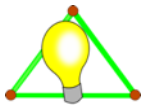
Open DC Grid Project

2022 January

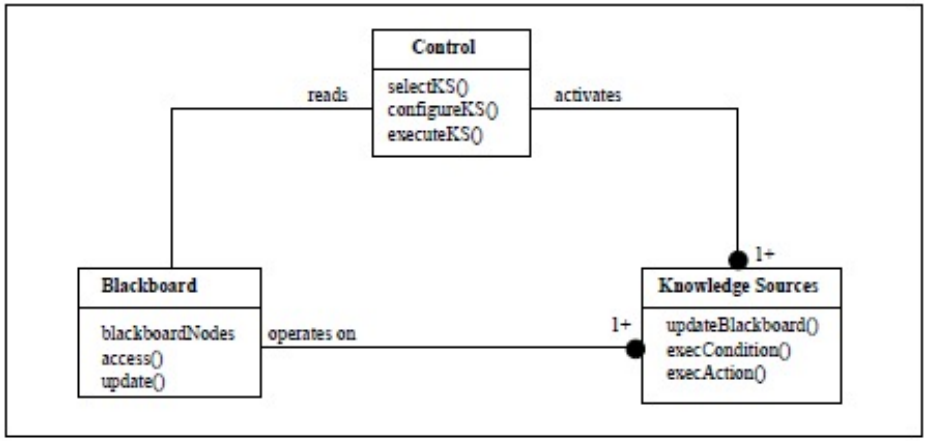


James Gula - jlgula@papugh.com
Martin Jäger – martin@libre.solar
Chris Moller – chris.moller@evonet.com

Agenda

* Polaris Status Report
* Related Standards / Industry Developments

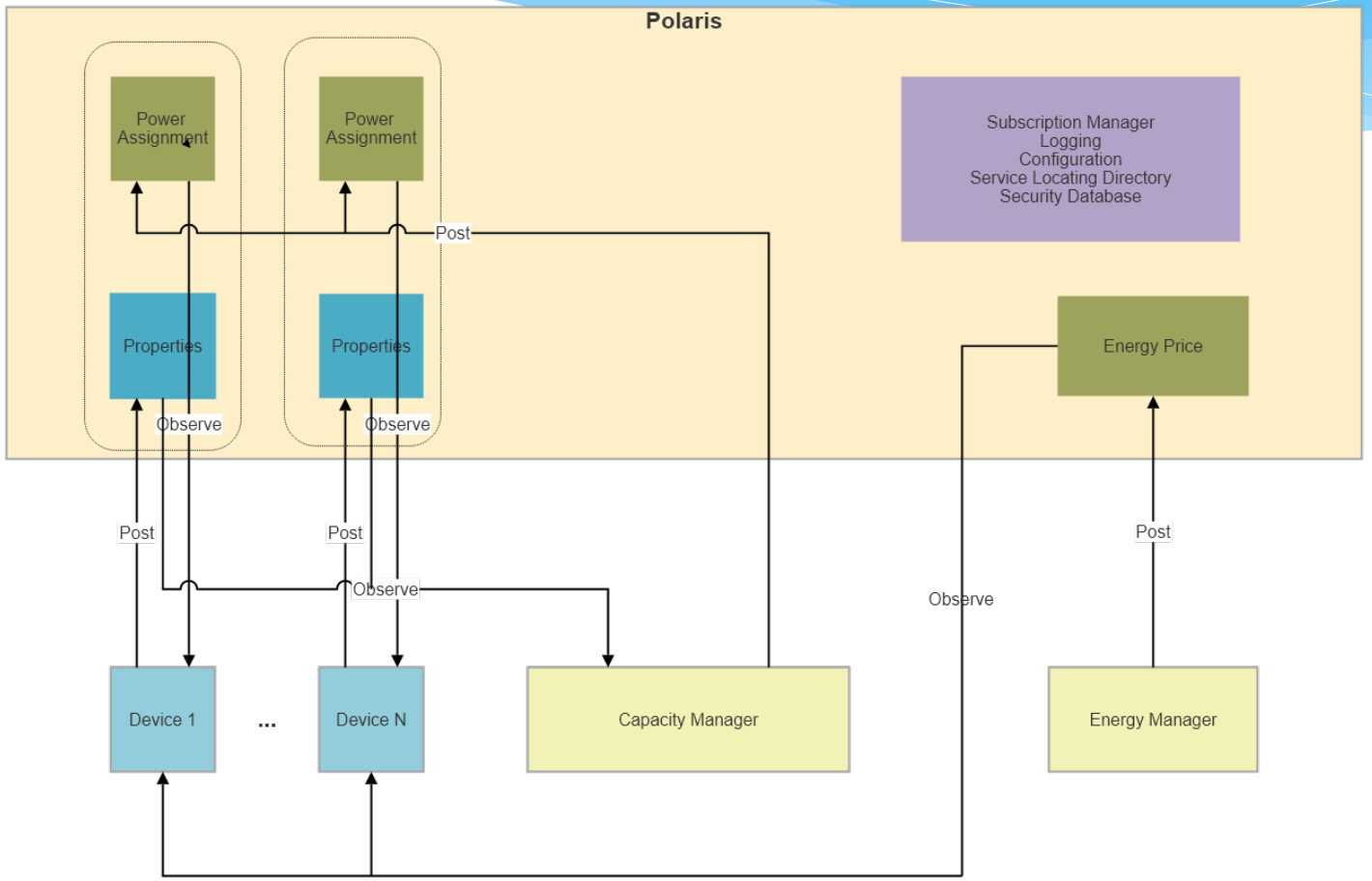
Polaris General Architecture



Wikipedia: Blackboard (design pattern)

* Blackboard design pattern – REST messages (Get/Put/Post/Delete)
  * Observable entities
  * Subscribe/notify via messages
  * Devices put/post status, observe commands
  * Control functions observe devices, put/post commands
* Plumbing
  * Service location
  * Security
  * Logging, configuration etc.

Polaris Block Diagram



Polaris OpenAPI
(YAML Fragments)

```yaml
openapi: 3.0.0
servers:
  - url: http://localhost:8080/v1
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/jlgula/Polaris/1.0.0
info:
  version: 1.0.0
  title: Polaris
  description: Microgrid controller
  contact:
    email: jlgula@papugh.com
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
tags:
  - name: device
    description: Devices managed by the controller
  - name: subscription
    description: Subscriptions that implement subscribe/notify
  - name: GC
    description: Operations the affect the grid controller as a whole.
paths:
  /gc/reset:
    post:
      tags:
        - GC
      summary: Resets the GC
      description: Removes all content and resets the GC.
      operationId: reset
      x-scala-package: gc
      responses:
        "201":
          description: Reset successful
          content:
            text/plain:
              schema:
                type: string
```

```yaml
/devices/{id}:
  get:
    tags:
      - device
    summary: Gets the device properties
    description: |
      Gets the properties of the device as an entity.
    operationId: getDevice
    x-scala-package: device
    parameters:
      - name: id
        in: path
        schema:
          type: string
        required: true
    responses:
      "200":
        description: Get successful
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Device"
      "404":
        description: Not found - the id does not corresponed to a registered device
        content:
          text/plain:
            schema:
              type: string
```

Polaris OpenAPI (Schema)

```yaml
components:
  schemas:
    Device:
      type: object
      required:
        - name
        - id
      properties:
        id:
          type: string
        name:
          type: string
        powerRequested:
          type: number
          description: the power in whats that the device needs to perform its primary function
        powerOffered:
          type: number
          description: the power in watts that the device has available to supply the microgrid
        powerPrice:
          type: number
          description: |
            The price at which the device is willing to buy or sell power. Can be used as a priority.
```

```
case class Device(id: String, name: String, powerRequested: Option[BigDecimal] = None,
powerOffered: Option[BigDecimal] = None, powerPrice: Option[BigDecimal] = None)
```

Polaris Entities

* Device
  * Includes name, ID, power requested, power offered, price
  * Power granted, power accepted, separate for permissions
  * Future: add more device details eg. device type etc.
* Subscriptions (observer URI, observed URI, action)
  * Add, remove, list
* GC
  * Includes power price, date/time, reset function
  * Future: add bus IDs for multiple buses per GC

Polaris Messaging

* HTTP with JSON encoding
  * URI path selects the entity
  * Action: http get/put/post/delete
  * Future: https, (identity × path × action) ACLs
* Adding actor messages for platform efficiency
  * Functions like capacity manager can elide serialization
* Futures
  * CoAP
  * Websockets (CoAP over TCP) - permits cloud GCs

Polaris Capacity Manager

* Observes /devices for post/delete (device add, remove)
* For each device (/device/{id} put):
  * Observe: PowerRequested, PowerOffered, Price
* Sort PowerOffered by price (lowest first)
* Calculate total power available
* Sort PowerAvailable by price (highest first)
* Select devices to receive power
* Calculate total power demand
* Select devices to source power
* Notify all devices of changes:
  * Put PowerGranted, PowerAccept

Polaris Device Energy Management

* Device chooses its own algorithm/behavior
* Default device algorithm:
  * All devices observe: /gc/PowerPrice
  * If gc price > device price set PowerRequested to zero
  * If gc price < device price set PowerOffered to zero
* Future:
  * Multiple buses with independent PowerPrice
  * Energy manager
    * Calculates price from sources?
    * LPD-like price adjustments to match supply/demand?
  * Price schedule entities like 2030.5

Polaris Next Steps

https://github.com/jlgula/Polaris

* Actor conversion of all modules
* Improved configuration, packaging, logging
* Service locator via DNS/Multicast (Akka-HTTP module)
* HTTPS, ACLs (via Akka-HTTP route functions)
* Alternate message formats: CoAP (Californium), WS (Akka)
* Additional control functions:
  * Energy manager
  * Battery manager
  * Explicit islanding states to match 2030.7,8
  * Droop control / stability?
* Zephyr device drivers
* Web UI (React front end for existing API)

Related Standards / Industry Developments

* P2030.10
  * 2030.10 is complete and published!
* LFEnergy ?
* OwnTech – Open Digital Power ?
* P2030.10.1 ?
* TMS ?
* GOGLA Interop activities ?
* OpenPAYGO Link ?
* Angaza Nexus Channel / Nexus Channel Core ?
* Open Connectivity Foundation / IoTivity -?

Next Meeting / Feedback

* Next Meeting
  * 8 February – 1500 UTC
  * Zoom – Meeting ID 87518284403 password: opendcgrid
* Sharing Portals
  * Web site: https://open-dc-grid.org/
  * GitHub: https://github.com/open-dc-grid