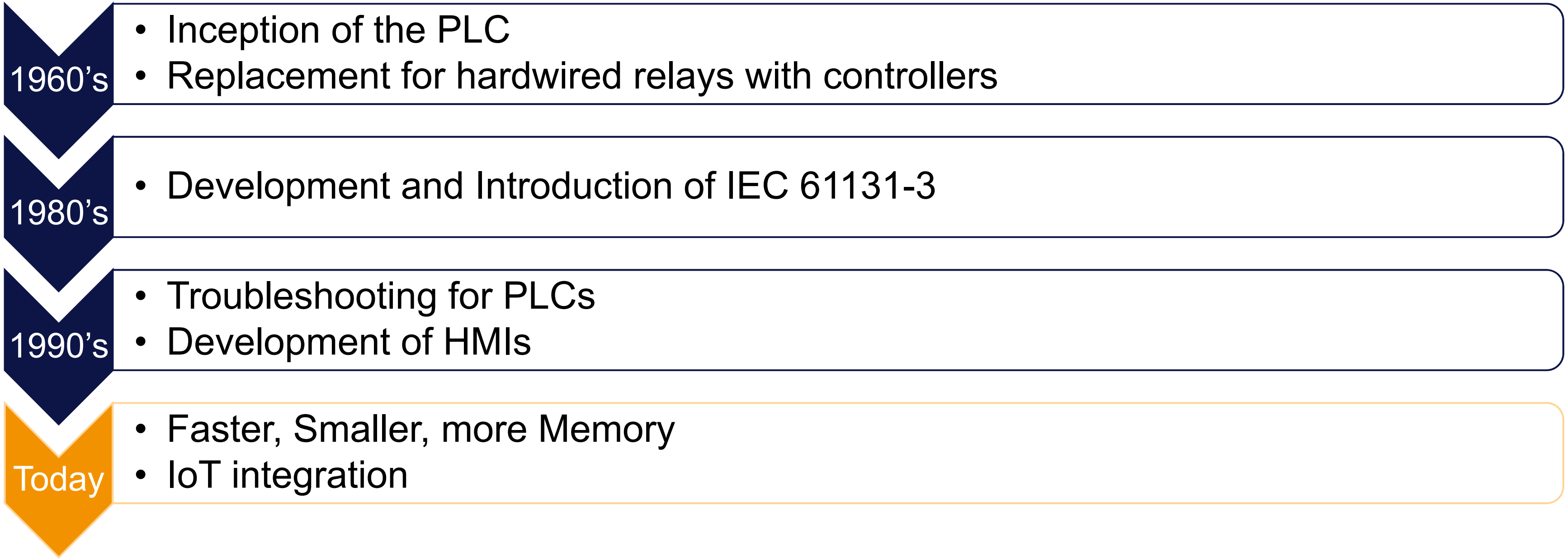
Applying Standardised Software Architectural Concepts to Design Robust and Adaptable PLC Solutions

Sylvia Huynh
PLC Developer and Support Engineer

ICALEPCS 2023
Cape Town, 09.10.2023

A Short History of PLCs



1960's
- Inception of the PLC
- Replacement for hardwired relays with controllers

1980's
- Development and Introduction of IEC 61131-3

1990's
- Troubleshooting for PLCs
- Development of HMIs

Today
- Faster, Smaller, more Memory
- IoT integration

Background

- PLC development has predominately been performed by those in mechanical or electrical engineering
- Technical debt has resulted in the need for redevelopment
- Incorporating a more software engineering driven approach can bring about many benefits to the PLC

Image credit to : https://www.xfel.eu

Architectural Design Records

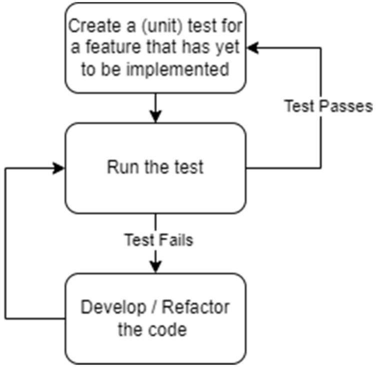
: Concept

- Understanding the justification of what was done.

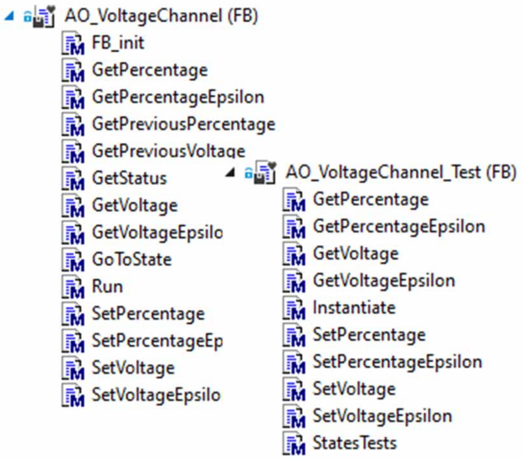
- ADR provide a template in a simple, straight forward manner, and doubles as documentation.

- An ADR comprises of:
  - Topic – What is the decision for – eg. Array Boundary declaration
  - Description - A short description of the design topic
  - Decision – What was decided
  - Status – Proposed, Accepted, Rejected, Superseded, Deprecated
  - Assumptions – Underlying assumptions such as cost, technology etc
  - Constraints – Additional constraints that are imposed upon the decision
  - Positions – All the positions considered, and their pros and cons
  - **Justification** – The reason this particular decision was made.
  - Implications – Foreseeable implications of this decision
  - Related resources or decisions – Any related resources for further reading, or related ADRs

Test Driven Development : Concept


Create a (unit) test for a feature that has yet to be implemented
Run the test
Test Passes
Test Fails
Develop / Refactor the code

- Testing PLC code can be challenging, especially when the equipment may not be accessible
- Test Driven Development ensures a high code coverage leading to a more robust and stable framework
- TcUnit:
  - Framework for easy integration
  - Incorporated into Git CI/CD

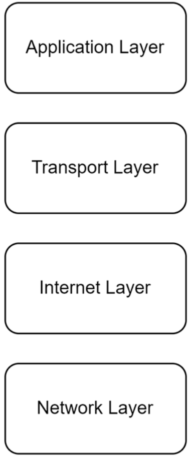


- AO_VoltageChannel (FB)
  - FB_init
  - GetPercentage
  - GetPercentageEpsilon
  - GetPreviousPercentage
  - GetPreviousVoltage
  - GetStatus
  - GetVoltage
  - GetVoltageEpsilo
  - GoToState
  - Run
  - SetPercentage
  - SetPercentageEp
  - SetVoltage
  - SetVoltageEpsilo

- AO_VoltageChannel_Test (FB)
  - GetPercentage
  - GetPercentageEpsilon
  - GetVoltage
  - GetVoltageEpsilon
  - Instantiate
  - SetPercentage
  - SetPercentageEpsilon
  - SetVoltage
  - SetVoltageEpsilon
  - StatesTests

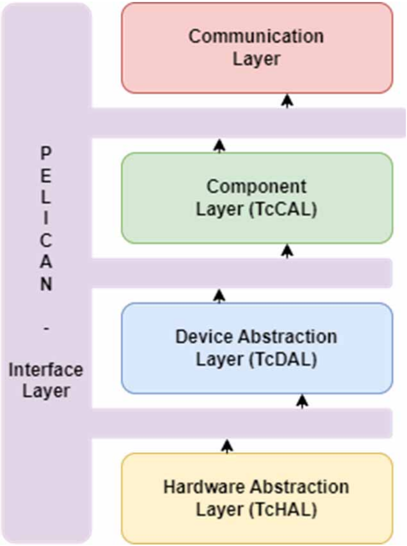
Layered Architecture Pattern

: Concept



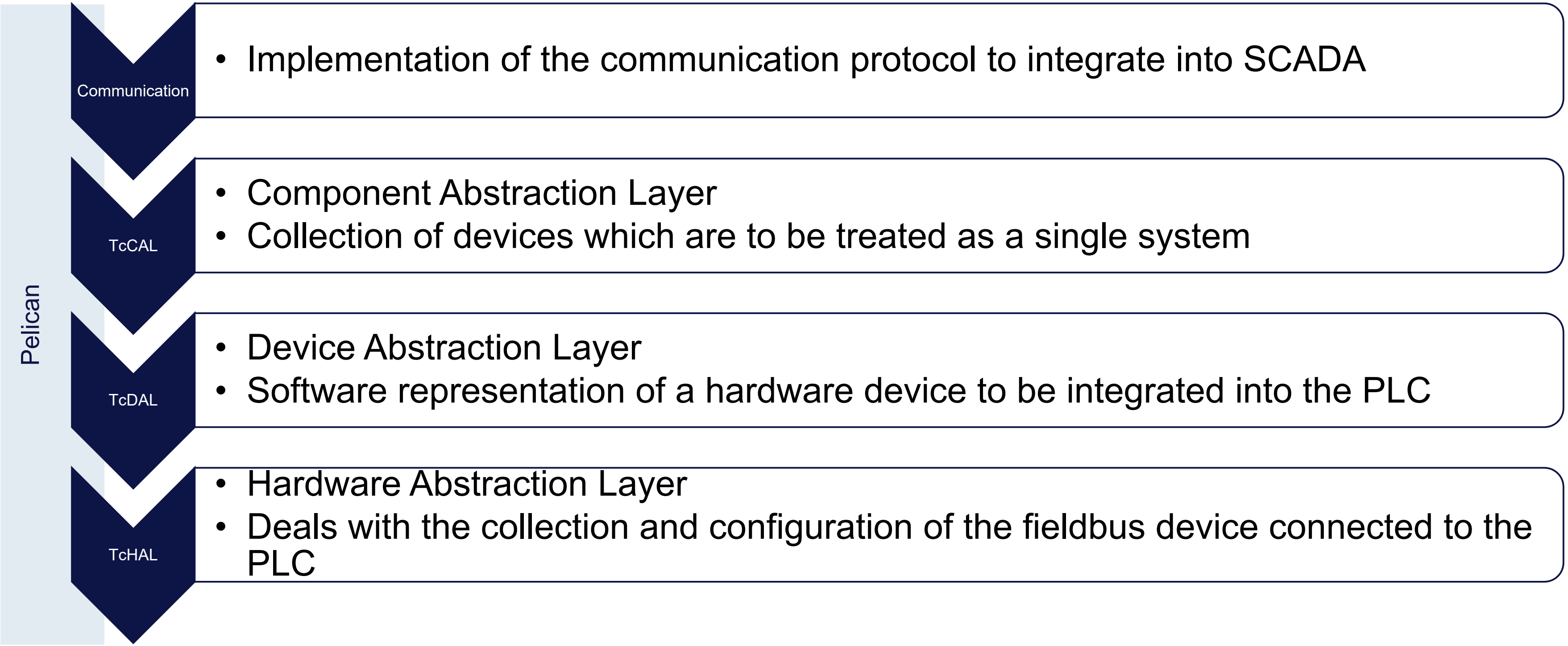
TCP Protocol

- A Layered architecture provides a separation of concerns
  - Easier to develop when there is a limited scope
  - Knowledge required can be restricted to the immediate layer, opening up development opportunities to others
- Each layer is responsible for a specific function
  - Changes between layers do not impact the other
  - Prevents code entanglement
  - Limits the dependencies when dealing with refactoring


PELICAN - Interface Layer
Communication Layer
Component Layer (TcCAL)
Device Abstraction Layer (TcDAL)
Hardware Abstraction Layer (TcHAL)

The TcZookeeper at a Glance

: Implementation



Pelican

Communication
- Implementation of the communication protocol to integrate into SCADA

TcCAL
- Component Abstraction Layer
- Collection of devices which are to be treated as a single system

TcDAL
- Device Abstraction Layer
- Software representation of a hardware device to be integrated into the PLC

TcHAL
- Hardware Abstraction Layer
- Deals with the collection and configuration of the fieldbus device connected to the PLC

Interface Concept

: Concept

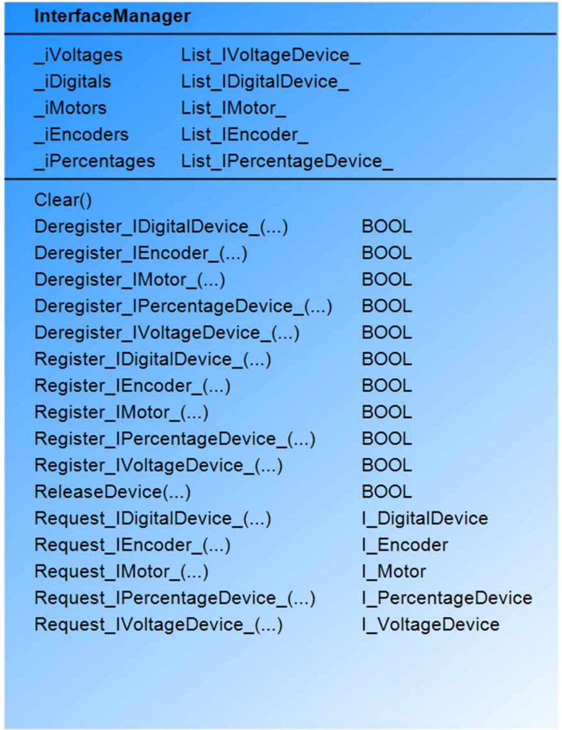
- An interface defines how one data structure can interact with another.
  - Available attributes and properties
  - Available functions

- Provides the developer with an outline of what information is available, making it easier to know what they can do in regards to implementation, without having to concern themselves with any of the technical details.

Observer Design Pattern

: Implementation

- An object can observe another, by requesting to register itself to the object which is to be observed
- The observed object then sends updates to all those who have registered, informing them of any values updates etc.
- Clear path of data being passed between objects across different architectural layers.
- This is implemented via the Interface Manager in the Pelican.

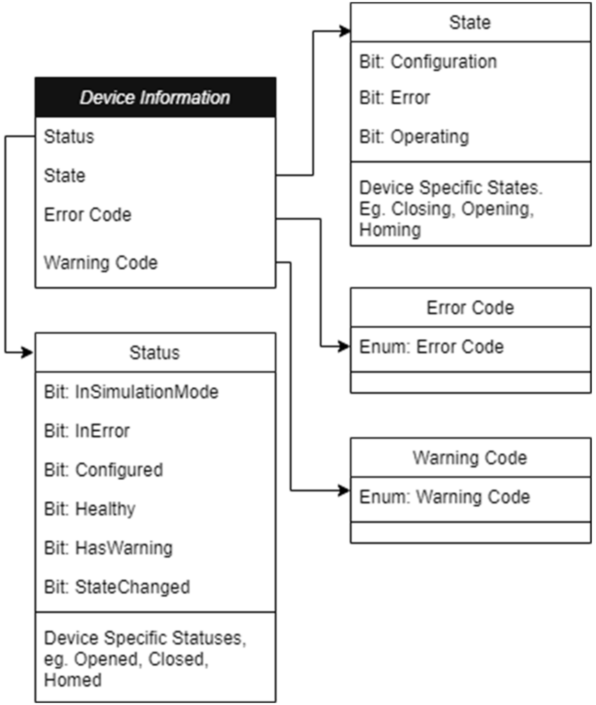


InterfaceManager

_iVoltages	List_IVoltageDevice_
_iDigitals	List_IDigitalDevice_
_iMotors	List_IMotor_
_iEncoders	List_IEncoder_
_iPercentages	List_IPercentageDevice_

Clear()	
Deregister_IDigitalDevice_(...)	BOOL
Deregister_IEncoder_(...)	BOOL
Deregister_IMotor_(...)	BOOL
Deregister_IPercentageDevice_(...)	BOOL
Deregister_IVoltageDevice_(...)	BOOL
Register_IDigitalDevice_(...)	BOOL
Register_IEncoder_(...)	BOOL
Register_IMotor_(...)	BOOL
Register_IPercentageDevice_(...)	BOOL
Register_IVoltageDevice_(...)	BOOL
ReleaseDevice(...)	BOOL
Request_IDigitalDevice_(...)	I_DigitalDevice
Request_IEncoder_(...)	I_Encoder
Request_IMotor_(...)	I_Motor
Request_IPercentageDevice_(...)	I_PercentageDevice
Request_IVoltageDevice_(...)	I_VoltageDevice

Finite State Machines in the quest of Modularity : Implementation



Device Information

Status

State

Error Code

Warning Code

State

Bit: Configuration

Bit: Error

Bit: Operating

Device Specific States. Eg. Closing, Opening, Homing

Error Code

Enum: Error Code

Warning Code

Enum: Warning Code

Status

Bit: InSimulationMode

Bit: InError

Bit: Configured

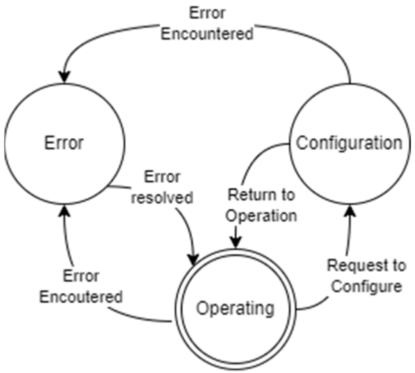
Bit: Healthy

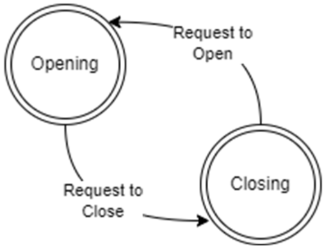
Bit: HasWarning

Bit: StateChanged

Device Specific Statuses, eg. Opened, Closed, Homed

- Every object is implemented as a device
  - Made up of a generic set of Device Information
  - Combined with any device specific features. E.g. Filtering on a 24V signal
- Core set of Operating States and Process States.
  - The transition between the Operation States for all devices is the same
  - Attention and focus is placed on the encapsulated device specific states.
- Simplifies interacting with objects cross the entire PLC libraries.





Support of Legacy Code – The Adapter Pattern

: Implementation

- Until it becomes possible to fully migrate an entire code base, legacy code will remain.
- In order to continue working on a new framework while maintaining the old, an adapter is required, bridging the two together.
- An adapter pattern describes how to connect two systems together in a way that the code designed is re-usable.
- This was achieved in order to adapt the I/O of the old framework into the TcHAL, which not only means we can run two frameworks simultaneously, but we can also utilise the feature set of the TcHAL within the legacy framework.

Future and Ongoing Developments

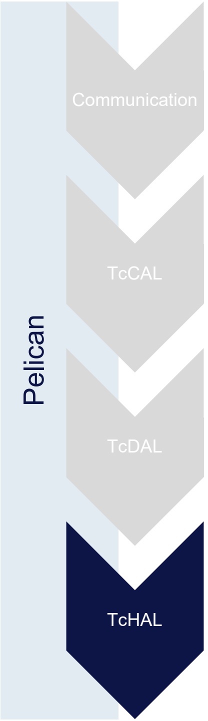
- Fully migrate the existing PLC code base to the TcZookeeper
- Plans to Open Source the TcZookeeper

Image credit to : https://www.xfel.eu

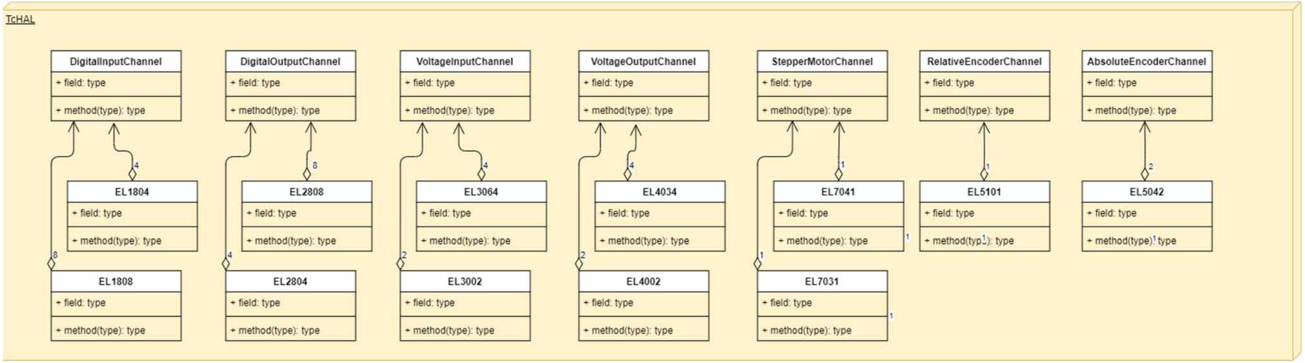
Sylvia Huynh
PLC Developer and Support Engineer
European XFEL
sylvia.huynh@xfel.eu

Questions?

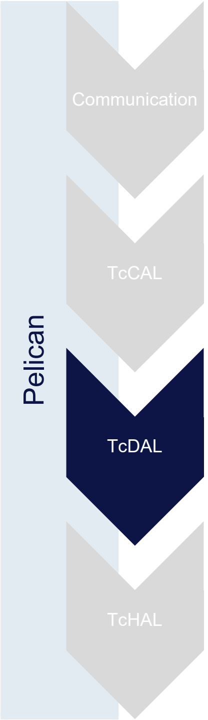
Hardware Abstraction Layer (TcHAL)



- Representation of the fieldbus device – EtherCAT terminals, EtherCAT devices etc
- Extracts all the information associated to that signal
- Encapsulates configuration and handling of the signal value into a SI unit

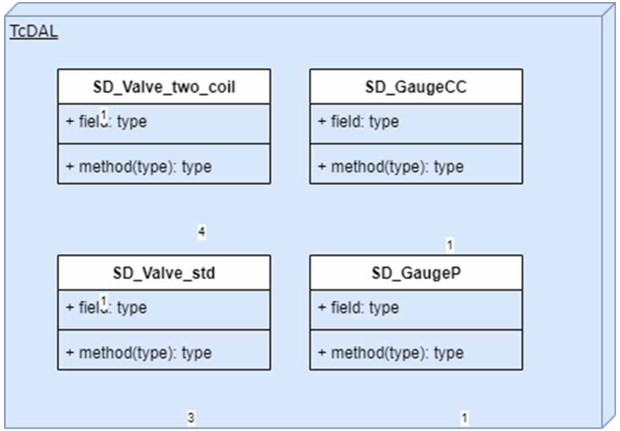


Device Abstraction Layer (TcDAL)

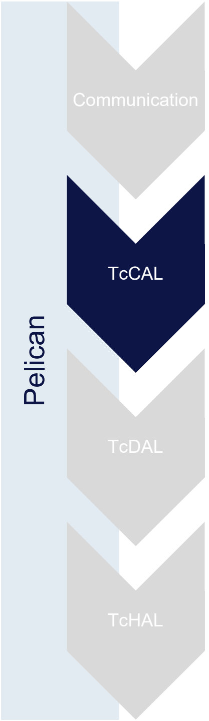


This layer provides the software representation of a device. A key benefit to a layered approach here is that on this layer, the developer can solely focus on:

- How the device behaves
  - What functions are available on the device
  - What properties are available on the device
  - All the execution in how the behavior is defined is encapsulated within the object or POU.
- The collection of signals that expected, without concerning themselves with where the signal is coming from

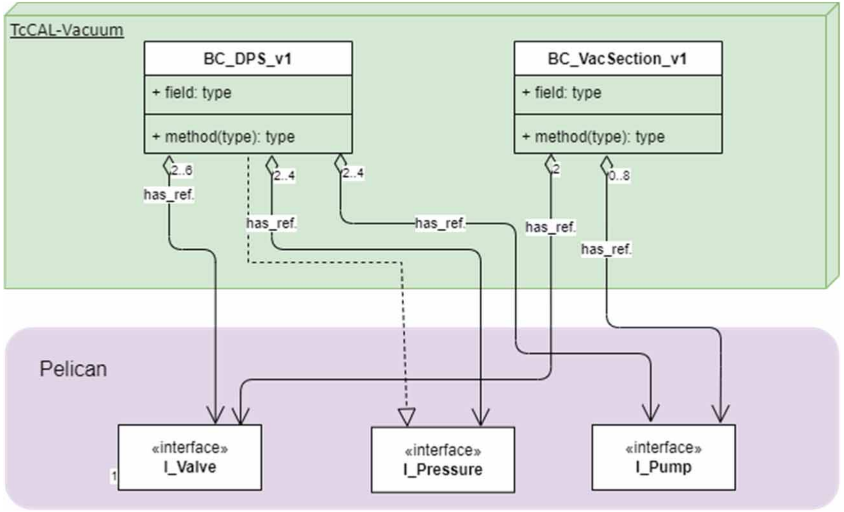


Component Abstraction Layer (TcCAL)

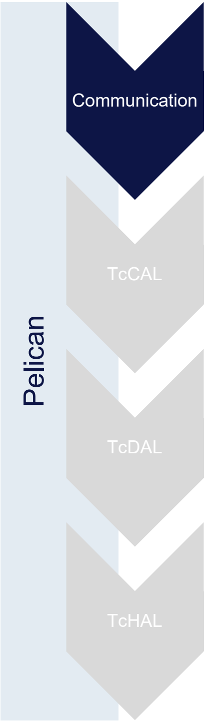


The Component layer brings together a group of devices from the TcDAL layer, and treats them as a single entity. Similar to how a single TcDAL device incorporates several signals from the TcHAL.

- Defines the primary controlling device
- All secondary devices relinquish control to the primary
- Components can then be used as building blocks in the PLC projects



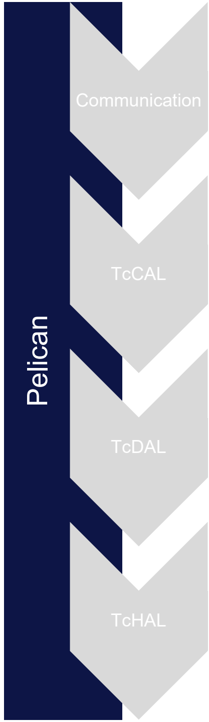
Communication Layer



To provide a means for the SCADA system to interact with the PLC, a communication protocol must be established. The handling of this protocol is defined within the communication layer.

- A single library is developed to handle one communication protocol
- The PLC can then interface with any communication protocol, or multiple protocols simultaneously.
- A library can be easily swapped out or added to provide the set of functions as needed.

Pelican



The Pelican holds all of the interface definitions.

- The interface defines what functions and attribute are available, and how they can be interacted with.
- All layers communicate between each other via the Pelican.
- The Interface Manager also lies within the Pelican
  - Helps maintain control over which devices in one layer are being interfaced to in another.