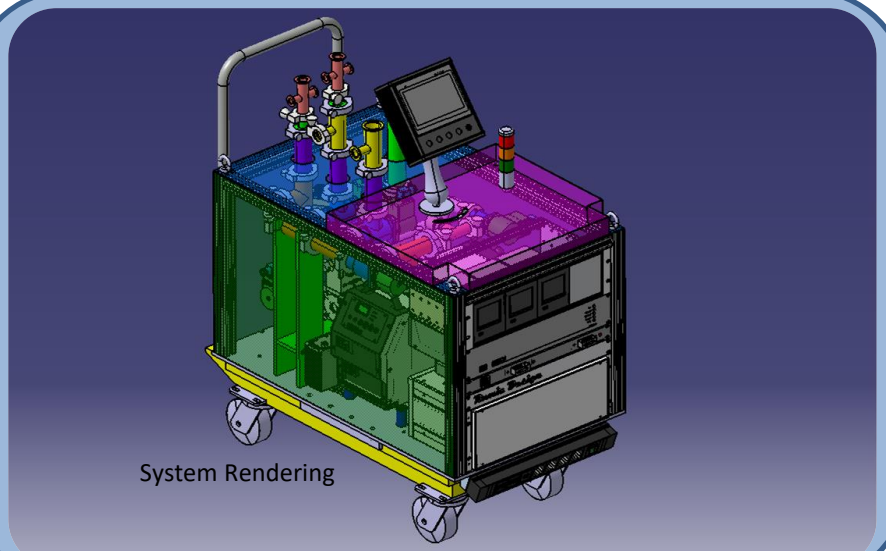
TUPV014

CONTROL SYSTEM OF A PORTABLE PUMPING STATION FOR ULTRA-HIGH VACUUM

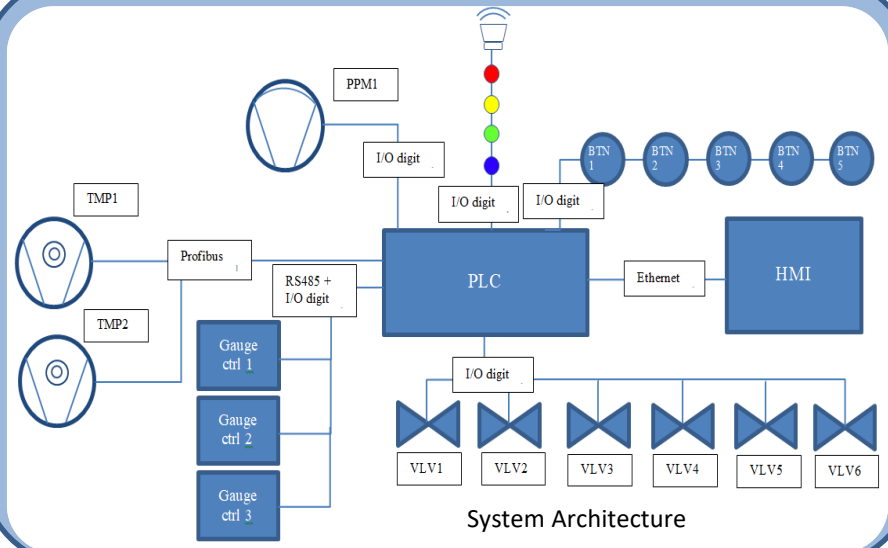
M.Trevi†, L.Rumiz, E.Mazzucco, D.Vittor, , Elettra Sincrotrone Trieste, Trieste, Italia

Elettra Sincrotrone Trieste

ICALEPCS 2021



System Rendering



System Architecture

Abstract

- We have designed a compact, independent and portable pumping station to pump the vacuum chamber and to restore the correct local pressure.
- The system automatically achieves a good vacuum level and can detect and manage vacuum leaks.
- By means of a touch screen an operator can start all the manual and automatic operations, and monitor the relevant variables and alarms.
- The system archives the operating data and displays trends, alarms and logged events; these data are downloadable on a removable USB stick.
- The control system has been implemented with a Beckhoff PLC (Programmable Logic Controller) with RS-485 and Profibus interfaces.
- This paper focuses in particular on the events management and object-oriented approach adopted to achieve a good modularity and scalability of the system

OOP PARADIGM

- Encapsulation
- Inheritance
- Polymorphism

Keywords IEC 61131 -3 not used

FB I/O interface

FB structure

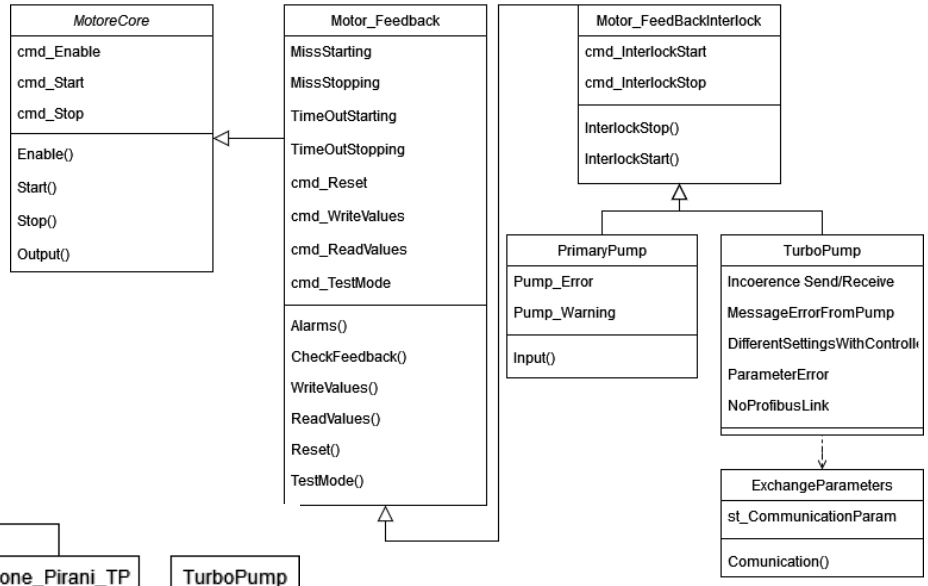
ALARMS AND COMAND MANAGEMENT

- Configuring alarms
- Who did what – Configuring Comand
- Record to export and to analyse

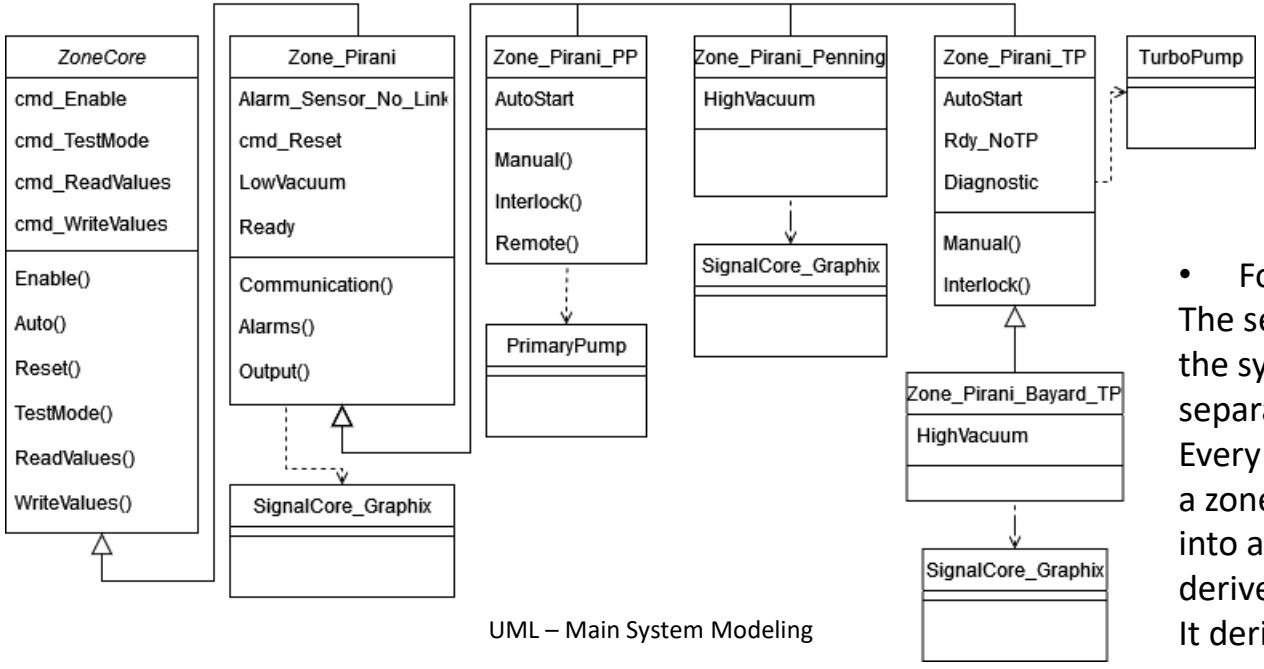
UML –Unified Modeling Language

The figures show two different UML diagram for two kinds of modeling:

- For a physical device - we have modeled two kind of different pumps (a primary pump and a turbo pump). The main difference is that the second one has the Profibus communication. But both are derived from the Motor_FeedBackInterlock class while the superclass is MotorCore



UML – Pump Modeling



UML – Main System Modeling

- For a logical function

The second modeling is how we have divided the system. A zone is a system section separated by a valve.
Every zone has at least a pressure sensor, but a zone can have a pump too. Modeling takes it into account and more complex zones are derived by a common class called Zone_Pirani. It derives from superclass ZoneCore

OBJECT ORIENTED PROGRAMMING

Elettra Sincrotrone Trieste

ICALEPCS 2021

Methods → Indipendent actions

Extends

Action structure:
- call derived FB (class);
- can add code after calling derived class if necessary;
- does not call the derived class in case of rewriting action (override)

FB I/O Interface

Inheritance with adding code

fb_S_Core.WriteValue* | fb_S_LevAlm.WriteValue* | fb_LogAlarmManager.Main*

```
g_fb_LogAlarmVacuumSystemMngr.Main(
    i_x_Trigger:= io_st_Cmd.ui_WriteValues.0,
    i_s_TextToLog := CONCAT (l_s_EffectivePath, ' - Write cmd '),
    io_ui_Code:= io_st_Cmd.ui_WriteValues,
    io_x_Get:= g_st_LogAlarmCmd.x_Get,
    io_x_Remove:= g_st_LogAlarmCmd.x_Remove,
    io_x_Reset:= g_st_LogAlarmCmd.x_Reset,
    io_x_Add:= g_st_LogAlarmCmd.x_Add);

    o_st_Status.x_ReadValues := FALSE;
// Set Scale value
    o_st_Status.dw_MaxRough := io_st_Data.dw_MaxRough;
    o_st_Status.dw_MinRough := io_st_Data.dw_MinRough;
    o_st_Status.r_MaxEng := io_st_Data.r_MaxEng;
    o_st_Status.r_MinEng := io_st_Data.r_MinEng;
```

Sensor Superclass code example

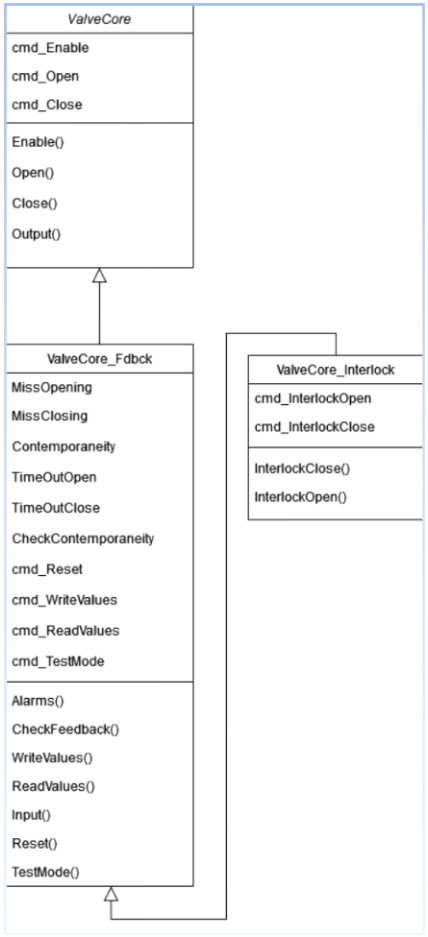
fb_S_LevAlm.WriteValue* | fb_LogAlarmManager.Main* | st_TurboPump_INOUT

```
l_fb_S_Core.WriteValue(  // write value managed by core
    i_st_Input:= i_st_Input,
    i_st_IN:= i_st_IN,
    io_st_Alarms:= io_st_Alarms,
    io_st_Cmd:= io_st_Cmd,
    io_st_Message:= io_st_Message,
    io_st_Data:= io_st_Data);

    o_st_Status.x_ReadValues := FALSE;

    o_st_Status.r_L_Threshold := io_st_Data.r_L_Threshold;
    o_st_Status.r_LL_Threshold := io_st_Data.r_LL_Threshold;
    o_st_Status.r_H_Threshold := io_st_Data.r_H_Threshold;
    o_st_Status.r_HH_Threshold := io_st_Data.r_HH_Threshold;
```

Sensor Dependent class code example



FB I/O INTERFACE

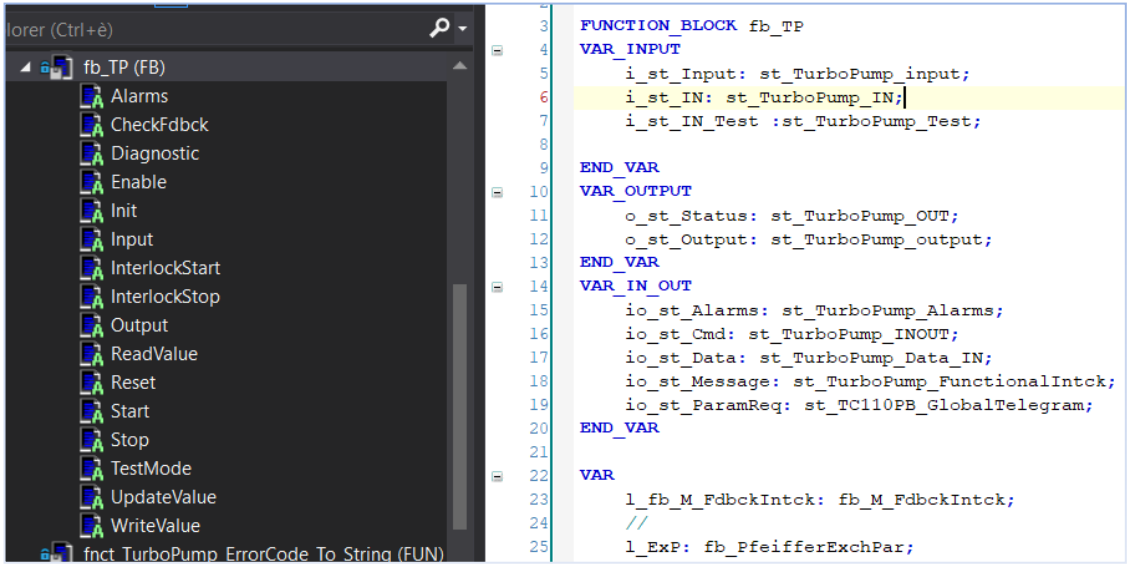
Elettra Sincrotrone Trieste

ICALEPCS 2021

Every action is separately called from outside the FB

Every FB I/O interface is divided in this way

Main body of FB is empty



FB (Function Block) code example

The parent class and the dependent classes are instanced in local variables

st_TurboPump_INOUT* | fb_TP

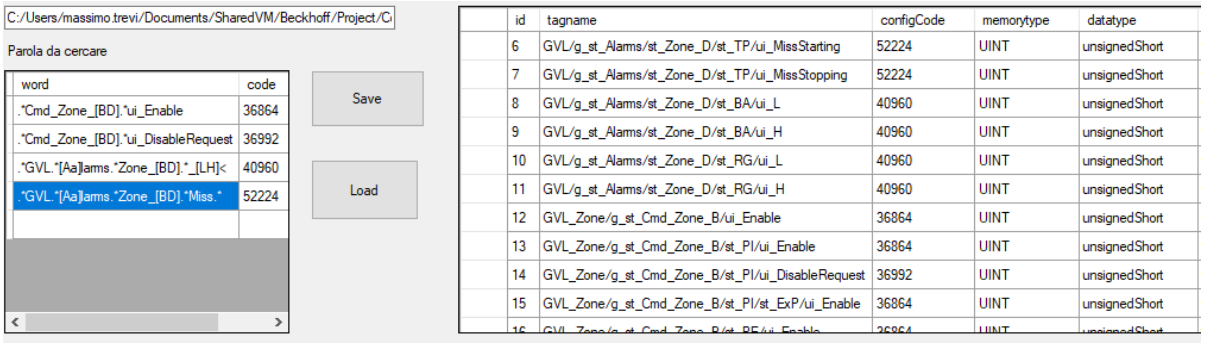
```
TYPE st_TurboPump_INOUT
STRUCT
    ui_Init:UINT:=c_ui_Log_OSComandType;
    ui_Enable: UINT:=c_ui_Log_ToggleComandType;
    ui_Auto:UINT:=c_ui_Log_RetainedComandType;
    ui_Start: UINT:=c_ui_Log_OSComandType;
    ui_Stop:UINT:=c_ui_Log_OSComandType;
    ui_Reset: UINT :=c_ui_Log_OSComandType;
    ui_Ack: UINT:=c_ui_Log_OSComandType;
    ui_TestMode: UINT:=c_ui_Log_ToggleComandType;
    ui_UpdateValues: UINT:=c_ui_Log_OSComandType; //update data from actual to output
    ui_ReadValues: UINT :=c_ui_Log_OSComandType;
    ui_WriteValues: UINT :=c_ui_Log_OSComandType; //Request to modify setting data
    ui_InterlockStart: UINT := c_ui_Log_ToggleComandType;
    ui_InterlockStop: UINT := c_ui_Log_ToggleComandType;
    ui_Diagnostic: UINT := c_ui_Log_ToggleComandType;
    ui_DisableRequest: UINT := c_ui_Log_ToggleComandType;
    st_ExP:st_PfeifferExchPar_INOUT;
END_STRUCT
END_TYPE
```

Data structure code example

ALARM and COMAND MANAGEMENT

Elettra Sincrotrone Trieste

ICALEPCS 2021



Part of Windows Form:

- Regex to find alarms and comands to find
- Setting code associated

Part of Windows Form

XML code with specific IDE language resulting by execution of program after push button on form

```
<requireAck>true</requireAck>
<blinkTxt>false</blinkTxt>
<requireReset>true</requireReset>
<actions>
    <macroAction actionFunction="setBit" parameters="ETOP7M/GVL/g_st_Alarms/st_Zone_C/st_TP/ui_MissStarting;
</actions>
<useractions/>
```

XML code extract

```
(* this code is used as IO variabile because configure kind and the status of it at th
i_usi_Code = x7,x6,x5,x4,x3,x2,x1,x0 -->
x15 = audit trail
x14 = allarme
x13 = warning
x12 = messaggio
---------------
x11 = resettabile
x10 = acknwoledge
x9 = limitAlarm (EXOR)    | -->
x8 = ValueAlarm (EXOR)    | --> se antrambi a 0, l'allarme è quella di defalut di EXOR
---------------
x7 = reserved
x6 = spare
x5 = acked
x4 = memory
---------------
x3 = spare
x2 = spare
x1 = To Acknowledge
x0 = status of alarm or comand



Codici gestiti:
1) 1100 1000 0000 0000, h#C800, 51200 - logged alarrm to reset
2) 1100 0000 0000 0000, h#C000, 49152 - logged autoresettable alarrm
3) 1010 0000 0000 0000, h#A000, 40960 - logged warning
4) 1001 0000 0000 0000, h#9000, 36864 - toggle type command
5) 1001 0000 1000 0000, h#9080, 36992 - OneShot type command
6) 1011 0000 0000 0000, h#B000, 45056 - message to log
7) 1010 0000 1000 0000, h#A080, 41088 - logged and retained command
8) 1100 1100 0000 0000, h#CC00, 52224 - logged alarm to acknowledge  and to reset
9) 1100 0100 0000 0000, h#C400, 50176 - logged autoresettable alarm to acknowledgea
*)
```

Legend and meaning of managed codes