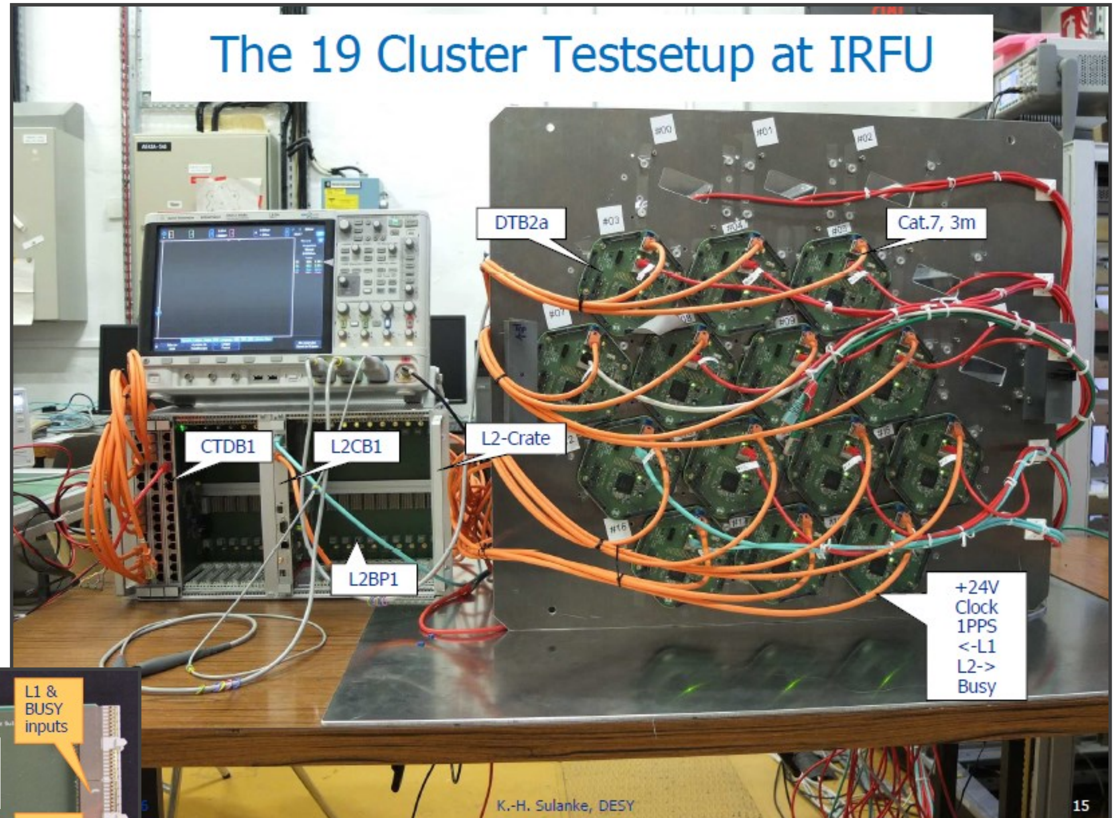
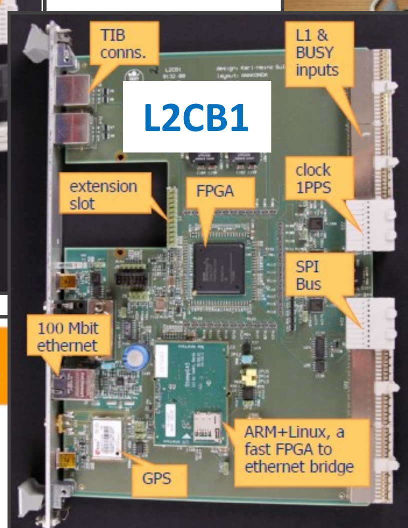
Prototyping the C++ OPC UA Server for the L2CTDB

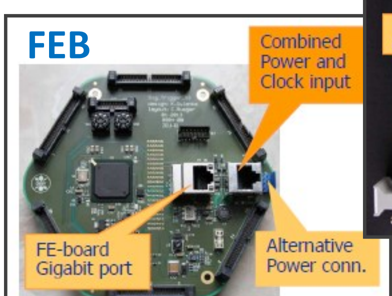
[current status]

Overview

FEB = Frontend Board
DTB = Digital Trigger Backplane
L2BP = L2 Backplane
CTDB = Clock & Trigger Distribution Board
L2CB = L2 Controller Board







Pictures from the presentation of K.Sulanke,"Digital Trigger, NectarCam F2F, 2016-04"

Overview

Picture from the User Manual, K.Sulanke, “Clock & Trigger Distribution Board Rev.1”

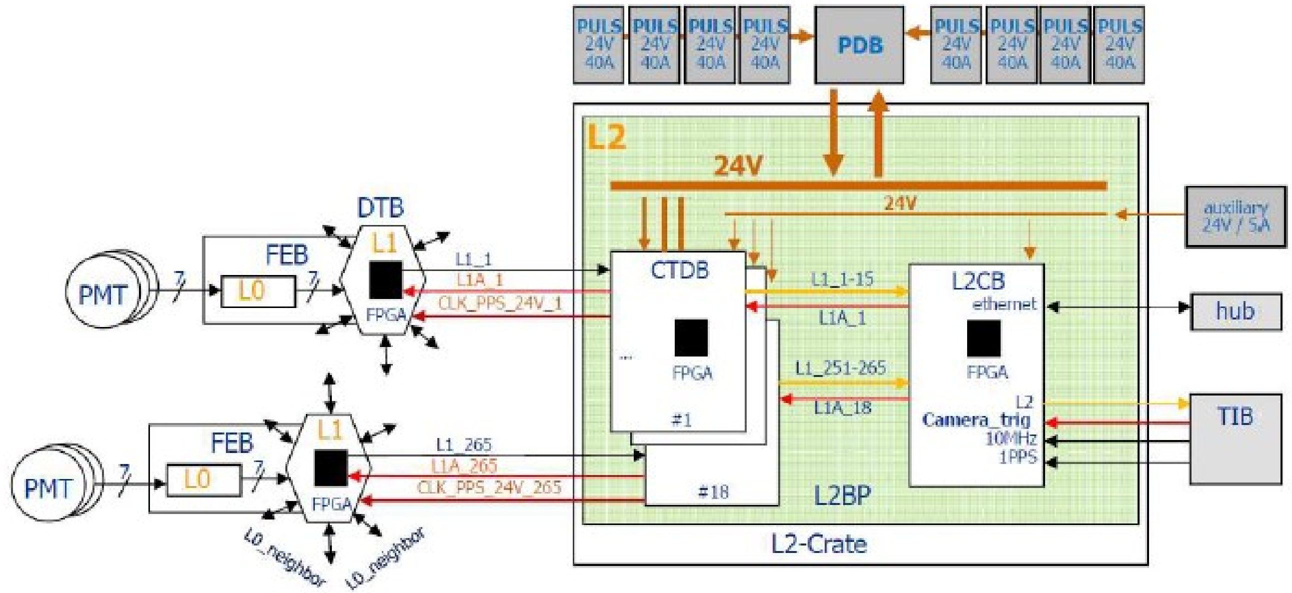


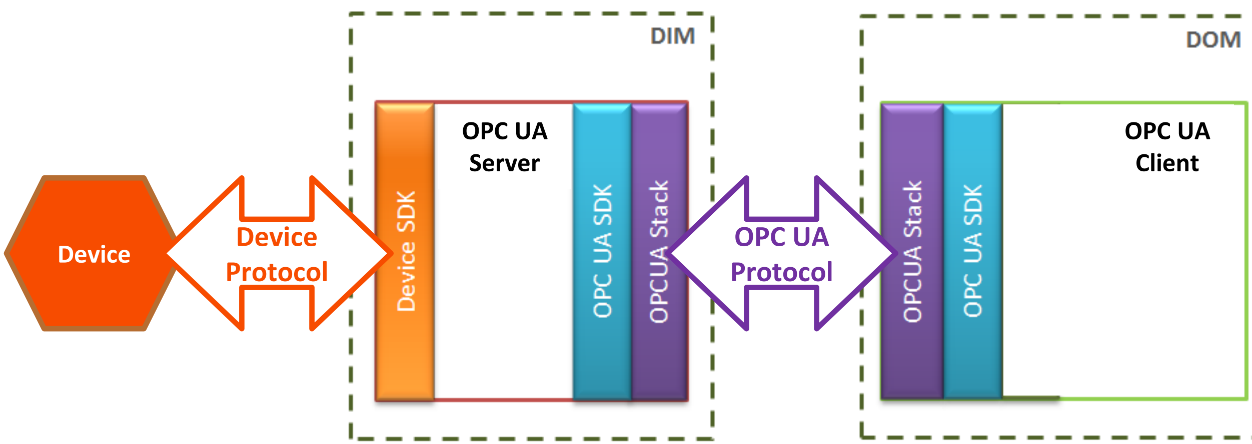
Figure 1: Trigger and Power Topology

- L2-crate includes: one Backplane (L2BP), 18 Clock & Trigger Distribution Boards (CTDBs) and one L2 Controller Board (L2CB)
- CTDB board is connected to 15 Frontend Boards (FEBs)
- L2CB comes with the Linux ARM-based on-board microcontroller unit with a Fast Ethernet to FPGA Bridge
    - the main features are: clock distribution, L1 trigger and “FEB_BUSY” receptions, L2+trigger Fan-out, FEB power control, current measurement, monitoring of FPGA configuration and trigger

The goals

- Implement the first (@DESY) C++ Server that represents the L2CB hardware via standard software layer based on the OPC UA protocol

- Provide a good examples to others on:
  - how to design the OPC UA Address Space
  - how to use the UnifiedAutomation OPC UA C++ SDK



Prototyping the C++ OPC UA Server for L2CB
The plan

1. C++ OPC UA SDK
   - (Cross-)Compile the SDK for the Linux ARM-based embedded system
   - Test the sample OPC UA servers on this system

2. Prototype the OPC UA Server Address Namespace (ANS)
   - Create Server Data Information Model (DIM)
   - Design (and review) the OPC UA Address Space for this DIM
   - Implement the OPC UA L2CB Server that provides this ANS

3. Design and implement (and review) the “L2CB Hardware Abstraction Layer” (HAL): a low-level communication layer for controlling the L2CB hardware (Marek Penno)

4. OPC UA L2CB Simulation Server
   - Design and implement the server that emulates behavior of the L2CB hardware - provides reasonable simulation data (as much as simple but still realistic)

5. Integrating the HAL into L2CB OPC UA Server
   - Cross-compile for the embedded system

6. Test OPC UA Server with the real hardware

Prototyping the C++ OPC UA Server for L2CB
Data Information Model

... to the Data Information Model

from several meetings and short notes ...

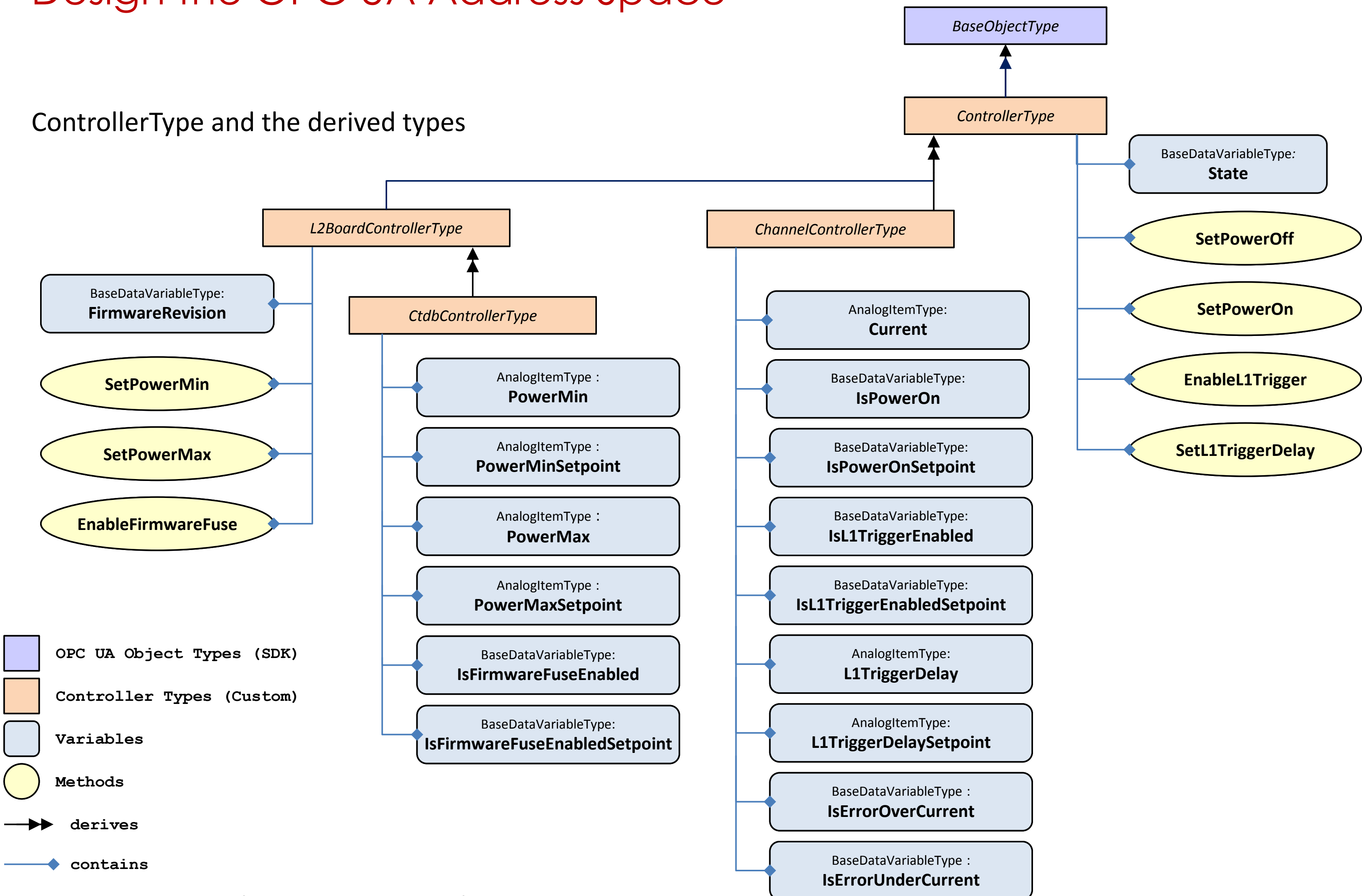
```
L2CB -> 1 :
 * [ro] has a firmware revision (uint16 ?  string ?)
 * [r/w] get/set L1 Trigger Channels mask 265 x 7 pixels
 * [r/w] enable/disable L1 Trigger Channels 265 x 7 pixels
 * [r/w] adjust L1 delays 265
 * [ro] read L1 pattern in case of camera trigger (?)

CTDB -> [18] slots 1..9, 13..21:
 * [ro] has a firmware revision (uint16 ?  string ?)
 * switch channels power on/off (15 channels, all?)
 * mask L2 trigger channels (?)
 * get FEB power currents (all? dedicated?)
 * get/set  global MAX/MIN power currents (12 bit uint)
 * enable/disable e-fuse
 * handle/monitor error conditions (over and under currents ?)
```

```
- L2CB
    |- String FirmwareRevision : ro
    |
    |- SetPowerOff()       <-- switches OFF the power of all power channels (18 x 15 channels)
    |- SetPowerOn()        <-- switches ON the power of all power channels (18 x 15 channels)
    |- SetPowerMin(double min)    <-- sets minimum power of all power channels (18 x 15 channels)
    |- SetPowerMax(doublemax)     <-- sets minimum power of all power channels (18 x 15 channels)
    |- EnableL1Trigger(bool enabled) <-- enables/disables L1 trigger of all trigger channels (18 x 15 channels)
    |- EnableFirmwareFuse(bool enabled) <-- enables/disables the firmware fuse of all CTDBs (18)
    |
    - CTDB 1..18
        |- String FirmwareRevision : ro
        |- String PowerMin : ro
        |- String PowerMinSetpoint : rw
        |- String PowerMax : ro
        |- String PowerMaxSetpoint : rw
        |- Boolean IsFirmwareFuseEnabled : ro
        |- Boolean IsFirmwareFuseEnabledSetpoint : rw
        |
        |- SetPowerOff()       <-- switches OFF the power of 15 power channels of a dedicated CTDB (one CTDB)
        |- SetPowerOn()        <-- switches ON the power of 15 power channels of a dedicated CTDB (one CTDB)
        |- SetPowerMin(double min)    <-- sets minimum power of 15 power channels of a dedicated CTDB (one CTDB)
        |- SetPowerMax(double max)    <-- sets minimum power of 15 power channels of a dedicated CTDB (one CTDB)
        |- EnableL1Trigger(bool enabled) <-- enables/disables L1 trigger of 15 trigger channels of a dedicated CTDB (one CTDB)
        |- EnableFirmwareFuse(bool enabled) <-- enables/disables the firmware fuse of one dedicated CTDB
        |
        |- Channel 1..15
        |       |- Double CurrentPower : ro
        |       |- Boolean IsPowerOn : ro
        |       |- Boolean IsPowerOnSetpoint : rw
        |       |- Boolean IsL1TriggerEnabled : ro
        |       |- Boolean IsL1TriggerEnabledSetpoint : rw
        |       |- Uint16 TriggerDelay : ro
        |       |- Uint16 TriggerDelaySetpoint : rw
        |       |- Boolean isErrorOverCurrent : ro
        |       |- Boolean isErrorUnderCurrent : ro
        |       |
        |       |- SetPowerOff()  <-- switches OFF the power of dedicated power channel
        |       |- SetPowerOn()   <-- switches ON the power of dedicated power channel
        |       |- EnableL1Trigger(bool enabled) <-- enables/disables L1 trigger of dedicated trigger channel
        |       |- SetL1TriggerDelay(unint16 delay) <-- enables/disables L1 trigger the dedicated trigger channel
        | ...
```

Prototyping the C++ OPC UA Server for L2CB

Design the OPC UA Address Space

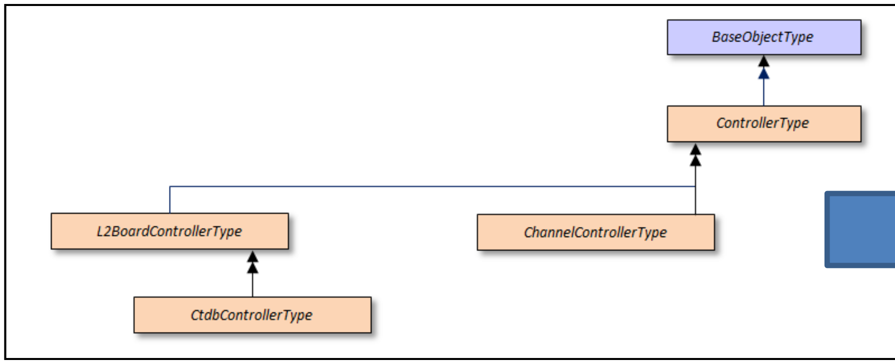
ControllerType and the derived types



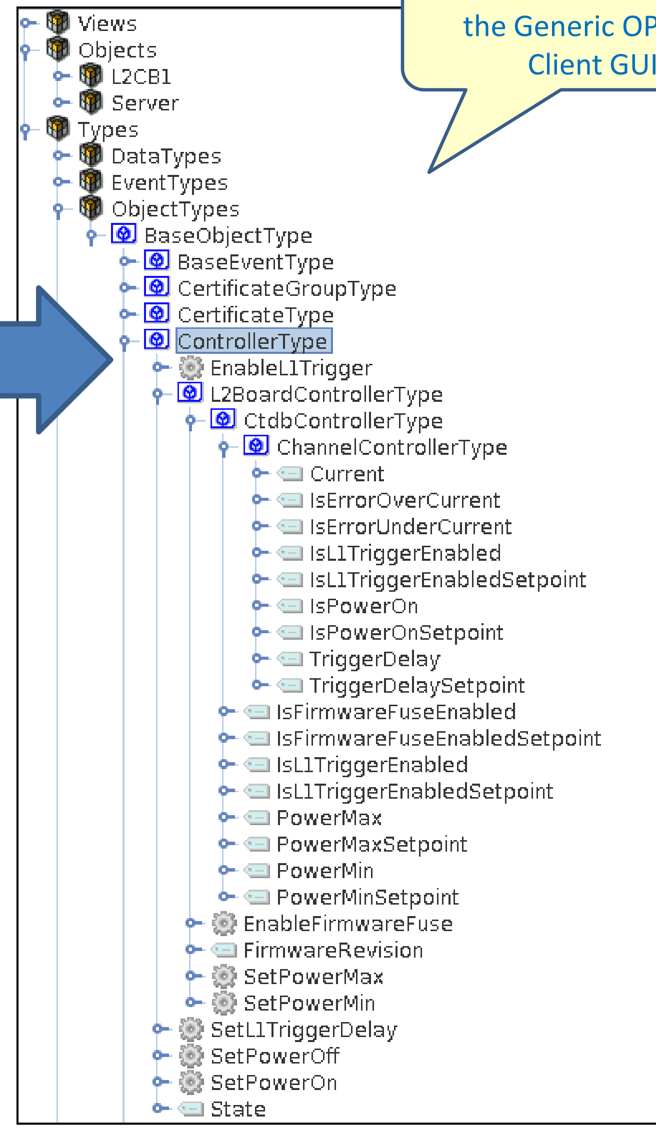
Prototyping the C++ OPC UA Server for L2CB

The OPC UA Server Address Space

From Design to Implementation …



Screen-shot from the Generic OPC UA Client GUI



Type

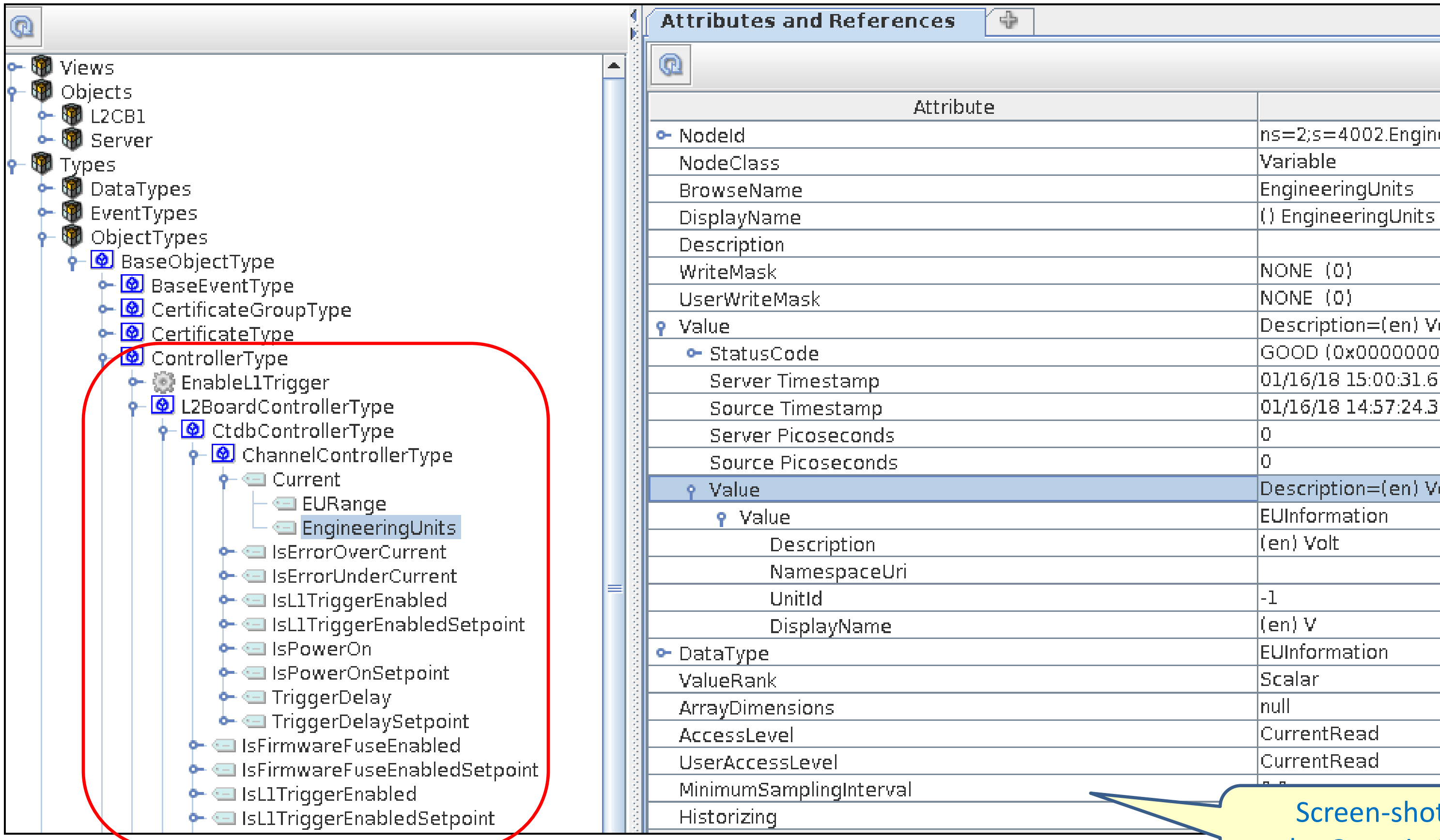
Variable

Method

Prototyping the C++ OPC UA Server for L2CB

The OPC UA Server Address Space

Example: the Attributes and References view of the OPC UA "Current" variable (EngineeringUnits Node)

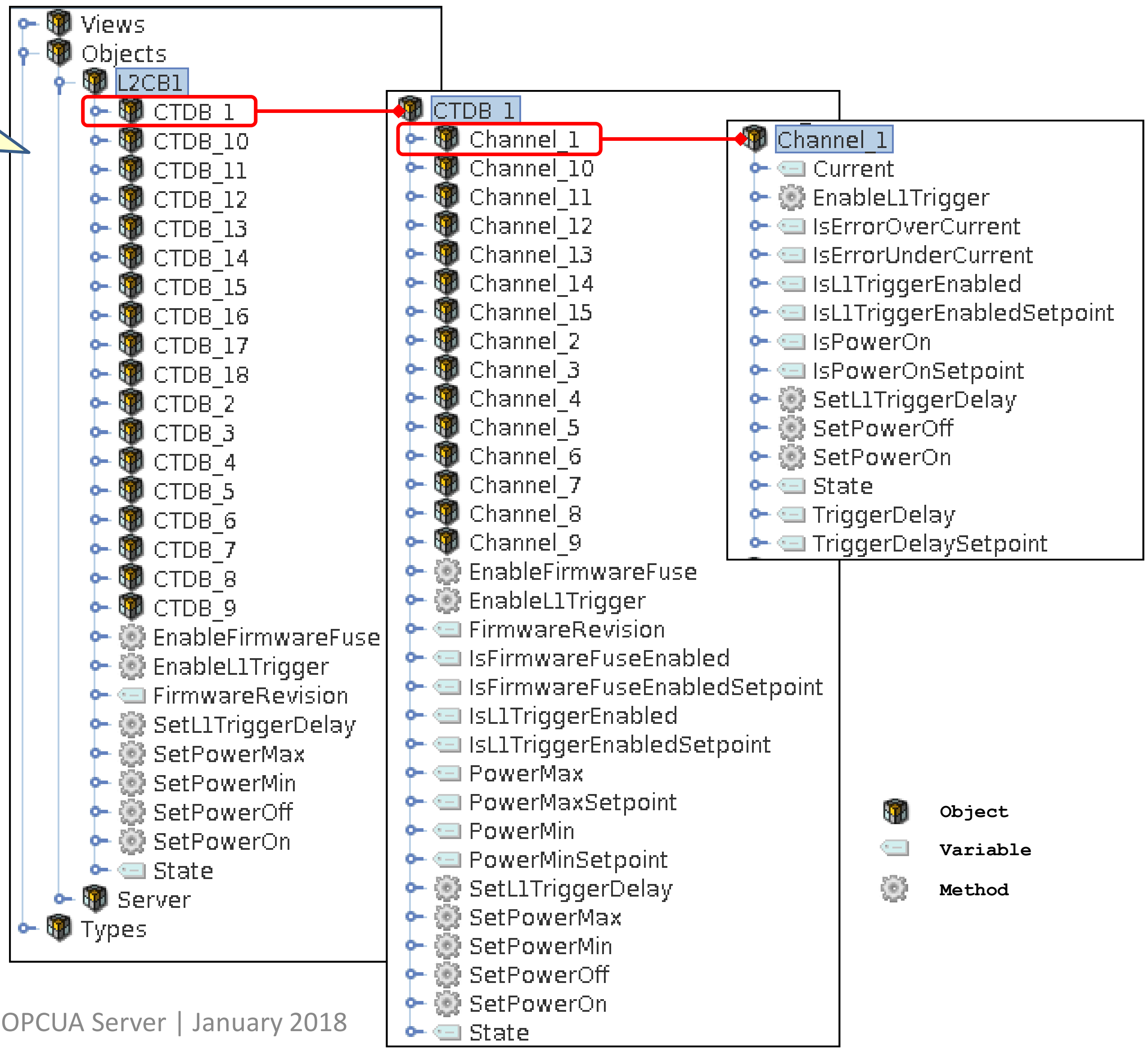


Screen-shot from the Generic OPC UA Client GUI

Prototyping the C++ OPC UA Server for L2CB

Object tree in the OPC UA Server Address Space

Screen-shots from the Generic OPC UA Client GUI



Prototyping the C++ OPC UA Server for L2CB

Summary

1. C++ OPC UA SDK
   - (Cross-)Compile the SDK for the Linux ARM-based embedded system.
   - Test the sample OPC UA servers on this system
   - Cross-compile the newest version 1.5.6

Difficulties (SOLVED):
- Problems with compiling the third-party libraries (e.g. zlib, xz)
- Security layer was not working properly with the pre-installed OpenSSL libs

2. Prototyping OPC UA Address Namespace (AN)
   - Create Server Data Information Model (SDIM)
   - Design (and Review) the AN for the SDIM
   - Implement the initial version of the OPC UA L2CB Server that provides this AN

3. Design/Implement (and Review) the "L2CB Hardware Abstraction Layer" (HAL): a low-level communication layer for controlling the L2CB hardware (Marek Penno)

DONE

4. OPC UA L2CB Simulation Server
   - Design and implement the server that emulates behavior of the L2CB hardware - provides reasonable simulation data (as much as simple but still realistic)

5. Integrating the HAL into L2CB OPC UA Server
   - Cross-compile for the embedded system

6. Test OPC UA Server with the real hardware

TODO