Open DC Grid Project

2020 August

James Gula - jlgula@papugh.com
Martin Jäger – martin@libre.solar
Chris Moller – chris.moller@evonet.com

Agenda

- Communications Overview
- IOT Stack – OFC and the Angaza Nexus Channel
  - Chad Norvell mailto:chad@angaza.com
- Open PAYGO Link – Solaris
  - Daniel Nedosseikine daniel@solarisoffgrid.com
- ODG Simulation Platform
- Related Standards / Industry Developments
- Next Meeting / Feedback

Communications - Applications

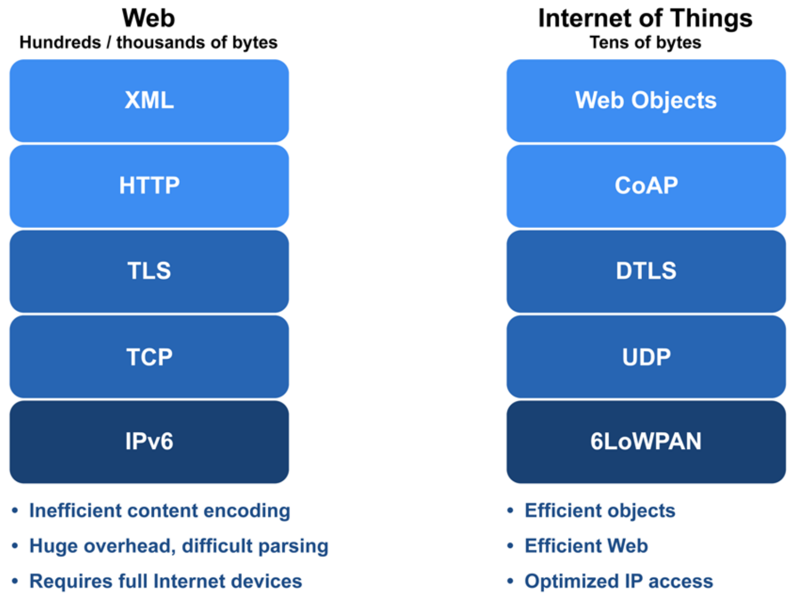
- Grid management
  - Route energy / power
  - Isolate faults
  - Grid configuration and monitoring
- Bus management
  - Allocate power from sources
  - Allocate power to loads
  - Sequence power on startup
- Device management
  - Device status - fridge temp
  - Device functions – dim a light
  - PAYGO – pass token

Communications - Constraints

* Cost
  * Common use cases very price sensitive
* Ease of use
  * Most functions must be plug and play
  * Minimal training
  * Tech support may not be available
* Security – as needed
  * Probably not needed for wired comm in home
  * Probably is needed between customers / wireless
* Interoperability – as needed
  * Many use cases have no Internet access
  * Businesses may need remote access to minimize travel
* Stability
  * Must preserve user investments – backwards compatibility
  * Potentially no opportunity for firmware upgrades
* Ease of implementation
  * Use existing open source code whenever possible
  * Easy to understand paradigms
  * Offer reference code
* Free Access
  * No patent licenses
  * Minimal dependence on purchased standards

Communications - Layers

* Multiple physical layers
  * ODGTalk for low cost
  * G3 PLC for long distance
  * CAN for performance
  * USB-PD, POE etc
* Routing only when needed
* Security only when needed
* Favor REST paradigm
  * CoAP with extensions



Communications – Presentation and Application

- Existing Models
  - Modbus etc – predefined registers with vendor extensions
  - ThingSet – JSON tree with CBOR, CoAP subset
  - Open Connectivity Foundation – JSON core
  - IEEE P2030.5 (SEP 2.0) – XML over CoAP
  - ISO etc etc
- Requirements (from ThingSet)
  - Flexible – independent of lower layer protocols
  - Compatible – easy to integrate with existing – CoAP etc
  - Human readable – text option
  - Compact footprint – code and message size
  - Schema-less and self explaining
  - Stateless
- Consistent mapping whenever practical

Angaza Nexus Channel / Core

See Angaza Presentation…

Solaris OpenPAYGO / Link

See Solaris Presentation…

ODG Simulation Platform Overview

- What is being simulated: connected devices
  - Communications message traffic
  - Power flow with energy storage
- Why
  - Easy debugging with repeatable test environment
  - Smooth transition from rich platform to constrained
  - Test harness: simulator can interact with live devices

ODG Simultation Platform Logical Architecture

* Simulates entire Grid: [{ Device}, {Bus}]
* Device: [{ Task }, { Connection}, { Port }]
* Task: { ConnectionPoint }, Port is subclass of task
* Connection: [[Task, CP], [Task, CP]]
* Bus: { [Device, Port]}

Note: energy and power are properties of devices, ports, buses

ODG Simulation Platform Execution Architecture

* Local – all devices in same app
  * Synchronous: grid invokes all tasks via clock tick
  * Async: tasks run in separate threads in real time
* Distributed (async only) – devices in separate apps, PCs, IOTs
  * Communicate via internet messages (UDP)
  * Potential bridge to other buses: LIN, CAN
* Programming platform choices:
  * JVM – tasks, simulator in Java, Scala, sync or async
  * Native – tasks, simulator in C, C++
    * Sync or async: Static link tasks to simulator app
    * Async: tasks running in Zephyr native POSIX
    * Async: STM32 etc in QEMU/Zephr
    * Async: live devices via internet / bridge
  * Browser / javascript (via Scala to Javascript translator)

ODG Simulation Platform "Operating System"

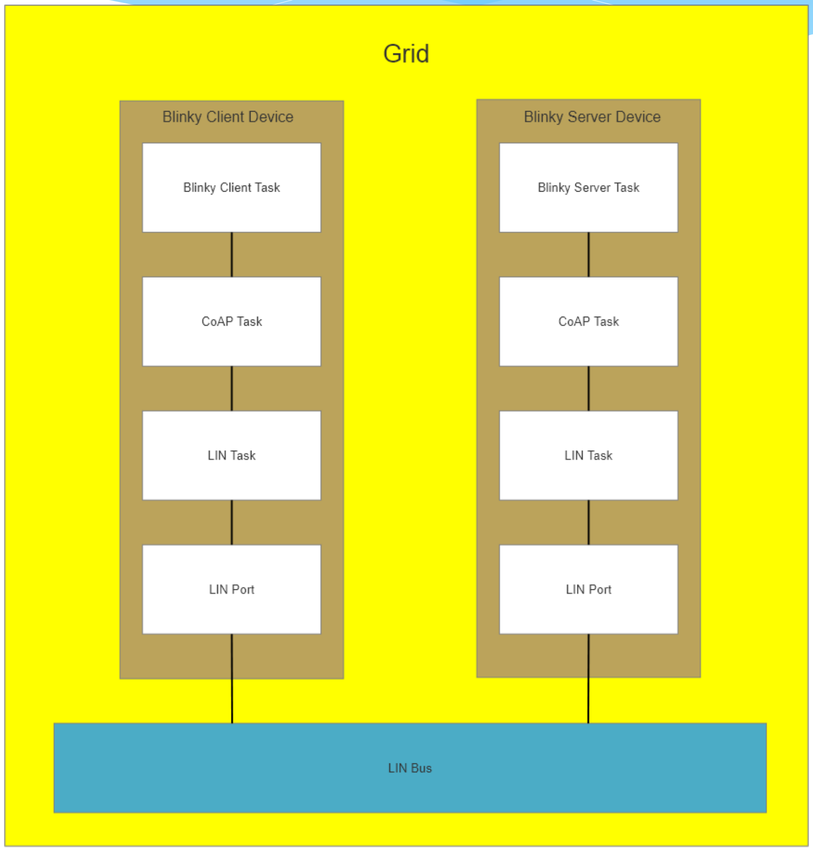
- Execution environment for tasks
- Basic functions
  - Allocate and send messages to other tasks (or bus ports)
  - Initiate / cancel timers
  - Basic info: time, configuration, deviceID etc
- Easily emulated on many platforms
  - Bare "iron" eg STM8
  - Zephyr
  - Java Virtual Machine (JVM)
  - Posix / native
  - Browser / Javascript / Node.js

ODG Simulation Platform
Key concept: task

- What is a task (aka actor)?
  - Thread-safe event queue
  - Single threaded dispatch method
  - State structure
- To thread or not to thread…
  - Synchronous – single thread runs all tasks in entire grid
    - => tasks are just state machines – no sleeps
  - Async – tasks with threads (even multiple) are OK
    - Java CoAP etc can be packaged as Task
- How does it sleep?
  - Task is runnable if anything in its event queue
  - Dispatches in a loop until queue empty (limits for errors)
  - Potential events: net messages, timer events

ODG Simulation Platform
Blinky Grid Simulation

- Structure statically defined
  - As code for testing
  - As .json file for simulation
- Run Options:
  - Sync: grid.runTicks(*n*)
  - Async: grid.run()
- LIN / UART Simulation
  - Sync: internal messages
  - Async: Multicast UDP



ODG Simulation Platform
Example: net blinky client task

Note: assumes CoAP task deals with retries

```
enum blinky_state{
  WAIT_FOR_TIMER,
  WAIT_FOR_GET,
  WAIT_FOR_SET,
  FAILED
};

struct {
  event_t *next;
  event_type type;
  union {
    net_message_t *message;
    const char *time_cookie;
  } data;
} event_t

struct {
  event_queue_t event_queue;
  system_t system;
  blinky_state state;
  net_message_t *message;
} blinky_state_t;

void blinky_init(blinky_state_t *state, system_t *system) {
  state->state = WAIT_FOR_TIMER;
  state->system = system;
  state->message = (*(system->allocate_message)())
  (*(system->create_timer))(state, (*(system->time)()), null))
}
```

```
void blinky_dispatch(event_t *event, blinky_state_t *state) {
  bool light_on = false;
  system_t *system = state->system
  switch(event->type) {
    case TIMER:
      (*(system->send))(state, my_port, blinky_format_get(state->message));
      state = WAIT_FOR_GET;
      break;
    case MESSAGE:
      net_message_t *message = event->data.message;
      if (message_parse_response_code(message) == FAILED) {
        state->state = FAILED;
        return;
      }
      switch(state.state) {
        case WAIT_FOR_GET:
          light_on = blink_parse_get_value(message);
          (*(system->send))(state, my_port, blinky_format_put(message, !light_on));
          state->state = WAIT_FOR_PUT;
          return;
        case WAIT_FOR_PUT:
          (*(system->create_timer))(state, time_add_usec((*(system->time)()), TIMER_DELAY_USEC), null))
          state->state = WAIT_FOR_TIMER;
          return;
      }
  }
}
```

ODG Simulation Platform Example: net blinky server task

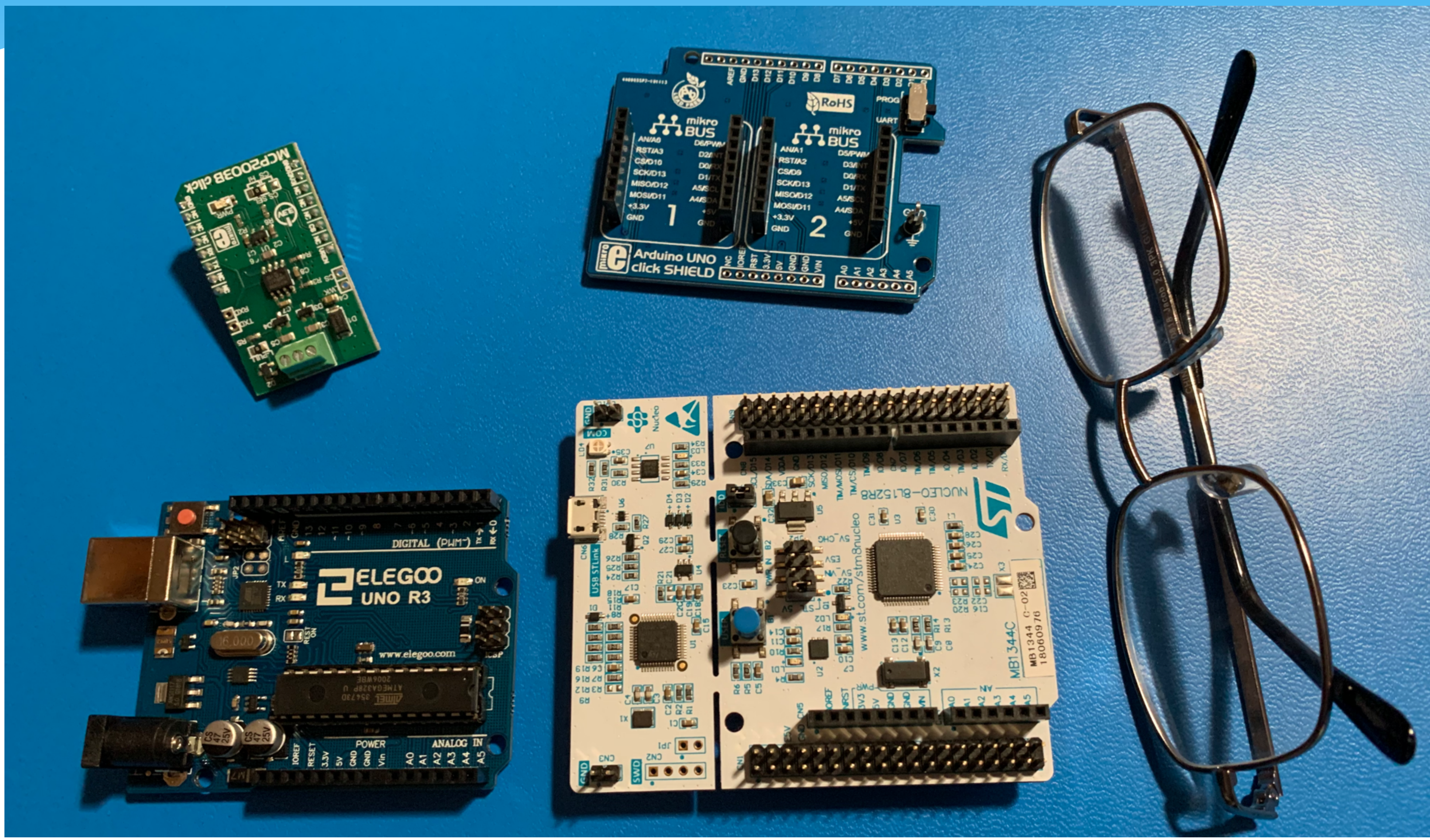
Implementation in Scala

```scala
case class BlinkyServerState(override val system: System, var lightOn: Boolean = false) extends TaskState(system)

case object BlinkyServerTask extends Task( name = "BlinkServer") {
  override def initialize(system: System): TaskState = BlinkyServerState(system)
  override def dispatch(event: TaskEvent, state: TaskState): Unit = dispatch(event, state.asInstanceOf[BlinkyServerState])
  def dispatch(event: TaskEvent, state: BlinkyServerState): Unit = event match {
    case TaskEvent.Message(_, buffer) if parseOperation(buffer) == Operation.Get => sendResponse(state, buffer)
    case TaskEvent.Message(_, buffer) if parseOperation(buffer) == Operation.Put =>
      state.lightOn = parseValue(buffer)
      sendResponse(state, buffer)
    case e: TaskEvent => super.dispatch(event, state)
  }

  def parseOperation(buffer: NetBuffer): Operation = Operation.parse(buffer.data(0))
  def parseValue(buffer: NetBuffer): Boolean = buffer.data(1) != 0
  def sendResponse(state: BlinkyServerState, buffer: NetBuffer): Unit = {
    buffer.reset()
    buffer.putByte(CoAPResponse.OK.value)
    buffer.putByte(if (state.lightOn) 1 else 0)
    state.system.send( task = this, this.connectionPoints.head, buffer)
  }
}
```

ODG Simulation Platform Hardware Lab



Related Standards / Industry Developments

- P2030.10
  - Ballot in progress
- P2030.10.1
  - Draft 3 released
    - No functional differences
    - Significant editing and clarification
- GOGLA Interop activities
  - ODG to present in September 3rd? meeting
- OpenPAYGO Link
- Angaza Nexus Channel / Nexus Channel Core
- Open Connectivity Foundation / IoTivity

Next Meeting / Feedback

- Next Meeting
  - 8 September 2020 – 1400 UTC
  - Zoom – Meeting ID 87518284403
- Sharing Portals
  - Web site: https://open-dc-grid.org/
  - GitHub: https://github.com/open-dc-grid
- Feedback?