

# PLC Framework at the EuXFEL

Sylvia Huynh  
PLC Developer and Support Engineer

ICALEPCS 2023  
Cape Town, 06.10.2023



# European XFEL at a glance

- Two main groups, each team managing ~100 PLCS
- Beckhoff 2.0 & 3.0
- Pilz and Siemens for safety systems
- Language: ST

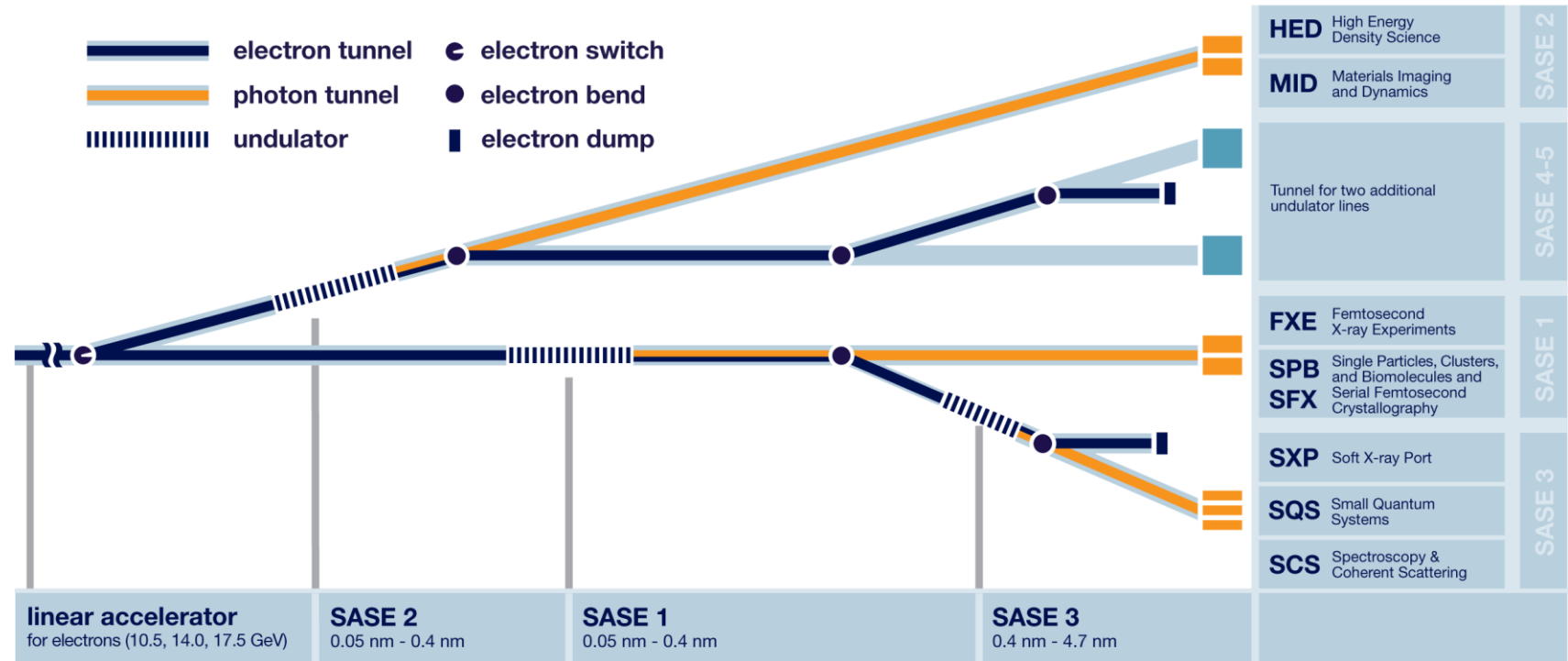


Image credit to : <https://www.xfel.eu>

## Current PLC Framework and Processes

- Core team of developers working on a custom PLC library and associated tools
  - PLCMS – EPLAN, CSV for Interlocks, Postgres BD, Python
  - PLC Project Builder – Beckhoff XAE
  - DocStrings and RTD for documentation
  - Client tool for parameter saving and writing – Python
  - Git CI/CD for versioning and deployment.
  
- Operations team generate the PLC projects
  - Partial automatic generation using the developed PLC teams

## Current Works

- 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

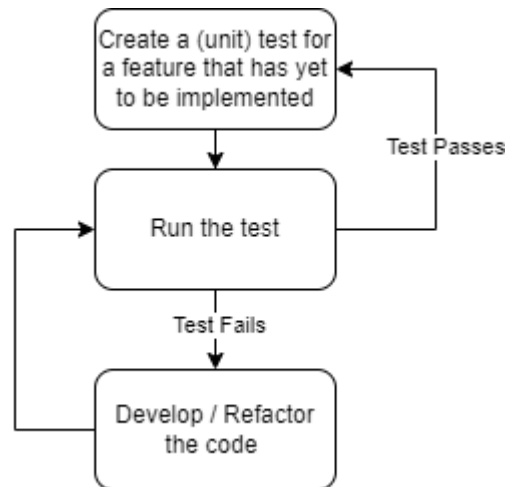


Image credit to : <https://www.xfel.eu>

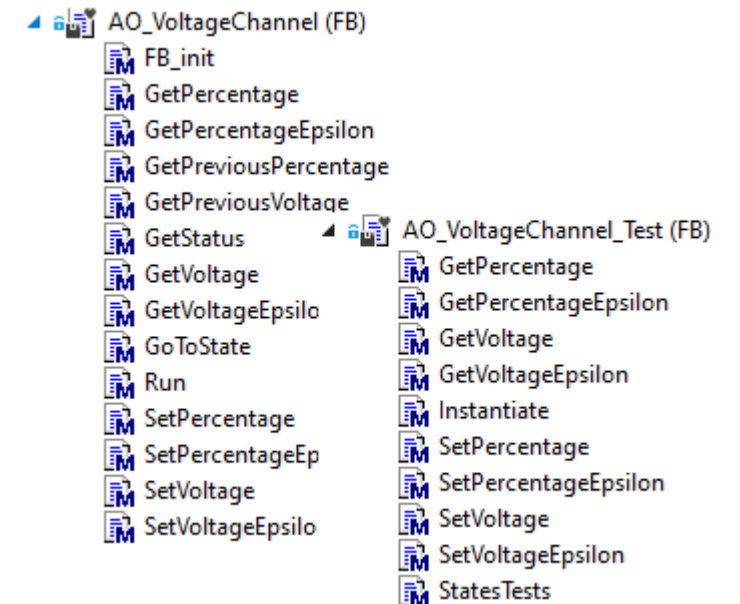
## Architectural Design Records

- 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  - Status – Proposed, Accepted, Rejected, Superseded, Deprecated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



- 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



## Layered Architecture Pattern

Application Layer

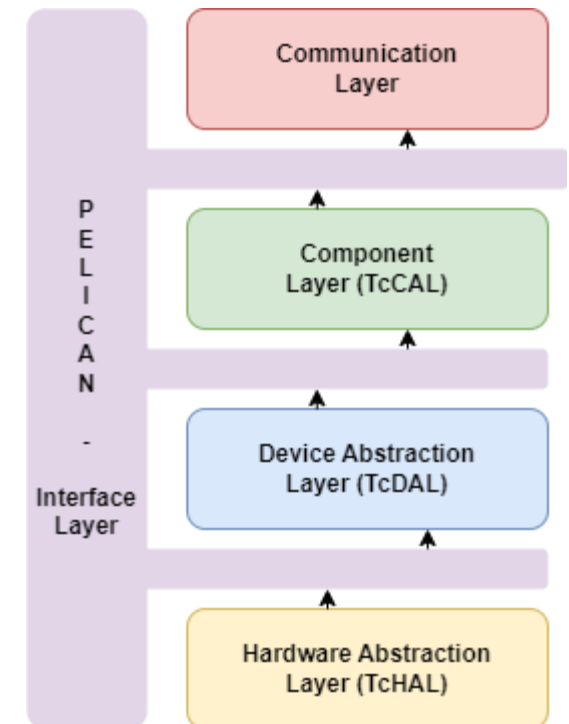
Transport Layer

Internet Layer

Network Layer

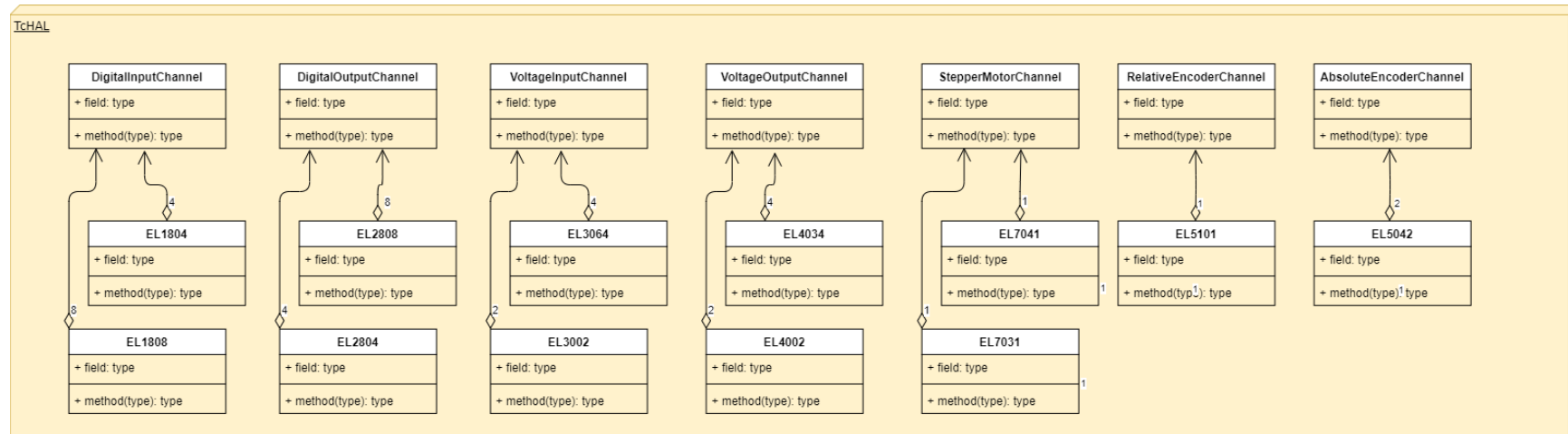
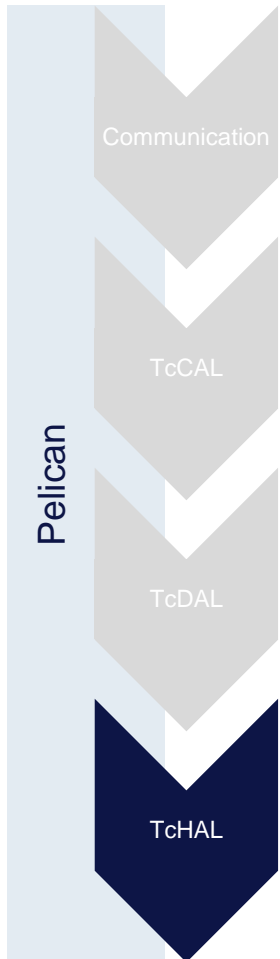
TCP Protocol

- 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



# Hardware Abstraction Layer (TcHAL)

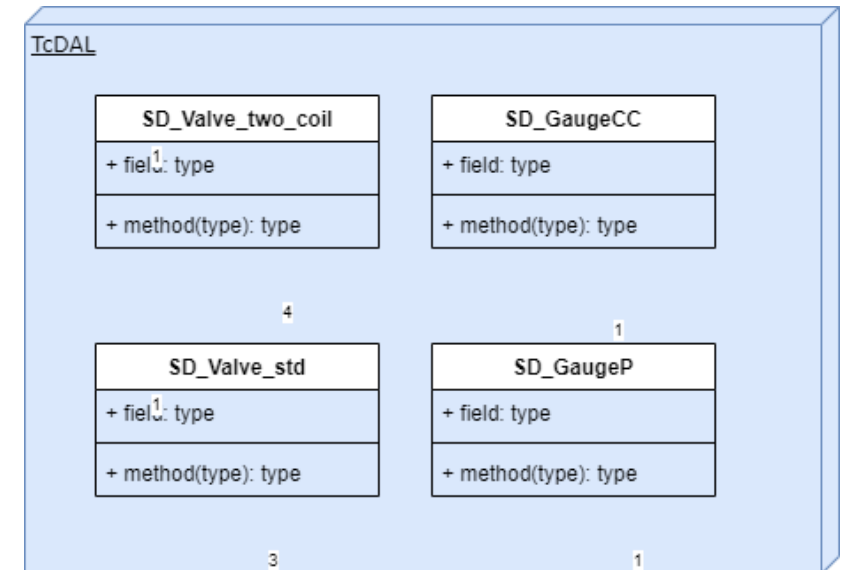
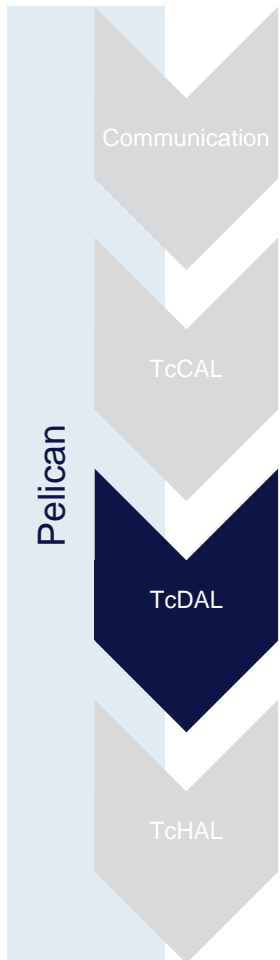
- Representation of the fieldbus device – EtherCAT terminals, EtherCAT devices etc
- Extracts all the information associated to that signal
- Encapsulates configuration and handling of the signal value into a SI unit



# 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:

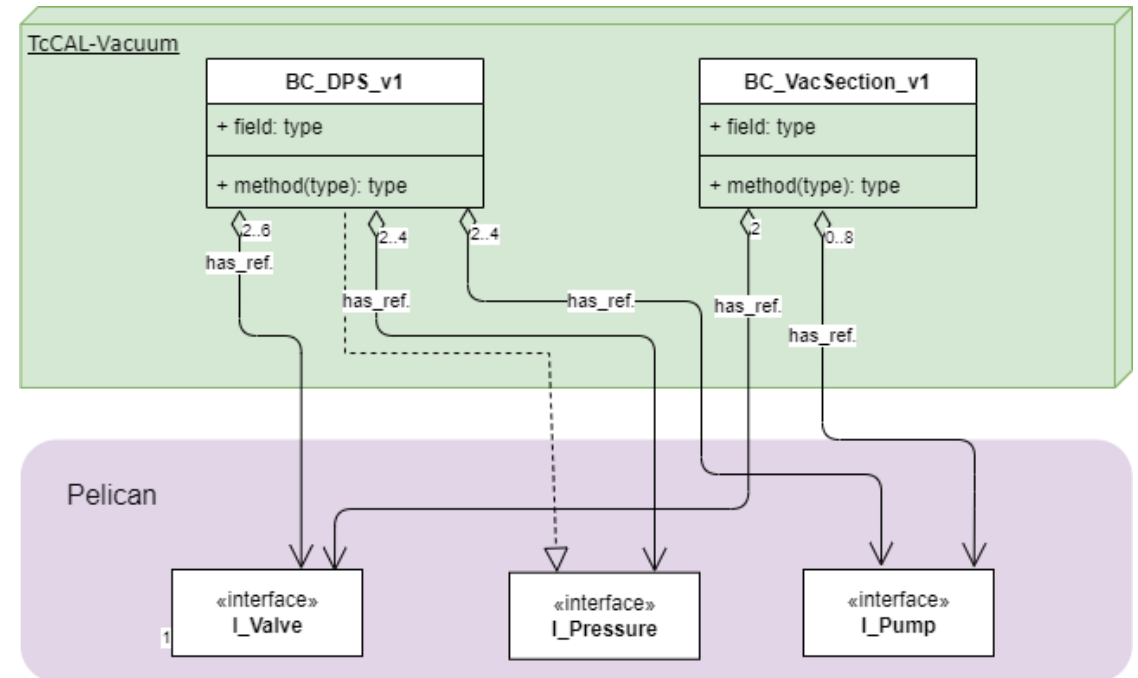
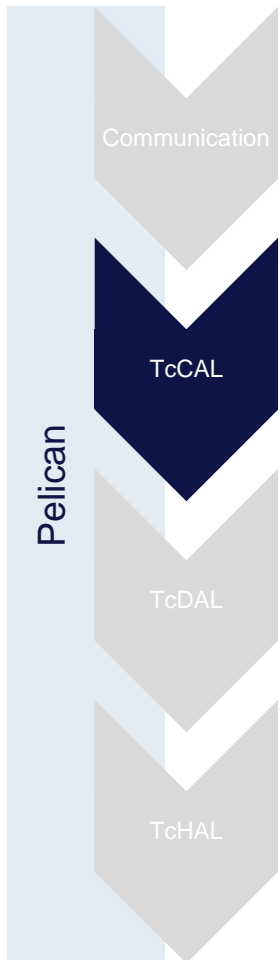
- How the device behaves
  - What functions are available on the device
  - What properties are available on the device
  - All the execution in how the behavior is defined is encapsulated within the object or POU.
- The collection of signals that expected, without concerning themselves with where the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

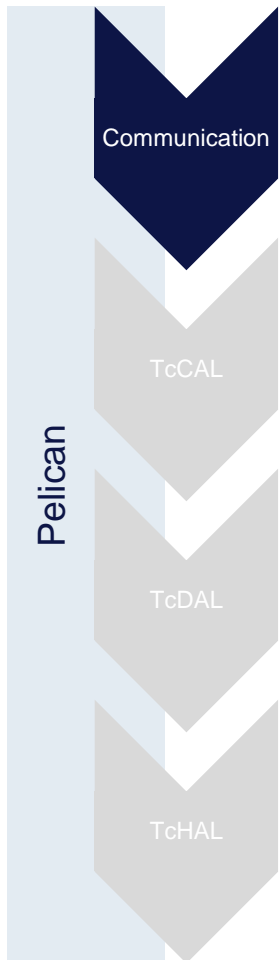
- Defines the primary controlling device
- All secondary devices relinquish control to the primary
- Components can then be used as building blocks in the PLC projects



## 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.



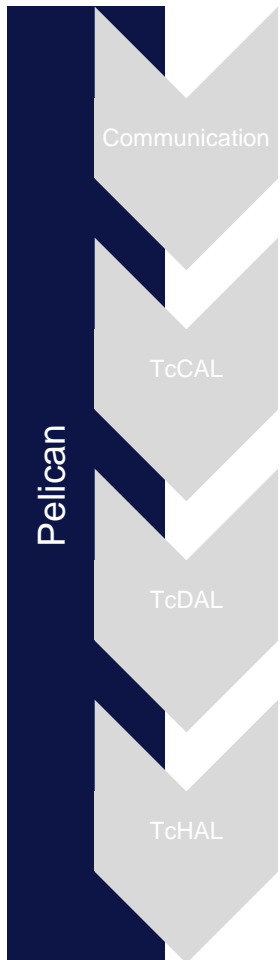
## Interface Concept

- 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.

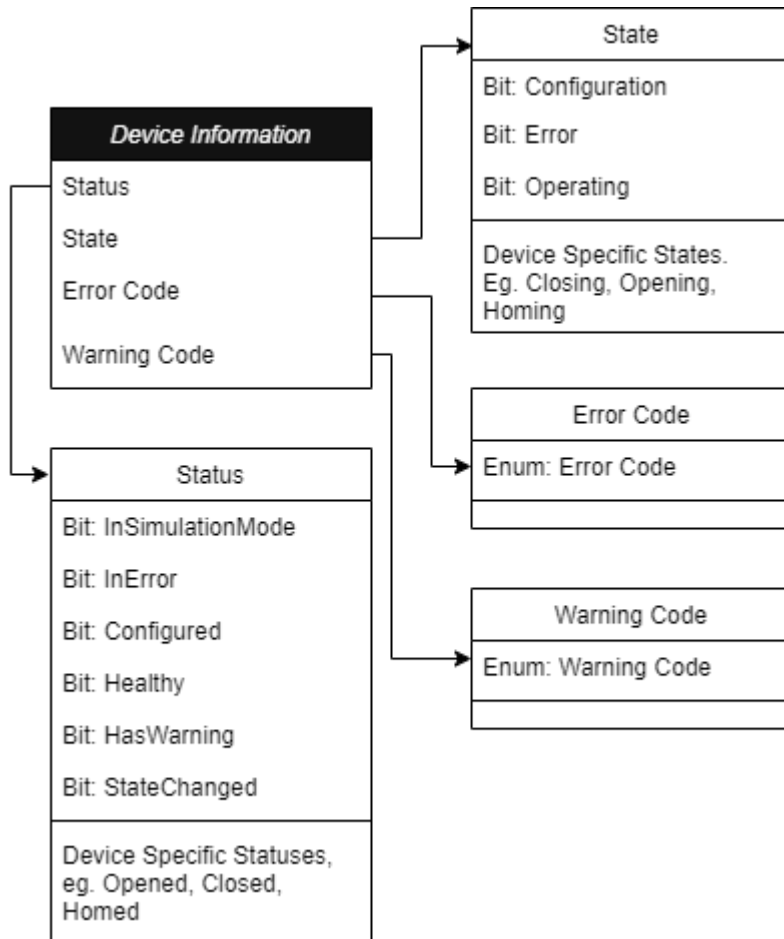
# Pelican

The Pelican holds all of the interface definitions.

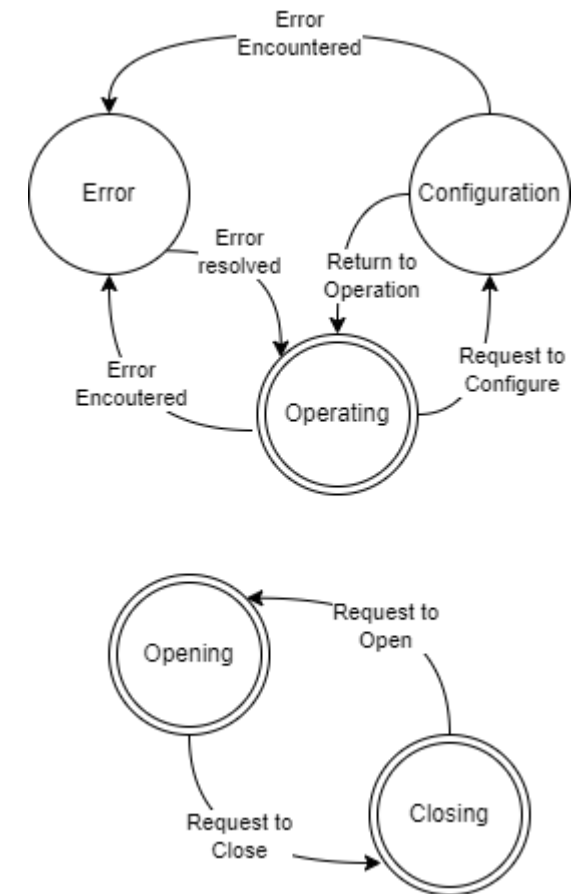
- The interface defines what functions and attribute are available, and how they can be interacted with.
- All layers communicate between each other via the Pelican.
- The Interface Manager also lies within the Pelican
  - Helps maintain control over which devices in one layer are being interfaced to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



# Finite State Machines in the quest of Modularity

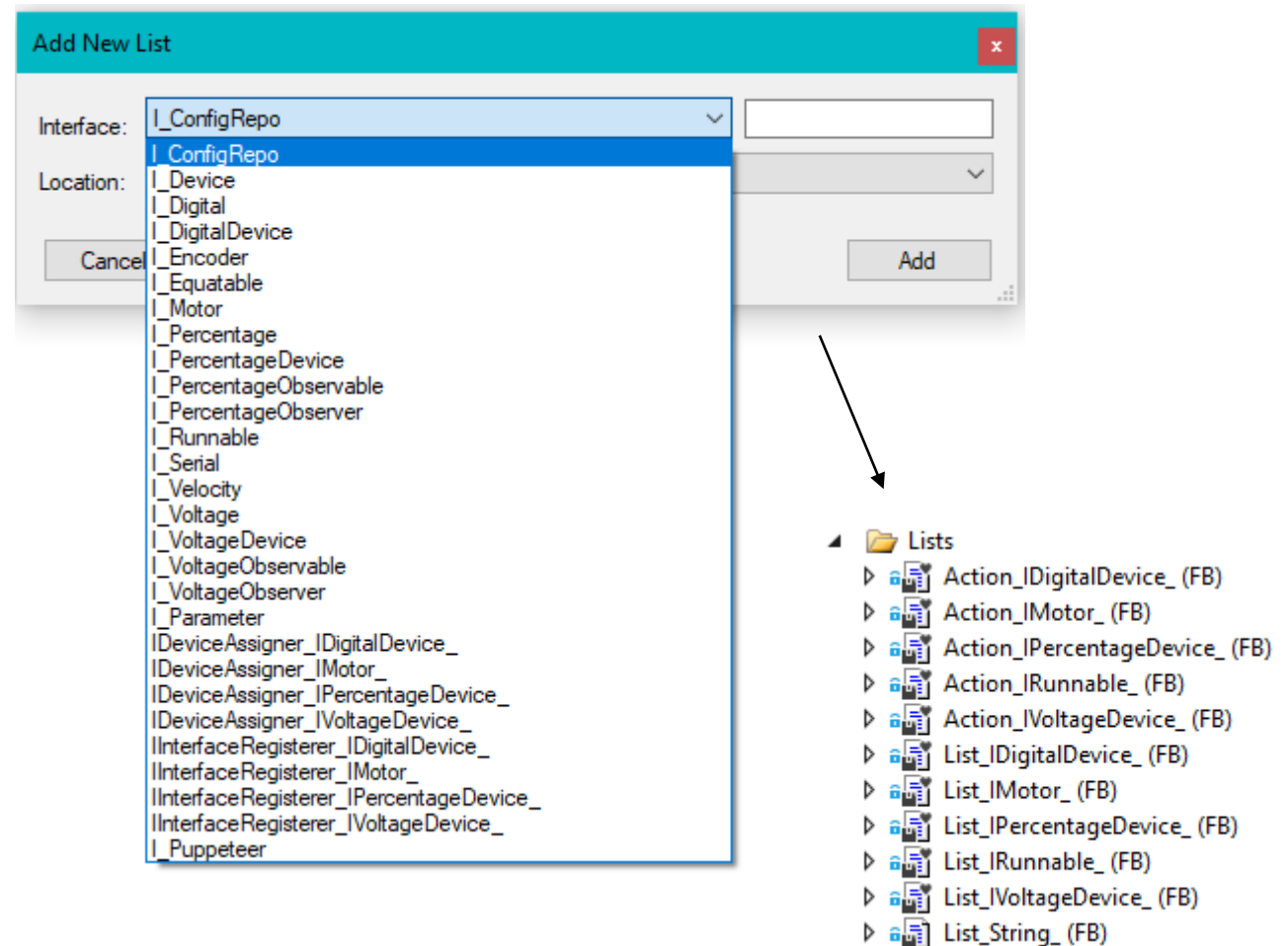


- Every object is implemented as a device
  - Made up of a generic set of Device Information
  - Combined with any device specific features. E.g. Filtering on a 24V signal
- Core set of Operating States and Process States.
  - The transition between the Operation States for all devices is the same
 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



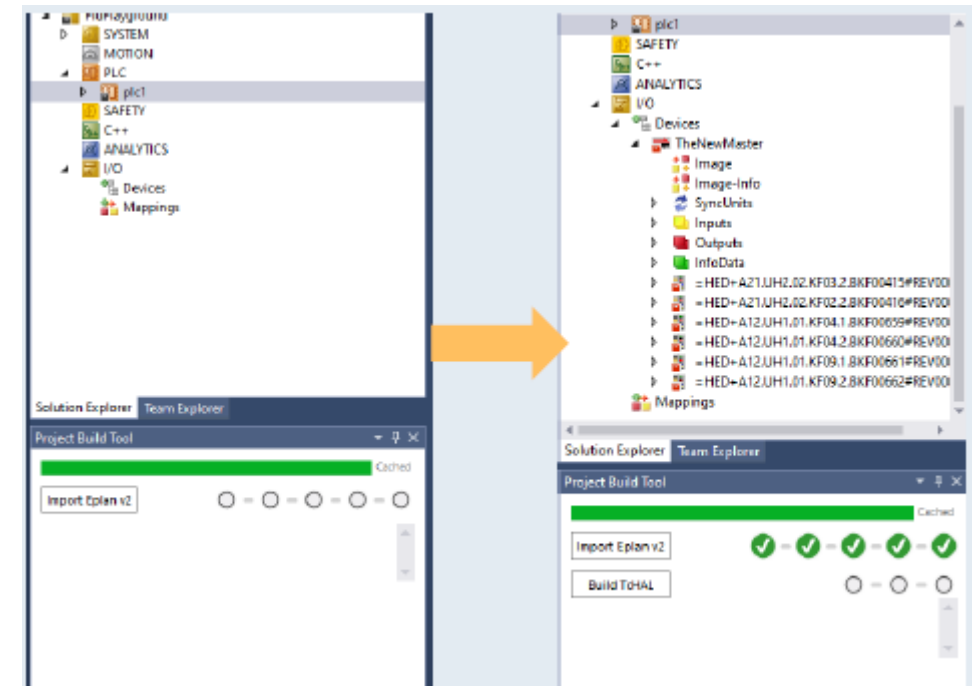
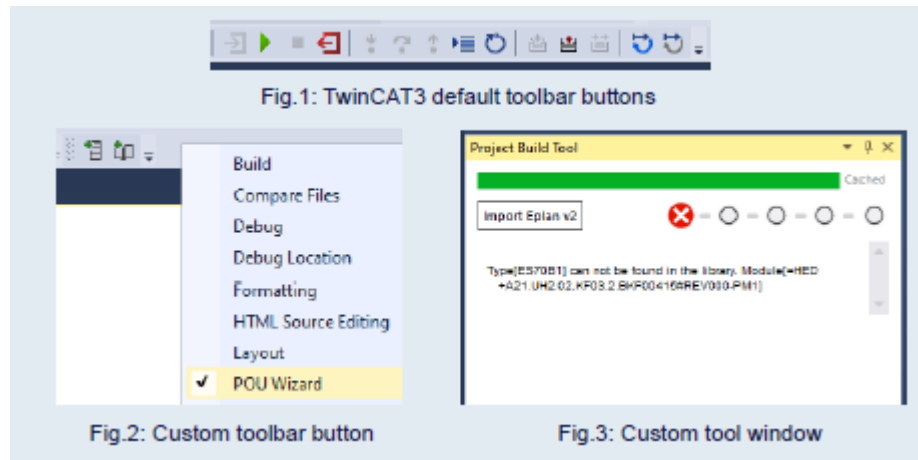
## Visual Studio Extensions I

- Utilising Visual Studio extensions to generate skeleton code based on a pre-existing template.
- Code specifics can be filled in by the developer
- Expedites development and reduces the opportunities for error



## Visual Studio Extensions II

- Modify existing POU's
- The tooling developed can be enacted as a wizard, guiding the user through a series of steps.
  - E.g. Modifying the TwinCAT hardware tree to match the EPLAN design export for a PLC project



## Support of Legacy Code – The Adapter Pattern

- 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.

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

Questions?

## INTEGRATING TOOLS TO AID THE AUTOMATION OF PLC DEVELOPMENT WITHIN THE TWINCAT ENVIRONMENT. \*

N. Mashayekh<sup>†</sup>, B. Baranasic, M. Bueno, T. Freyermuth, P. Gessler, S. T. Huynh, N. Jardón Bueno, J. Tolkiehn, L. Zanellatto, European X-Ray Free-Electron Laser, Schenefeld, Germany

### *Abstract*

Within the myriad of day to day activities, a consistent and standardised code base can be hard to achieve, especially when a diverse array of developers across different fields are involved. By creating tools and wizards, it becomes possible to guide the developer and/or user through many of the development and generic tasks associated with a Programmable Logic Controller (PLC).

At the European X-Ray Free Electron Laser Facility (EuXFEL), we have striven to achieve structure and consistency within the PLC framework through the use of C# tools which are embedded into the TwinCAT environment (Visual Studio) as Extensions. These tools aid PLC development and deployment, and provide a clean and consistent way to develop, configure and integrate code from the hardware level

Automation Interface library. This combination is highly beneficial for automated PLC project generation, code injection, and hardware linking. Visual Studio extensions enable custom tools and workflows that seamlessly integrate into the TwinCAT environment, while the TwinCAT Automation Interface library provides programmatic access to TwinCAT's features, allowing for automation of tasks and adaptation of some useful features of modern software languages. Together, these tools enhance productivity and reduce the potential for errors in automation workflows at EuXFEL.

### VISUAL STUDIO EXTENSIONS

A solid foundation provides the ideal canvas upon which