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?

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

Gustavo Sverzut Barbieri
Computer Engineer
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
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
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

- 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
# 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
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
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

<table>
<thead>
<tr>
<th>Date</th>
<th>Unit</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-05-2016</td>
<td>npd</td>
<td>Soliciting pool server 198.55.111.50</td>
</tr>
<tr>
<td>20-05-2016</td>
<td>npd</td>
<td>Soliciting pool server 104.232.3.3</td>
</tr>
<tr>
<td>20-05-2016</td>
<td>npd</td>
<td>Soliciting pool server 52.c.0.56.137</td>
</tr>
<tr>
<td>20-05-2016</td>
<td>npd</td>
<td>Soliciting pool server 129.250.38.251</td>
</tr>
<tr>
<td>20-05-2016</td>
<td>systemd</td>
<td>Stopped Run FBP Script when using Soletta Devapp in web browser.</td>
</tr>
<tr>
<td>20-05-2016</td>
<td>systemd</td>
<td>Stopping Run FBP Script when using Soletta Devapp in web browser.</td>
</tr>
</tbody>
</table>
| 20-05-2016 | polkitd(authentication) | Operator of unix-session c2 successfully authenticated as unix-user:bot name:1.11112
| 20-05-2016 | sol-ftp-runner| output Hello World! (string)                                            |
| 20-05-2016 | systemd       | Started Run FBP Script when using Soletta Devapp in web browser.       |
| 20-05-2016 | systemd       | Stopped Run FBP Script when using Soletta Devapp in web browser.       |

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?

Want to know more about FBP
Flow Based Programming?

See my other talk:
Flow Based Programming Applied to IoT Development
October 11th at 17h10