processes)
- Internally can use Event-Driven Programming (Main Loop), Threads...

Users only manage connections.

Everything else is done by the FBP core or the components

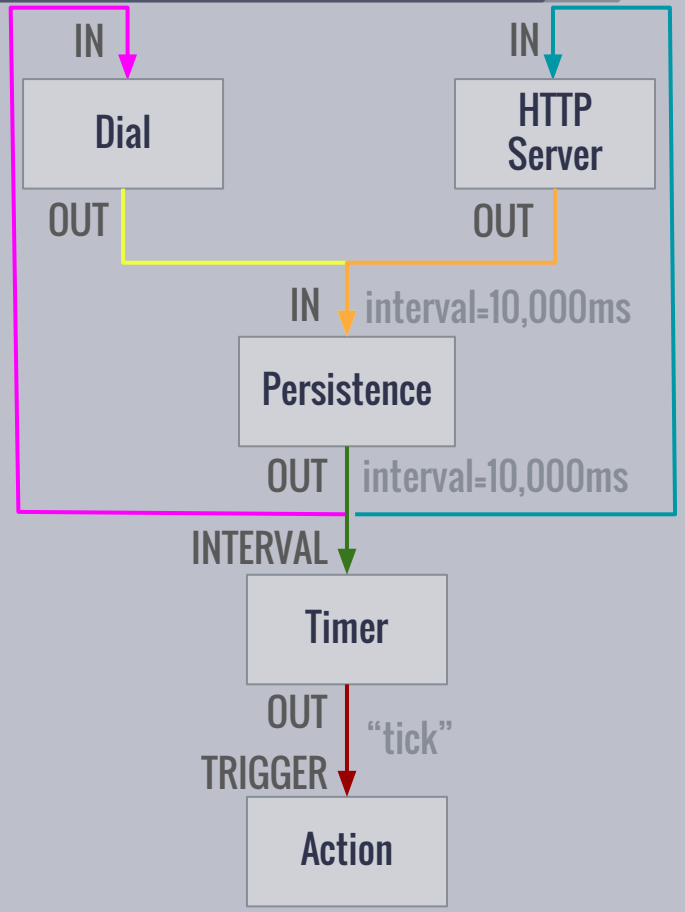
FBP: Example

```
# Create Instances (timer is in milliseconds!)
dial(my_dialer_type)
http_server(http-server/int:url="/timeout_ms")
persistence(persistence/int:name="timeout_ms",
     storage="fs",default_value=10000)
timer(timer)
action(my_action_type)

# Connect Instances
dial        OUT -> IN persistence
persistence OUT -> IN dial

http_server OUT -> IN persistence
persistence OUT -> IN http_server

persistence OUT -> INTERVAL timer
timer       OUT -> TRIGGER  action
```



FBP: Pros & Cons

Cons:

- Paradigm shift
- Although small, still adds overhead compared to carefully written C code
- Requires “bindings” (node type module) to use 3rd party libraries
- Needs balance on what to write as FBP and what to create custom node types

Pros:

- No leaks or SEGV, reduced blaming!
- Simple interface (nodes & ports) eases team collaboration
- Easy to read, write and visualize, aids communication with customers & designers
- Super fast prototyping & testing

FBP: show me the size!

- Intel Quark SE DevBoard
- Zephyr OS
- Soletta
- FBP using OIC/OCF light server
  - IPv6
  - OIC/OCF (UDP + CoAP + CBOR)
  - GPIO
- Auto-generated code from FBP

Flash - Kb

107

RAM Peak - Kb

32

Developer Tools

ProFUSION
embedded systems

Developer Tools - code generators

`sol-oic-gen.py`
generates node types C code from OIC/OCF JSON specs

`sol-flow-node-type-gen.py`
generates node types C boilerplate from JSON specs

`sol-fbp-generator`
generates C from FBP

Less manual work

Less errors

Easier migration to new APIs

Ease of use with no runtime overhead

Developer Tools - DevApp

- Web-based IDE using node.js and angular.js
- Can be executed on target (on-board development - Linux)
- Systemd journal viewer
- Built-in documentation
- Text Editor with syntax highlight and code completion
- FBP runner, inspector and viewer (Graphviz)
- Try Soletta without installing it! All you need is a browser and an SD/USB drive

https://github.com/solettaproject/soletta-dev-app

EDITOR JOURNAL CHEAT SHEET

File Edit View

INSPECTOR CONFIGURATION

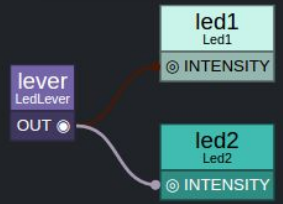
Projects

- demo
  - minnow-calamari
    - sol-flow.json
    - sol-flow-intel-minnow-max-linux_gt_3_
    - calamari-rgb-led.fbp
    - calamari-lever.fbp
    - calamari-led.fbp
    - calamari-buttons-rgb-led.fbp
    - calamari-button-accumulator-persiste
    - calamari-7seg-value.fbp
    - calamari-7seg-segments.fbp
    - README
  - grove-kit
    - sol-flow-intel-galileo-rev-g.json
    - sol-flow-intel-edison-rev-c.json
    - sol-flow-grove-sound-sensor.json
    - sol-flow-grove-relay.json
    - sol-flow-grove-button.json
    - lcd
    - grove-temperature-sensor.fbp
    - grove-sound-sensor.fbp
    - grove-rotary-angle-sensor.fbp
    - grove-relay.fbp
    - grove-light-sensor.fbp
    - grove-led-wave-generator.fbp
    - grove-led-accumulator.fbp
    - grove-buzzer.fbp
    - grove-button.fbp
    - README

Code Viewer : calamari-led.fbp

```
1  #!/usr/bin/env sol-fbp-runner
2  #
3  # This example links the Calamari lever (SPI) to intensity light of
4  # Led1 and Led2 (PWM leds), as defined in sol-flow.json in this
5  # directory.
6
7  lever(LedLever) OUT -> INTENSITY led1(Led1)
8  lever OUT -| INTENSITY led2(Led2)
9
```

Expected '->' between connection statement. e.g. 'node(nodetype) OUTPUT_PORT_NAME -> INPUT_PORT_NAME node2(nodetype2)'



FBP Running Status: .grove-lcd-fade.fbp - inactive (dead)

Syntax Highlight and as-you-type error checking

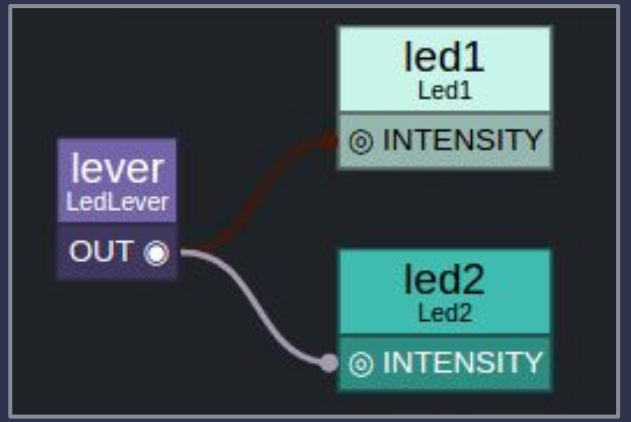
```
lever(LedLever) OUT -> INTENSITY led1(Led1)
lever OUT -| INTENSITY led2(Led2)
```

Expected '->' between connection statement. e.g. 'node(nodetype) OUTPUT_PORT_NAME

systemd journal log viewer

Date	Unit	Message
20-05-2016 15:46	ntpd	Soliciting pool server 198.55.111.50
20-05-2016 15:46	ntpd	Soliciting pool server 104.232.3.3
20-05-2016 15:46	ntpd	Soliciting pool server 2001:67c:1560:8003::c7
20-05-2016 15:46	ntpd	Soliciting pool server 52.0.56.137
20-05-2016 15:46	ntpd	Soliciting pool server 129.250.35.251
20-05-2016 15:46	systemd	Stopped Run FBP Script when using Soletta Devapp in web browser.
20-05-2016 15:46	systemd	Stopping Run FBP Script when using Soletta Devapp in web browser...
20-05-2016 15:46	polkitd(authority=local)	Operator of unix-session:c2 successfully authenticated as unix-user:botta name::1.1112 [systemctl stop fbp-runner@-tmp-singlesession-fbp_run.en
20-05-2016 15:46	sol-fbp-runner	output Hello World! (string)
20-05-2016 15:46	systemd	Started Run FBP Script when using Soletta Devapp in web browser.
20-05-2016 15:46	systemd	Stopped Run FBP Script when using Soletta Devapp in web browser.
20-05-2016 15:46	polkitd(authority=local)	Operator of unix-session:c2 successfully authenticated as unix-user:botta name::1.1091 [systemctl restart fbp-runner@-tmp-singlesession-fbp_run.e

On-the fly FBP visualization using graphviz



Future Plans

Contributions are welcome!

- More Node.JS bindings
- Python bindings
- Fancier FBP Web Inspector
- Visual Editor
- DevApp generating firmware images
- FBP meta-type for LWM2M, OIC and BLE
- FBP statically linking disk size optimizations
- Use mempools for fixed size objects
- Port to ESP8266

Thank You!

Questions?

Gustavo Sverzut Barbieri
<barbieri@profusion.mobi>

github.com/solettaproject/

Want to know more about
FBP
Flow Based Programming?

See my other talk:

Flow Based Programming Applied to IoT Development

October 11th at 17h10