



ARISTOTLE UNIVERSITY OF THESSALONIKI

FACULTY OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER
ENGINEERING

Master Thesis:

**Development of tools for automatic generation of
PLC code**

Koutli Maria

Supervisors:

Prof. Chasapis Georgios (AUTH)
Rochez Jacques (CERN)

April 2014





ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία:

**Ανάπτυξη εργαλείων αυτόματης παραγωγής
κώδικα για PLC**

Κουτλή Μαρία

Επιβλέποντες:
Καθ. Χασάπης Γεώργιος (ΑΠΘ)
Rochez Jacques (CERN)

Απρίλιος 2014

Acknowledgements

I would firstly like to thank, Jacques Rochez, for giving me the opportunity to work under his supervision, on this interesting topic, for these last thirteen months. During this time, he tried to pass me his knowledge and experience and he was always willing to help me, answer my questions and support me.

Moreover, I would like to thank my university supervisor, Prof. Chasapis Georgios, who supported my work from the beginning, for his advices and help.

I would also like to thank the people in the PLC section for their warm welcoming and the collaborative environment that they offered me. It has been a pleasure to meet them and work among them.

Also, I would like to give my credits to the ICE group in general, for their friendly atmosphere and their team spirit.

Finally, a big thank to my family and friends for being next to me (although far away) in this wonderful experience.

Abstract

This Master thesis was performed at CERN and more specifically in the EN-ICE-PLC section. The Thesis describes the integration of two PLC platforms, that are based on CODESYS development tool, to the CERN defined industrial framework, UNICOS. CODESYS is a development tool for PLC programming, based on IEC 61131-3 standard, and is adopted by many PLC manufacturers. The two PLC development environments are, the SoMachine from Schneider and the TwinCAT from Beckhoff. The two CODESYS compatible PLCs, should be controlled by the SCADA system of Siemens, WinCC OA. The framework includes a library of Function Blocks (objects) for the PLC programs and a software for automatic generation of the PLC code based on this library, called UAB. The integration aimed to give a solution that is shared by both PLC platforms and was based on the PLCOpen XML scheme. The developed tools were demonstrated by creating a control application for both PLC environments and testing of the behavior of the code of the library.

Περίληψη

Η παρούσα Διπλωματική Εργασία εκπονήθηκε στο CERN και συγκε-
κριμένα στον τομέα EN-ICE-PLC. Η εργασία περιγράφει την ενσωμάτωση
δύο κατασκευαστών Προγραμματιζόμενων Λογικών Ελεγκτών (PLC), οι
οποίοι έχουν ως βάση τους το εργαλείο ανάπτυξης λογισμικού CODESYS,
σ' ένα πλαίσιο που έχει καθοριστεί στο CERN και αφορά τις βιομηχανικές
εφαρμογές, το UNICOS. Το CODESYS είναι ένα εργαλείο ανάπτυξης
λογισμικού για PLC, που βασίζεται στο πρότυπο 61131-3 της IEC και
έχει υιοθετηθεί από πολλούς κατασκευαστές PLC. Οι δύο πλατφόρμες
PLC που χρησιμοποιήθηκαν, είναι το SoMachine της Schneider και το
TwinCAT της Beckhoff. Τα δύο PLC θα πρέπει να ελέγχονται μέσω ενός
συστήματος ελέγχου και τηλεμετρίας (SCADA) της Siemens, το WinCC
OA. Το πλαίσιο του UNICOS περιλαμβάνει μία βιβλιοθήκη με κώδικα για
PLC εφαρμογές και ένα λογισμικό αυτόματης παραγωγής PLC κώδικα
βασισμένου σ' αυτή τη βιβλιοθήκη, το UAB. Η ενσωμάτωση των δύο
αυτών κατασκευαστών PLC είχε σκοπό να παρέχει μία λύση που είναι
κοινή και για τους δύο και βασίστηκε στο XML σχήμα PLCOpen XML. Η
λειτουργικότητα των εργαλείων που αναπτύχθηκαν, αναδείχτηκε μέσω
της δημιουργίας μιας βιομηχανικής εφαρμογής ελέγχου και την εφαρμογή
τεστ για την απόκριση του κώδικα της βιβλιοθήκης και για τις δύο
πλατφόρμες.

Contents

Abstract	ii
Περίληψη	iii
Contents	v
List of Figures	vii
List of Tables	viii
List of Acronyms	ix
1 Introduction	1
1.1 About CERN	1
1.2 About EN-ICE-PLC section	1
1.3 Subject of the thesis	2
1.4 Methodology and goals	3
1.5 Structure of the thesis	4
2 Standards for programmable controllers	5
2.1 A general introduction to programmable controllers	5
2.2 The IEC 61131 standard	5
2.3 The IEC 61131-3 standard	6
2.4 The <i>PLCOpen</i> Organization	9
2.5 Description of the PLCOpen XML scheme	9
2.6 CODESYS V3-an industrial IEC 61131-3 PLC programming platform	10
3 The UNICOS framework and the UAB tool	13
3.1 The UNICOS framework	13
3.1.1 General	13
3.1.2 The UNICORE component	15
3.1.3 The CPC component	16
3.1.3.1 CPC Objects	17
3.1.3.2 TSPP/Modbus communication	20
3.2 The UAB tool	22
3.2.1 The Core	27
3.2.2 The Plug-ins	27
3.2.3 The Resource Package	28
4 CODESYS plug-ins and Resource Package	30
4.1 Modification of the UAB CODESYS plug-ins	30
4.1.1 Instance Plug-in	33

4.1.2	Logic Plug-in	35
4.2	Creation of the CODESYS Templates	37
4.2.1	Instance Templates	38
4.2.2	Logic Templates	41
4.3	Creation of CODESYS Baseline	43
4.3.1	CPC Objects library	44
4.3.2	Modbus TSPP Communication library	45
5	Demonstration and testing of CODESYS Plug-ins and Resource package	51
5.1	An example of platform independence on a real control application	51
5.1.1	Description of the control application	51
5.1.2	UAB generation procedure	56
5.1.2.1	UAB Inputs	56
5.1.2.2	Generation of the output files	59
5.1.3	Importation to Somachine	65
5.1.4	Importation to TwinCAT	69
5.1.5	Supervision with WinCC OA	72
5.1.5.1	Control of the application	75
5.1.6	Supervision with Magelis Vijeo Touchpanel	76
5.1.7	Hardware of the application	77
5.2	Validation of Field Objects	79
5.3	CODESYS and Process Simulation	81
5.3.1	Twincat	81
5.3.2	SoMachine	83
6	Conclusions	85
6.1	General	85
6.2	Future possibilities	85
	Appendix A' CPC Objects	87
	Appendix B' Mapping of addresses	87
	Appendix Γ' Field Object Test	88
	Appendix Δ' PLCOpen XML and CODESYS XML file formats	88
	References	100

List of Figures

1	UNICOS-CPC and CODESYS	3
2	Work-flow diagram	4
3	PLCOpen XML scheme [13]	11
4	The 3 layers of a UNICOS control application [3]	14
5	IEC61512-1 Continuous Process Control Model [7]	14
6	UNICOS Architecture [9]	15
7	Analog Object	17
8	CPC Objects Architecture	21
9	Baseline library	22
10	MODBUS TSPP Communication	23
11	The UAB tool	24
12	The UAB Bootstrap	25
13	Creation of a UAB Application	26
14	CPC Wizard panels	27
15	Plug-ins used for creating a CPC CODESYS application [16]	30
16	The execution order of the POUs in the topology file.	33
17	Java Classes for Instance generation	34
18	Modification of the Instance Plug-in	36
19	Instance Template Structure	39
20	Creation of Instances file	40
21	Logic Templates categories	42
22	Middleware Communication	46
23	Creating a CODESYS application with UAB [16]	52
24	Overview of the process	53
25	Process decomposition	54
26	User Template definition in the Spec file	56
27	Specifications file-Analog Input objects	58
28	Create a new UAB application	59
29	Application General Data	60
30	PLC Specifications	61
31	CoDeSys Generators	61
32	Instance generator	62
33	Logic Generator	63
34	WinCC OA Generator	64
35	Touchpanel Generator	65
36	Log Window	65
37	Demonstrator SoMachine	67
38	Connection to M258	67
39	Import PLCOpen XML files into SoMachine	68
40	Choose the objects for importation	68
41	Connect to TwinCAT target	69
42	Import PLCOpen XML files to TwinCAT	70

43	TwinCAT input/output variables	71
44	Demonstrator-I/O modules in the TwinCAT application . .	72
45	TwinCAT menu	72
46	Import Demonstrator's database in WinCC OA 3.11	73
47	Front-End Diagnostics	74
48	Navigation panel	74
49	Parametrization of SoMachine panel in the Window tree . .	75
50	WinCC OA panel for Demonstrator	76
51	Magelis Vijeo Touchpanel	77
52	Hardware equipment	79
53	Full Stop Test for OnOff object	80
54	TwinCAT- OPC Configuration	83
55	WinCC OA panel of the QSDN application	83
A'.1	I/O and Field Objects	89
A'.2	Interface and Control Objects	90
B'.1	PLC variables mapping	91
B'.2	Status,Event and Request Registers	91
B'.3	Binary and Analog Status Registers	92
Γ'.1	Test WinCC OA operation orders	93
Γ'.2	Test Full Stop behavior	94
Γ'.3	Test Temporary Stop behavior	95
Γ'.4	Test Start Interlock behavior	96
Δ'.1	A PLC program in PLCOpen XML and in CODESYS XML file format	98

List of Tables

1	Functions and FBs used by CPC objects	44
2	The CPC objects of the Demonstrator application	57
3	Schneider's M258 I/O modules for Demonstrator	66
4	Beckhoff's EPC I/O modules for Demonstrator	71
5	Beckhoff's IPC I/O modules for Demonstrator	71
6	Sensors and actuators of the Demonstrator application . . .	78

List of Acronyms

CERN	the European Organization for Nuclear Research
CPC	Continuous Process Controls
EPC	Embedded PC
FB	Function Block
GUI	Graphical User Interface
GVL	Global Variable List
ICE	Industrial Controls and Engineering
IEC	International Electrotechnical Commission
I/O	Input/Output
IPC	Industrial PC
JCOP	Joint COntrols Project
LHC	Large Hadron Collider
OLE	Object Linking and Embedding
OPC	OLE for Process Control
PAC	Programmable Automation Controller
PC	Personal Computer
PID Controller	Proportional-Integral-Derivative Controller
PLC	Programmable Logic Controller
POU	Program Organization Unit
RTC	Real Time Clock
SCADA	Supervision Control and Data Acquisition
TC	Technical Committee
TSPP	Time Stamped Push Protocol
UAB	UNICOS Application Builder
UNICOS	UNified Industrial Control System
UTC	Coordinated Universal Time (Temps Universel Coordonné)
XML	eXtended Markup Language

1 Introduction

1.1 About CERN

CERN, the European Organization for Nuclear Research, was founded in 1954 by 12 member states¹ and is located at the Franco-Swiss borders near Geneva. Nowadays it has 20 member states and collaborates with over 600 institutes and universities around the world, among them the Aristotle University of Thessaloniki. Its main research focuses on particle physics and it has an expanded physics program varying from the study of the Standard Model to the quest of the effects of cosmic rays in clouds. The research is divided in many experiments with most popular the four big LHC experiments, ATLAS, CMS, ALICE, LHCb. High-energy physics require accelerating particles to almost the speed of light to energies as high as 4 TeV. Inside the LHC, accelerated particles collide at the four experiment points, where detectors are used to detect the produced particles, right after the collisions.

A lot of engineering effort is required to accomplish the system's operation. Super conducting magnets, cooled down with helium to 1.9 K (-271.3°C), are used to bend the particles during their travel in the LHC ring. In the control centers, servers processing around 1050 MB of data per second, need to be cooled down in order to operate. Also, different types of gas have to be provided with the right pressure, quality and mixture to particle detectors. The above are few examples of the many industrial-like applications needed to be controlled and monitored at CERN in order to have the whole system running properly.

1.2 About EN-ICE-PLC section

As part of the Engineering Department (EN) of CERN, the ICE (Industrial Controls and Engineering) Group deals with the industrial controls. The control systems at CERN are based on industrial equipment and are configured with CERN-developed frameworks as UNICOS, JCOP, RADE. The ICE group consists of the four following sections: SCADA Systems (SCADA), Measurement, Test and Analysis (MTA), PLC and Front-ends (PLC), Software Interfaces and Components (SIC).

There is a huge variety of control applications covered, concerning cooling and ventilation, cryogenics, vacuum, gas and safety systems. The above applications are developed for the two CERN supported PLC (Programmable Logic Controller) brands, Siemens and Schneider. The programming of the PLCs is based on UNICOS (UNified Industrial Control System) framework [7] and more specific on the CPC (Continuous Pro-

¹Belgium, Denmark, France, the Federal Republic of Germany, Greece, Italy, the Netherlands, Norway, Sweden, Switzerland, the United Kingdom, and Yugoslavia.

cess Controls) package [15] which is dedicated to the PLC programming. UNICOS defines a set of device types (Objects) for both control and supervision. It also offers an automatic code generation software for PLCs, called UAB (UNICOS Application Builder), which is based on this framework.

UAB is developed at CERN and more specific in the EN-ICE group. Among other projects it is used for the generation of PLC code for the two CERN approved industrial PLC companies, Siemens and Schneider. Dedicated UAB Plug-ins and the respective Resource Package (RP),² as well as a library of the CPC Function Blocks, known as Baseline, have been developed for each PLC brand. The UAB tool has been developed in *Java* and *Jython* programming languages.

1.3 Subject of the thesis

CODESYS(ConTroller DEvelopment SYstem) V3 is a PLC development software of 3S company and has been the new integrating platform in UNICOS-CPC (UCPC) and consequently in UAB. The big range of CODESYS compatible industrial controllers and the support of IEC 61131-3 PLC programming by the tool, have led to the choice of this integration. A first implementation of CODESYS V3 in CPC version 6 was performed by another student's project [12]. A library of the CPC Objects (Baseline), two plug-ins and their respective RP for the UAB tool, had been created for that project. Those plug-ins generated the importation files for the CODESYS-based, PLC development software, SoMachine. The output files of the plug-ins and the library files had CODESYS XML file format. The target was a Schneider SoMachine M258 controller.

The initial goal of this project was the integration of a second CODESYS based development environment, Beckhoff TwinCAT. The aim was to give a complete solution for creating industrial control applications, based on UCPC framework, with a Beckhoff controller programmed with the use of the UAB tool, and WinCC OA SCADA supervision³. The need for standardization and vendor independence, which is one of the main claims of CODESYS software, since it supports the IEC 61131-3 standard, became clear from the first steps of this project. Consequently, a unified solution for the two CODESYS-based platforms, SoMachine and TwinCAT, became the main goal of the project. The proof of this concept would open the floor to new CODESYS-based PLC platforms for standardized and hardware independent control applications at CERN.

UCPC principles should be followed in order to create a CPC Object Baseline, the UAB plug-ins and the respective Resource Package for the development of a control application. The above should serve both So-

²The RP contains the inputs of the plug-ins for the generation.

³A Siemens HMI Software former known as PVSS, officially supported at CERN

Machine and TwinCAT, and potentially any other CODESYS-based development environment for the creation of a PLC application. The relation between UCPC framework and a control application developed for a CODESYS compatible controller can be seen in figure 1.

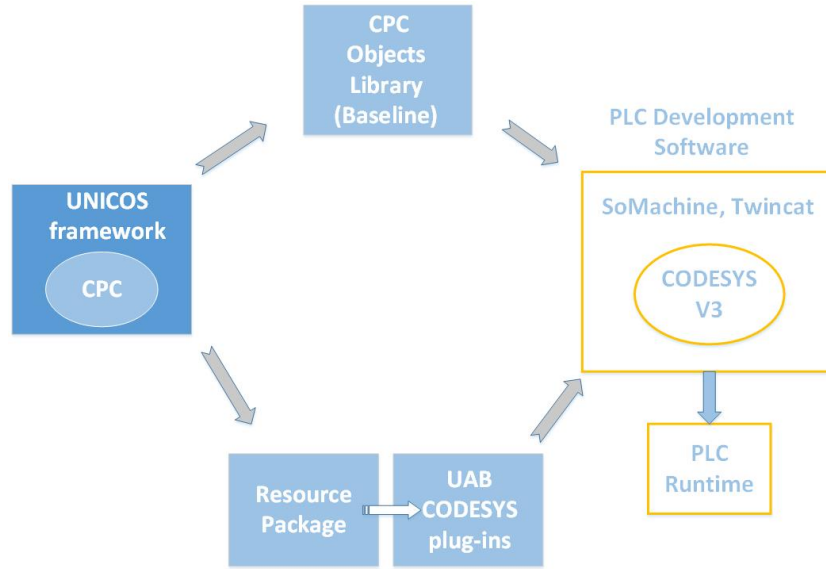


Figure 1: UNICOS-CPC and CODESYS

1.4 Methodology and goals

For this second implementation of CODESYS V3 to UCPC, the UAB plug-ins should be able to generate the importation files with the PLC code for both TwinCAT and SoMachine software. In the Resource Package, the *Jython* Templates that are used by the plug-ins for the generation should be provided as well. Also a CPC Baseline library should be created for both systems. TwinCAT 3.1 and SoMachine 3.0 are the two development software versions that were used for this project. TwinCAT version 3 and higher does not support the CODESYS XML importation files that have been used for SoMachine in the previous implementation. Thus, a new solution for the UAB output file format for CODESYS had to be found.

The answer to this problem and to the need for standardization came from the *PLC Open* organization. The *PLC Open* organization provides a standard XML scheme for all IEC 61131-3 languages. Fortunately, both TwinCAT 3.1 and SoMachine 3.0 support the PLCOpen XML standard for their importation files, so it was chosen to be the output file format of the UAB plug-ins for CODESYS.

Apart from the changes concerning the UAB tool, improvements in the communication with the SCADA system WinCC OA and a first release

of the CPC Baseline library for CODESYS needed to be done. Moreover, testing and validation of the functionality of the solution and generally enhancements and completions since the previous implementation had to be performed. The diagram in figure 2 shows the work-flow that was followed.

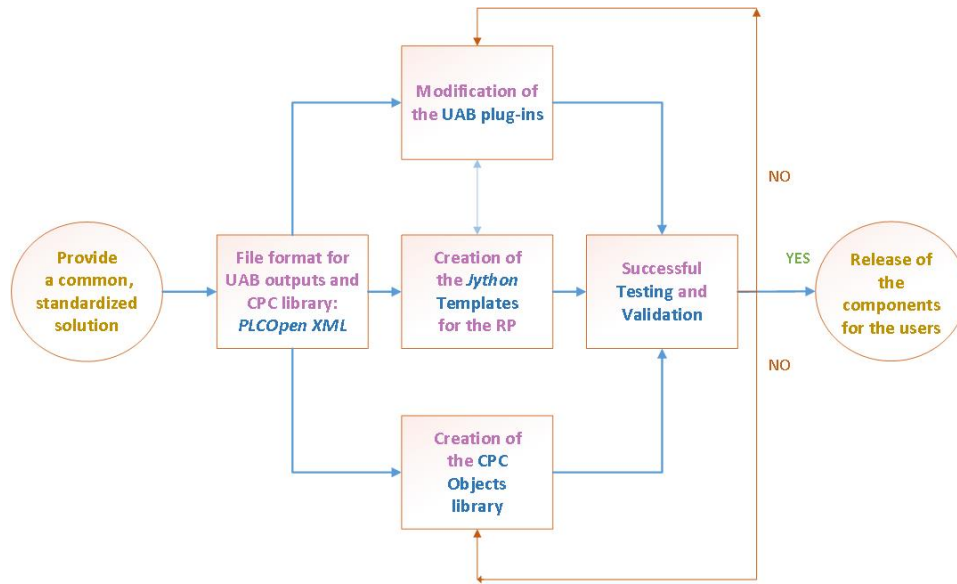


Figure 2: Work-flow diagram

1.5 Structure of the thesis

In this thesis the project is analyzed in six chapters. In this chapter there is a general introduction to the problem and a quick description of the solution that was given. The second chapter refers to the standards that were used in this implementation and the third chapter describes the framework on which this work was based. In the fourth chapter the solution which was given is described in detail and in the fifth chapter the demonstration of the developed tools and the tests that were performed to the Baseline library are listed. Finally in the sixth chapter the conclusions of the whole project are mentioned.

2 Standards for programmable controllers

2.1 A general introduction to programmable controllers

In the late 1960s the American motor car manufacturer General Motors was interested in the application of computers to replace the relay sequencing used in the control of its automated car plants. In 1969 it produced a specification for an industrial computer to which Bedford Associates (later called Modicon) and Allen Bradley, responded. Each, produced a computer system which had little resemblance to the commercial minicomputers of the day. The computer was designed to survive in industrial environments under dirt, dust, very high or very low temperatures and vibrations, which are rather different requirements than the ones for a conventional computer. This industrial computer was connected to input and output cards which deal with digital and analog signals at the usual voltages encountered in industry (24V DC to 230V AC). The program was stored in battery-backed up or non-volatile memory. Also the system's cycle had to be very accurate with a real-time response (around 1ms) depending on the application. [11]

In the following years there was a rapid evolution of conventional computers (PCs) which was not followed with the same rhythm by the PLC industry. In the recent years PLC manufacturers tend to exploit more and more the technology that already exists for conventional PCs and combine it with the PLC capabilities. This combination has given new PLC successors⁴. PACs (Programmable Automation Controllers) are controllers with more functionality offering a combination of logic, motion and continuous control in the same hardware platform and many communication standard choices. ARC Advisory Group found this term in order to differentiate PLCs and others machines with more advanced features, though nowadays the distinction is quite difficult due to the parallel evolution of PLCs. The IPC (Industrial PC) is a general purpose PC which was designed to be used in industrial environments. These controllers, which are powerful machines, are able to run faster and can handle special tasks requiring more computing capabilities than a classic PLC. Programmable controllers are continuously evolving focusing on new hardware technologies, high level programming and communication by using a variety of standards.

2.2 The IEC 61131 standard

PLCs have been the main implementation platform for industrial control system applications for many years, since their first appearance in

⁴The abbreviations PLC, IPC, EPC and PAC will be used in this document to stand for programmable controllers, as is the common practice in the automation industry.

1969. Various manufacturers have developed PLC hardware and software with different Runtime environments, operating systems, and programming languages. This has lead to many incompatibilities between the different suppliers and the need for standardization. In order to standardize and reduce the complexity for the users, the International Electrotechnical Commission (IEC) released the standard IEC 61131. [10]

IEC 61131 consists of the following parts under the general title: Programmable controllers.

- Part 1: Establishes the definitions and identifies the principal characteristics relevant to the selection and application of programmable controllers and their associated peripherals.
- Part 2: Specifies equipment requirements and related tests for programmable controllers (PLC) and their associated peripherals.
- Part 3: Defines, for each of the five most commonly used programming languages, major fields of application, syntactic and semantic rules, basic sets of programming elements and applicable tests.
- Part 4: Gives general overview information and application guidelines of the standard for the PLC end-user.
- Part 5: Defines the communication between programmable controllers and other electronic systems.
- Part 6: Is reserved.
- Part 7: Defines the programming language for fuzzy control.
- Part 8: Gives the guidelines for the application and implementation of programming languages, defined in Part 3, for programmable controllers.
- Part 9: Defines a single-drop digital communication interface for small sensors and actuators (SDCI) (commonly known as IO-Link), which extends the traditional digital input and digital output interfaces as defined in IEC 61131-2 towards a point-to-point communication link.

2.3 The IEC 61131-3 standard

IEC 61131-3 [8] is the third part of IEC 61131 standard for programmable logic controllers, and was first published in December 1993 by IEC organization. The 2nd edition was released in 2003 and currently there is a 3rd edition which was published in February of 2013. This last

edition is fully compatible with the previous one, adding more extensions, like for Object Oriented Programming (OOP).

Looking closer to IEC 61131-3 standard it can be seen that it gives guidelines for two main fields concerning, the Programming Languages and a software model of Common Elements.

IEC 61131-3 specifies the syntax and the semantics of five programming languages for PLCs, embedded controls, and industrial PCs. It defines two textual languages, Instruction List (IL) and Structured Text (ST), and two graphical languages, Ladder Diagram (LD) and Function Block Diagram (FBD). An additional set of graphical and equivalent textual elements named Sequential Function Chart (SFC) is defined for structuring the internal organization of programs and function blocks. The choice of programming language is dependent on the control system and on the programmer's background. Ladder Diagram has its roots in USA. It is based on the graphical presentation of Relay Ladder Logic. Instruction List is its European counterpart. As textual language, it resembles assembly. Function Block Diagram is very common to the process industry. It expresses the behavior of functions, function blocks and programs as a set of interconnected graphical blocks, like in electronic circuit diagrams. Structured Text is a very powerful high-level language with its roots in Ada, Pascal and "C". It contains all the essential elements of a modern programming language, including selection branches (IF-THEN-ELSE and CASE OF) and iteration loops (FOR, WHILE and REPEAT). These elements can also be nested. It can be used excellently for the definition of complex function blocks, which can be used within any of the other languages.

The standard specifies also the logical structure of a programming language(common elements), including:

- **Configuration:** This language element represents a programmable controller system as defined in IEC 61131-1⁵. Within a configuration one or more Resources can be defined.
- **Resources:** They correspond to a "signal processing function", its "human machine interface" and "sensor and actuator interface" functions. It is a processing facility like a CPU. Within a resource, one or more Tasks can be defined.
- **Tasks:** An execution control element provided for periodic or triggered execution of a group of associated programs. It mainly contains the list of programs that will be executed in its cycle and settings as the cycle time, the priority etc.

⁵User-built configuration, consisting of a programmable controller and associated peripherals, that are necessary for the automated system.

- **Programs:** They are the highest level program organization unit (POU). They are built from a number of different software elements written in any of the IEC defined languages. Typically, a program consists of a network of Functions and Function Blocks, which are able to exchange data.
- **Functions and Function Blocks:** Are the basic building blocks, containing a data-structure and an algorithm.

Other common elements also defined in the standard are the elementary data types (BOOL, INT, REAL, WORD etc.) used in PLC programming. Additionally to these types, generic and derived (user or manufacturer specified) data types can be declared. Furthermore the variables are defined. A variable can be declared to be one of the elementary types or one of the derived types and can be assigned to a hardware address (input %I, output %Q, memory %M). The scopes of the variables are normally limited to the organization unit in which they are declared, e.g. local. This means that their names can be reused in other parts without any conflict. If the variables should have global scope, they have to be declared as global variables. Parameters can be assigned with an initial value at start up and cold restart, in order to have the right setting.

Advantages of the standard

Since IEC 61131 is an international accepted standard many vendors are expected to gradually make programming software which supports it. Although it might be a new start for a programmer who is used to program on other principles in the end it reduces the time and effort to learn programming in different platforms, since the programmer has to learn it once and then can apply it to many different controllers. This means reduced waste of human resources, in training, debugging, maintenance and migrating from devices manufactured by different vendors or porting a program from one vendor's programming software to another. Furthermore since programming philosophy is common the most suited platform for the real needs of a control system can be selected regardless of the supplier. Also supplier specific language-extensions that prevent the inter-changeability can be avoided and fewer errors occur because of defined data types. Finally these programming techniques can be usable in a broad environment: general industrial control combining different components from different programs, projects, locations, companies and/or countries and an availability of high-level programming languages.

2.4 The *PLCOpen* Organization

The *PLCOpen* [13] is a vendor-independent organization which was founded in 1992 after the IEC 61131-3 standard was published. Its members, which represent various industries, joined together in order to achieve harmonization of control programming and interfacing engineering. The organization promotes and gives technical specifications around IEC 61131-3 complying systems in order to reduce cost in engineering. This includes certification and exchange. There are six technical and six promotional committees working within *PLCOpen* organization. The Technical Committees (TC), with representatives of *PLCOpen* members, work on specific items enhancing standardization.

TC1 deals with the standards focusing mainly on IEC 61131-3.

TC2 is dedicated to motion control. It provides a standard which defines motion control libraries which are hardware independent. It also includes rules for compliance and certification.

TC3 is about certification and conformity testing. There are three levels of certification, Base, Conformity and Reusability levels.

TC4 together with OPC Foundation since 2008 they try to combine their technologies to a vendor-independent communication architecture.

TC5 defines safety concepts and FBs for IEC 61131-3 control applications. And finally,

TC6 defines a standard XML scheme that enables exchange of PLC programs between different software platforms.

2.5 Description of the *PLCOpen* XML scheme

The IEC 61131-3 standard does not define a standard file exchange format between different manufacturers. To solve this problem *PLCOpen* TC6 has defined an open extensible markup language (XML) interface to achieve inter-changeability between development environments. The first version of the scheme was published in June of 2005. Two more have followed on December of 2008 and on May of 2009. CODESYS since version 3.3 SP1 in late 2009 started to gradually support the scheme. Also AutomationML, a group of well known industrial manufacturers, is contributing and supporting part of this scheme for sequencing.

The *PLC Open* XML scheme [14] supports all the five languages defined by the IEC 61131-3. This means that apart from textual information also graphical information can be transferred, like where the blocks are and how they are connected.

The use of the XML language has many advantages first of all because it is extendable. Also an XML document can easily be checked for its consistency with the scheme that is provided⁶. This file format apart from exchange of PLC code between different industrial developing software is supported by other software tools, like modeling or visualization tools.

Development platforms that support this scheme allow the export and import of a whole Configuration(as defined in IEC 61131-1) in this format. This includes the expression of many IEC 61131-3 elements as the:

- Textual Programming Languages – IL and ST
- Graphical Programming Languages – LD, FBD
- Structural Language – SFC
- Graphical Information, like place and position, and routing of connections
- Comments
- Program Organization Units – (User Derived) Functions, Function Blocks and Programs
- (User Derived) Data types
- Variables (Local, Global)
- Project information (layered structure)
- Mapping information
- Task Configuration
- Supplier specific information

The scheme structure for some of the POU's attributes can be seen in figure 3.

2.6 CODESYS V3-an industrial IEC 61131-3 PLC programming platform

CODESYS is a development tool for IEC 61131-3 and hardware independent PLC programming by 3S (Smart Software Solutions GmbH) company. 3S was founded in 1994 and version 1.0 of the product was released the same year. The company was also a member in the German working group for the recent 3rd edition of the IEC 61131-3. Through the

⁶The scheme can be accessed for free at the PLCOpen Organizations website [13].

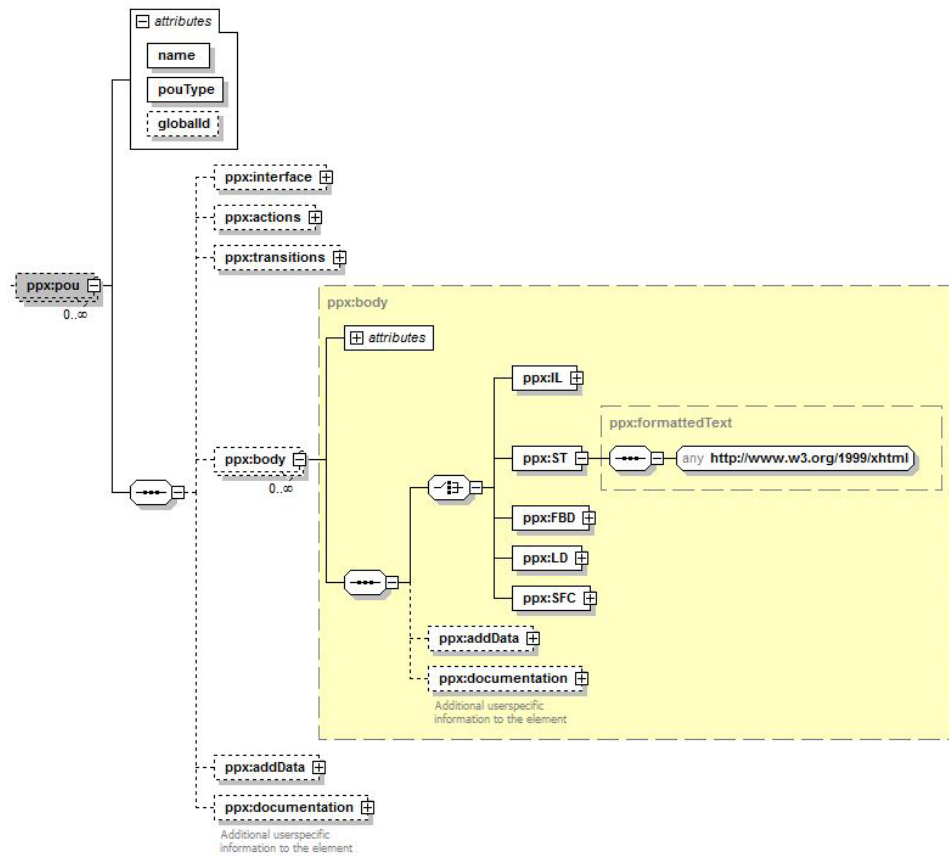


Figure 3: PLCOpen XML scheme [13]

years the product has reached maturity proving its reliability and robustness in industry. Many manufacturers desiring to follow the IEC 61131-3 standard have selected CODESYS to extend their device specific software. The product is license free, which one of the reasons that it became so popular and was adapted by so many manufacturers (around 300).

Its integrated compiler transforms the compiled code into native machine code which is supported by the most popular CPU families, like ARM-based CPUs, Intel 80x86, 80186, Pentium and many more. Also CODESYS runtime system is available for many embedded systems and PC based platforms and also offered for many operating systems as Windows(XP/7/CE), LINUX (OSADL), VxWorks. CODESYS offers a control runtime development kit, which is used by hardware manufacturers in order to implement the PLC runtime system of CODESYS in their controller. A SoftPLC solution(as target system) is offered for Windows PC-based systems that gives the opportunity to run a PLC program in almost any Windows PC.

Finally, nowadays many manufacturers offer a large choice of CODESYS compatible devices, as PACs and IPCs. In opposition to a standard PLC, the internal memory of an IPC is far less constrained in terms of size and IPCs running CODESYS can therefore integrate complex algorithms developed in high level programming languages (e.g.C++). [16], [1]

The above reasons lead to the choice of development platforms based on CODESYS V3, as the new implementation platforms for UNICOS Continuous Process Control (CPC) applications and the PLCOpen XML format as the generated output file format of the UAB tool, which is used for importation to the PLC development software and provides the PLC code based on IEC 61131-3 principles.

3 The UNICOS framework and the UAB tool

3.1 The UNICOS framework

3.1.1 General

UNICOS is a CERN defined framework, first introduced at CERN control systems in early 2001, for the development of the LHC cryogenics control system. Since then, it is used at CERN for developing control applications for two or three layers control systems [9]. The three layers of a typical control system are the Supervision, the Control and the Field layer. The supervision layer refers to control and monitoring of the industrial application, by SCADA systems. The Control layer is based on industrial front-ends, which execute the control logic and the field layer consists of the sensors and the actuators, connected to the front-ends via I/O cards or field buses. Field layer components are either controlled by commands sent to them by the logic deployed in the control layer or they sent signals to this logic.

Laboratories, as CERN, which are developing control systems had to face the problem of different solutions for each layer by different suppliers which could not easily be combined. Possible solutions could be the Distributed Control Systems (DCS) or some companies which offer integrated solutions for the three layers. The first solution would increase a lot the cost of the control system and also some implementations would be difficult to be integrated to accelerator and detector complex systems. The second one implied a different solution per supplier, which would make the laboratory dependent on an external company for the control hardware and software equipment [7].

UNICOS framework was finally the solution given for unifying the SCADA, PLC and field buses layers under one standard. The framework provides developers of SCADA and PLC applications a library of objects for all the items of the process (CPC Component) and guidelines for programming the logic of the control application based on this library. The objects represent hardware devices in the process like an On-Off valve or a pump and "software devices" like a PID Controller or a Process Control unit. These library components are used as common language by the process engineers and programmers to perform the functional analysis⁷ of the control application. In figure 4, the three layers of a UNICOS control system are presented.

The framework was based on IEC 61512-1(former ISA 88.00) standard's approach for process modeling by dividing the process in a hierarchy of objects. For continuous process control the standard defines the

⁷A document with the description of the process based on UNICOS objects. This document gives the guidelines for filling the specifications file

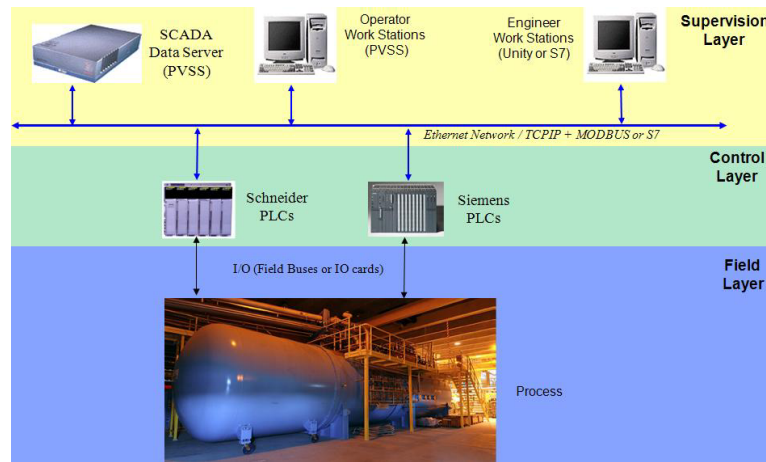


Figure 4: The 3 layers of a UNICOS control application [3]

following three modules. The Unit module has complex processing activities. The Equipment module has limited processing activities and controls specific equipment. And, the Control module corresponds to the sensors and actuators of the control system. The hierarchy of the modules can be seen in figure 5. The first implementation of this norm for UNICOS objects included three sub-categories of objects: I/O, field and process control (interface objects are added later). The I/O and Field objects are considered as Control modules and the Process Control objects can be considered either as Equipment or as Unit modules according to the complexity of their process logic.

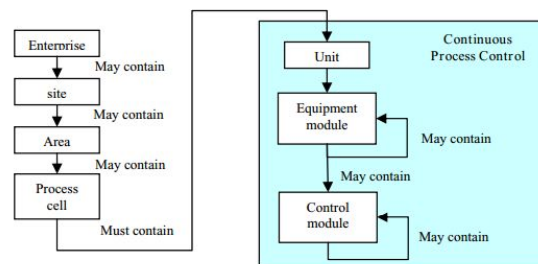


Figure 5: IEC61512-1 Continuous Process Control Model [7]

UNICOS framework is composed of reusable libraries for both supervision and front-ends and consists of two main components: UNICORE and Continuous Process Control (CPC), figure 6.

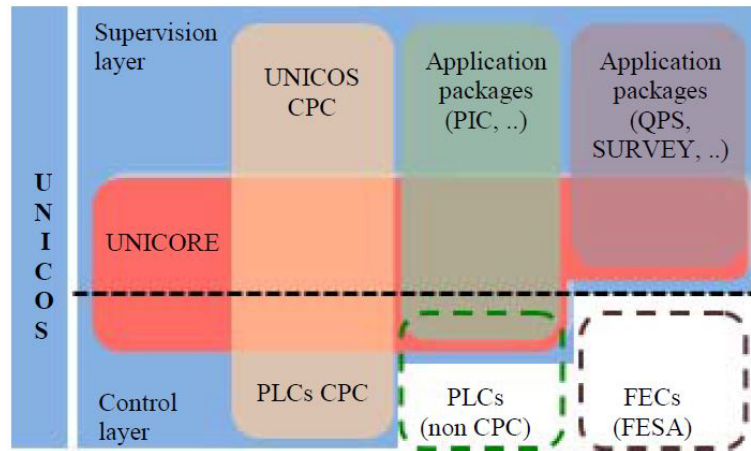


Figure 6: UNICOS Architecture [9]

3.1.2 The UNICORE component

UNICORE covers the development in the supervision and the process control layers. New packages can be added to it to fulfill the different features for each application. Supervision development is based on WinCC OA (former PVSS) SCADA tool, enhancing its capabilities for UNICOS applications. UNICORE is providing many user interfaces for monitoring (e.g. Client and Server Controls MiddleWare (CMW)) and adding new packages. Moreover, for monitoring a browser-like interface is offered in order to navigate through the application's panels and access to devices is possible even without creating a panel, by the tree device overview. Also, process alarm and event lists are featured. UNICORE also offers application distribution in many SCADA data servers and enables device and file access control. These features give a lot of capabilities to the users without demanding a WINCC OA scripting language knowledge [9].

For the front-end layer UNICORE provides the communication protocol between the PLC and the SCADA for an Ethernet TCP/IP connection, the so-called TSPP (Time Stamp Push Protocol) [4]. UNICOS TSPP is an event driven communication protocol that extends MODBUS and S7 protocols that are used from Schneider, CODESYS and Siemens PLCs respectively. The protocol allows the operators to know the exact time that an event occurred.

Examples of application packages added to UNICORE are: the SURVEY (Control system for the magnets alignment), the QPS (Quench Protection System for superconducting magnets), the PIC (Powering Interlock Controller, for powering permissions to LHC magnets) and the CIET (Cryogenics Instrumentation Expert Tool) packages. For the supervision UNICORE and the specific application package are used. For the control

layer Front-End Computers (FECs) are used while the development of the application is based on another framework called FESA (Front-End Software Architecture). Also, PLCs not based in CPC are used. For the field layer WorldFIP fieldbus is used for the connection to the equipment.

3.1.3 The CPC component

CPC package is a basic package integrated to UNICORE. The package is used both in supervision and control layers in order to develop a continuous process control application. It includes a library of objects and certain structure rules which are used for the PLC programming. Also automated tools based on this principle are offered in order to reduce the programming effort. Specific device objects can be added if CPC objects don't cover all of the application's needs. Examples of CPC applications at CERN are, the LHC Cryogenics Control system, the Gas system of the LHC experiments, Cooling and Ventilation systems, Vacuum systems and Interlock and Collimator systems.

UNICOS-CPC concept has been implemented for specific industrial suppliers which are officially supported at CERN. For the supervision layer the WinCC OA SCADA system from Siemens and local touchpanels from Siemens and Schneider are supported. For the control layer CPC is implemented for Schneider's Premium, Quantum and M340 for Unity development software and Siemens' S7 300 and S7 400H for Step 7 Pro development software. The supported fieldbuses used with the above controllers are the: MODBUS, PROFIBUS-DP, PROFUBUS-PA, PROFINet and CANOpen. Currently, under this project, new PLCs, fieldbuses and development environments based on CODESYS V3 platform have been integrated to UNICOS framework. These are, the Embedded PC (EPC) CX-5020 and the Industrial PC (IPC) C6920 from Beckhoff for TwinCAT development software and the M258 PAC (Programmable Automation Controller) from Schneider for SoMachine development software. Also, the EtherCAT fieldbus was used for the implementation with the Beckhoff IPC.

For every object in the PLC program there is a corresponding, linked object in the SCADA side. UNICOS communication protocol, TSPP is responsible for the communication and the synchronization between the PLC objects and their respective SCADA objects. In the PLC side the object is represented by a function block, while in the SCADA side by a library of code, which provides an interface widget and an interactive faceplate. A paradigm of a CPC object (Analog) representation in both layers is shown in figure 7.

A software tool, called UAB, has an integrated CPC Component, that is responsible for the generation of the files, which are imported to the PLC software and the SCADA. These files contain the PLC program and

the SCADA database respectively. The generated PLC program is divided in two categories: *the Instances*, where the instantiation of the CPC objects of the application is done and *the Logic*, where the programming logic for the Field and (Process) Control objects is implemented.

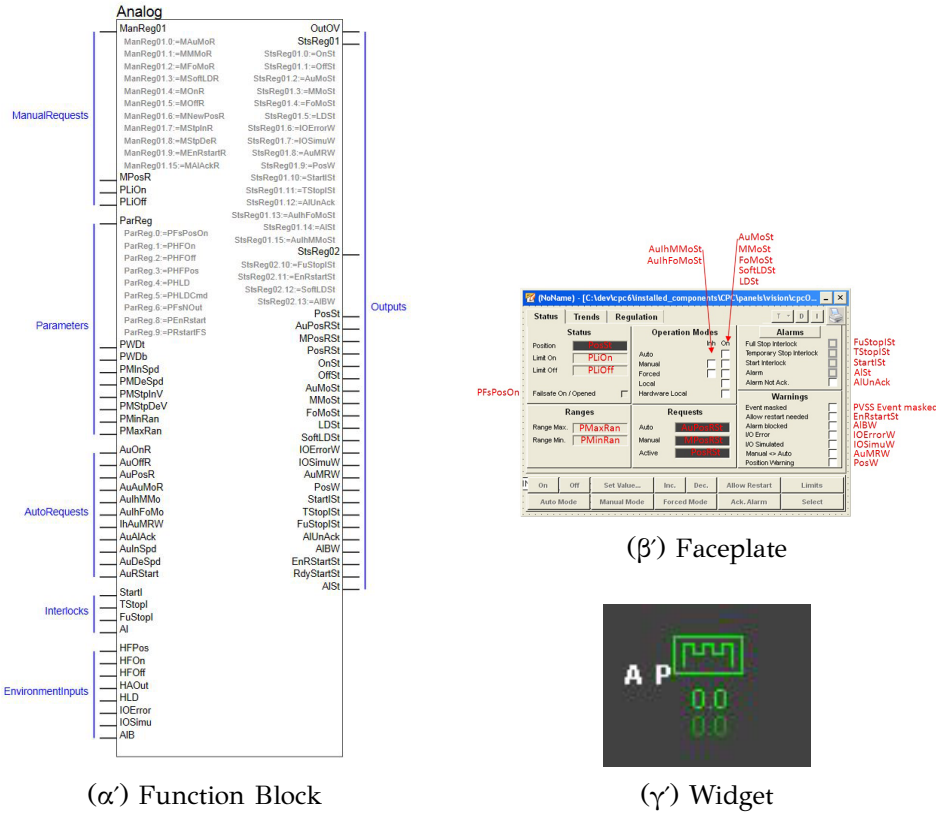


Figure 7: Analog Object

3.1.3.1 CPC Objects The full list of CPC objects(version 6) available for building a continuous process application are presented below. They are divided in four sub-categories according to their process functionality:

- *I/O Objects*: They provide the link between the inputs/outputs of the PLC and the control application. Inputs/outputs can be sensors, actuators, field buses or even internal memory(for test purposes).

Analog Input(AI) The object is used to connect the analog sensor signals to their associated Field object in the control system. The Analog Input object converts the process signal value received from the sensor to an engineering value to be used by the control logic. e.g. a voltage value from a pressure sensor converted to a pressure value. The signal is an integer.

Analog Input Real(AIR) It has the same functionality as the Analog Input with only difference that the signal is a real number.

Analog Output(AO) The Analog Output is used to connect the Field Objects with the associated analog actuators e.g. a pump. The object converts the engineering value coming from the logic to an output value for the actuator, e.g. the percentage of the voltage supply for a motor converted to the supply voltage depending on the motor's nominal voltage.

Analog Output Real(AOR) It has the same functionality as the Analog Output with only difference that the signal is a real number.

Digital Input(DI) The Digital Input object is used to connect the digital sensor signals to their associated Field object in the control system, e.g. a digital level sensor of a tank or a feedback from a device position(on/off).

Digital Output(DO) The Digital Output is used to connect the Field Objects with the associated digital actuators e.g. with an On-Off valve.

- *Interface Objects:* These objects concern the parametrization and status between the SCADA Interface and the PLC.

Analog/Word Status(AS/WS) These objects are the simplest way to represent the transfer of data from the PLC to the SCADA with the UNICOS standard. The input signal is directly mapped into the output memory. The Word Status has more functionality on the SCADA side according to the way the bits of the WORD are read.(e.g. stepper positions)

Analog/Digital/Word Parameter Accordingly these objects represent the transfer of data from the SCADA to the PLC with the UNICOS standard and they are used to change PLC parameters (e.g. alarm ranges).

- *Field Objects:* They represent the physical field equipments (e.g.: valves, motors, ...). The field objects are always connected to I/O objects.

Local The object is a representation of an equipment which is handled manually on the process field and gives its position feedback as an input to the PLC e.g. hand valves, manual pumps.

OnOff OnOff object corresponds to an actuator which is driven by a digital signal, like on-off valves, motors etc.

Analog On the contrary Analog corresponds to equipment driven by an analog signal, as analog valves, heaters etc.

Anadig The object is used for driving actuators with PWM. Two digital outputs can be connected to the object, one positive and one negative. Two options are available, (1) *Classic unipolar PWM* using one single output (positive output). (2) *Bipolar PWM* with a closed loop using the two outputs .

AnaDO This object represents process equipment driven by a digital signal (generally for the powering) and by an analog signal for the positioning. It combines Analog and OnOff objects hybrid behavior and it can be found in most of the industrial pumps in fields like vacuum, cooling and HVAC.

Mass Flow Controller(MFC) MFC object represents a Mass Flow Controller device. The object is mainly used in gas systems. A different calibrated curve is selected according to the gas used. Its particularity is that it is connected directly with the signals of the hardware without using an I/O object before. The reason is that the maximum flow range for each different gas it handles is not fixed and so an analog input or output object can't be used.

- *Control Objects*: These objects perform process logic control actions, alarms and interlocks, and feedback control. They are always acting on Field Objects.

Controller(PID) The object is based on a standard PID function and performs a regulation control loop. The controller has 3 working states available for all operation modes: (a) Regulation: The output signal is driven by the PID function. (b) Output Positioning: The control loop is opened and the output signal is assigned to a desired value (Auto or Manual position request). (c) Tracking: The output signal is set equal to the position request.

Process Control Object(PCO) The object is a processing unit which controls field objects or other PCOs. PCOs consist of logic sections which will be described later.

Digital Alarm(DA) This object performs a check of a Boolean condition and produces an interlock.

Analog Alarm(AA) Based on the position of an analog signal between 4 trigger levels the object can produce a warning on the value(L,H) or an interlock alarm(LL,HH). Warning is an indication (e.g. visual information on SCADA) of a potential problem. In this case, there is no action on the process. Interlock is an asynchronous condition allowing an actuator or a unit to stop or preventing from starting for security reasons. The possible

interlocks are: (1) *Full stop* interlock: Set the object “Off” (all dependent objects are set to their fail-safe position) and wait manual acknowledge before restarting. (2) *Temporary Stop* Interlock: Set the object “Off” (all dependent objects are set to their fail-safe position) and restart automatically when the interlock disappears. (3) *Start* Interlock: Prevent the object from starting (all dependent objects stay in their fail-safe position).

Common functionality to most of the objects(except interface objects) is the implementation of 4 different operation modes (Auto, Manual, Forced, Local), which can be controlled by the SCADA operator.

- Auto Mode: In this mode the object executes the order given by another object which is higher in the hierarchy(parent object).
- Manual Mode: In manual mode the execution order is given by the SCADA operator e.g. to open a valve. The object can return to Auto Mode by auto requests.
- Forced Mode: The order is coming again from the operator but in this case the object can not return to Auto Mode by auto requests(only from manual ”auto mode” requests).
- Local Mode : It has the same functionality as the Manual Mode but the orders come from a touchpanel.

Some objects offer all the four modes functionality and others only some of them. In figure 8 you can see the hierarchy of a CPC application architecture from the PLC view, based on some of the objects described above.

A detailed view of all the CPC objects and their associated input/output pins can be found in the Appendix A’.

3.1.3.2 TSPP/Modbus communication The communication between the PLCs and the WinCC OA SCADA follows the UNICOS framework rules and is implemented by the Modbus/TCP and S7 drivers of WinCC OA for Schneider/CODESYS and Siemens PLCs respectively. TSPP is an extension of Modbus/TCP and S7 protocols, which is defined by UNICOS framework. Communication is only possible if necessary programming is made in the PLC, commonly known as middleware. The implementation is different for each PLC platform, but the main principles are common. The TSPP/S7 and TSPP/Modbus protocol differences will not be mentioned here. TSPP/Modbus will be mainly described, since only Modbus communication was used for CODESYS integration to UNICOS.

Each CPC object has some dedicated output variables for its Binary and/or Analog Status and some dedicated input variables for the Binary

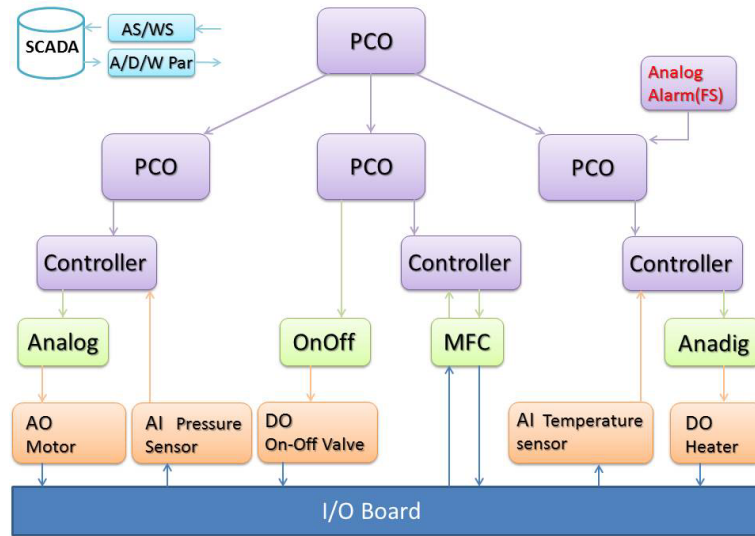


Figure 8: CPC Objects Architecture: Control(in purple), Field(in green), I/O(in orange) and Interface(in blue) Objects

and/or Analog Requests from SCADA. Binary/Analog Statuses are transferred from the PLC to SCADA, while Requests refer to the orders that are sent from the SCADA operators to the PLC. A change in the Binary status between two consecutive PLC cycles creates an Event. Events are also transferred to SCADA and are stored to a specific archive memory.

The Modbus/TCP driver can be used for Modbus/TCP or UNICOS TSPP protocol at the same time. The driver decides whether to use a UNICOS frame or a Modbus frame for the slave by observing the reference number of the frame. The reference number for a UNICOS TSPP frame is 0xFFFF. The Analog/Binary statuses and the Events are 16-bit WORDS that are sent in tables by using TSPP frames (different for TSPP/Modbus and TSPP/S7). The maximum size of a TSPP frame is 253 bytes, but is limited by Modbus/TCP to 200 bytes (100 WORDS of 16 bits). The Writing Multiple Registers Modbus function(Function code 16) is used for writing the data. The Request frame that SCADA sends to the PLC is a standard Modbus frame and consists of one or more 16 bit words.

In the PLC the Binary and Analog data are put into tables. A table contains 96 16-bit WORDS of Binary or Analog statuses. If one element in the table changes the whole table is sent. The changed tables are sent every cycle with a rate that is defined by the user. This is known as Push protocol. The table which is finally sent is an 100 WORD table. 96 16-bit WORDS are used for the Binary or Analog data and four WORDS are

reserved for the Timestamp (3 WORDS) and the Offset address⁸. When a Binary status changes, it causes the creation of an Event. The Event table contains maximum 19 Events. An Event contains the value of the Binary status before and after the change, its Offset address and the Time that this change occurred. The table is sent either when it is full or every 10 seconds. Although the data in the Events correspond to the Binary status data, different Timestamps must be used because the time that the Event data table is sent may be older than the regular sent Binary status table.

The above logic is implemented by Function Blocks dedicated for the Modbus/TSPP communication, which are instantiated inside the PLC program. These FBs are not considered as CPC objects but they are part of the Baseline library (figure 9).

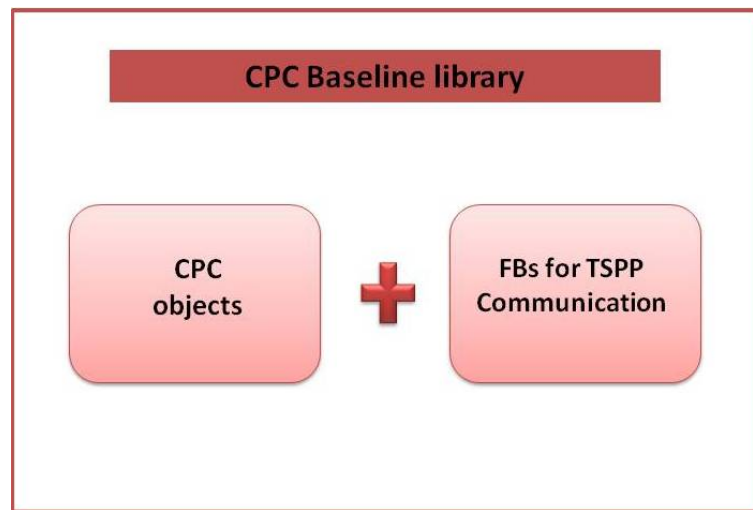


Figure 9: Baseline library

The WinCC OA driver must be a master and a slave at the same time for the UNICOS protocol: The Binary/Analog statuses and the Events are only sent from the PLC, no polling is required. In this case the driver has to be the Modbus slave (TCP server). The driver sends the SCADA Requests as a Modbus master (TCP client). The data flow for the UNICOS TSPP protocol is shown in the following figure. As it can be seen only Write Requests are used.

3.2 The UAB tool

The need for decreasing the development and commissioning time and producing well structured PLC code for large applications has lead to a software tool for the generation of the PLC code and the supervision

⁸It is the mapping address of the Binary/Analog Status variable.

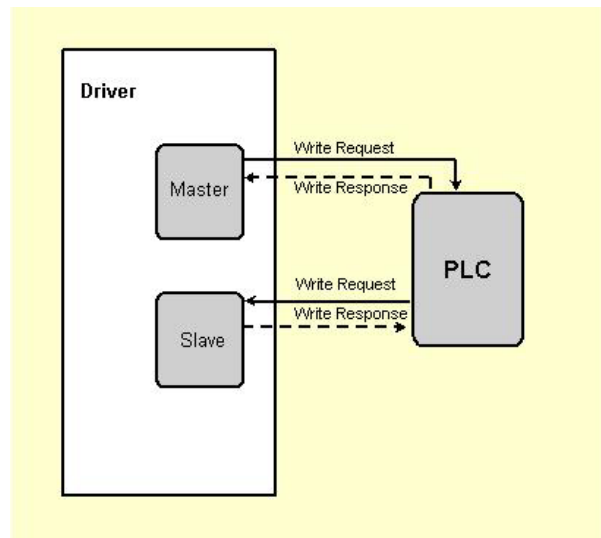


Figure 10: MODBUS TSPP Communication

database. The first code generation tool for UNICOS-CPC control systems was based on Microsoft Access. It had dedicated tools for both Schneider and Siemens platforms. The tool limits for bigger projects with several thousands of objects triggered the production of a new generation tool based on the latest software engineering concepts [16]. The migration to CPC version 6 introduced a new generation software, called UAB (UNICOS Application Builder). UAB is based on the software factory concept which allows focusing more on the expected result rather than on the means to produce this result. UAB is not limited to UNICOS or even code generation, and its architecture can be adapted to many domains. [6]

Figure 11 represents the architecture of the UAB tool. In this figure the TCT (Type Creation Tool), the CIET (Cryogenics Instrumentation Expert Tool) and the CPC (Continuous Process Control) Components can be seen. These are some of the UAB Components. The CPC component has three main parts. These are: the Core, the Plug-ins and the Resource package. These software components have different lifecycles and can be managed and released independently. The inputs and outputs of the UAB tool are also presented in this figure. More specifically for CPC component are:

The inputs (from the user):

- The Specifications file (Spec file): It is an xls/xml file which includes a declaration of all the objects needed for the control application and their specific attributes. The first template of the spec file is generated by the TCT based on the definitions of the different UNICOS device types and contains the structure that the user has to fill in. The user should fill in the spec with the objects and their attributes according

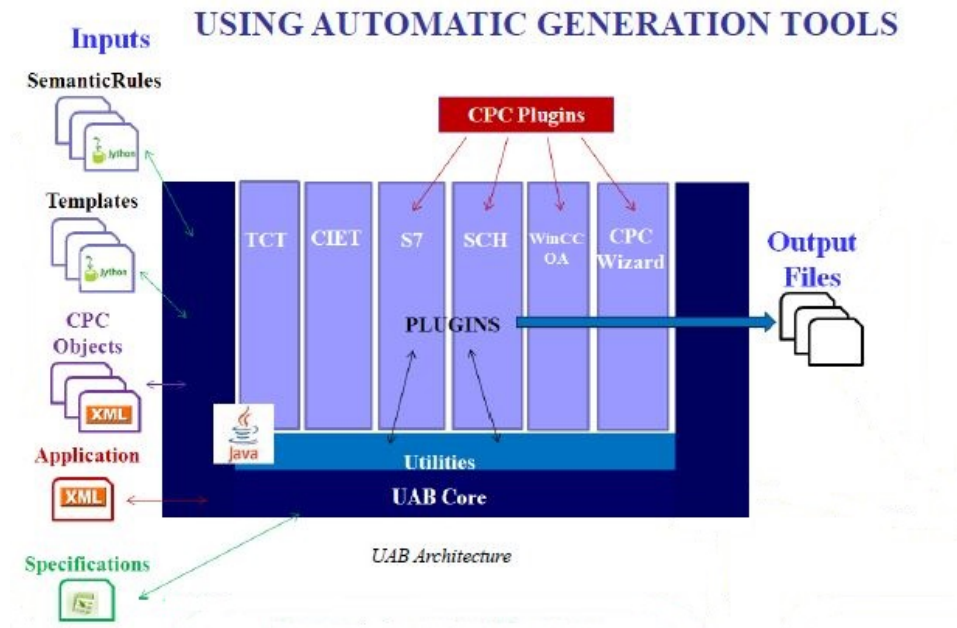


Figure 11: The UAB tool

to the functional analysis of the control application.

- The Logic User Templates: They are *Jython* scripts which contain the PLC code for the specific logic of the application. The templates are created by the user and are complementary to the templates that come with the Resource Package.
- The CPC Wizard parameters: The user has to fill specific parameters for the application in the UAB Wizard GUI such as the PLC Name, parameters for the communication with the SCADA (e.g. IP of the server), and parameters concerning the mapping of variables in both sides.

The outputs:

- The files for the PLC application, which are the outputs of the Instance and Logic plug-ins. These files contain the PLC program for the control application and are ready for importation to the PLC development software (e.g. Step 7 for Siemens).
- The supervision database, which is the output of the WinCC OA plug-in and can be imported to the WinCC OA application. Optionally, if a touchpanel is used for local control, the touchpanel generator can be used to generate a file for the development software

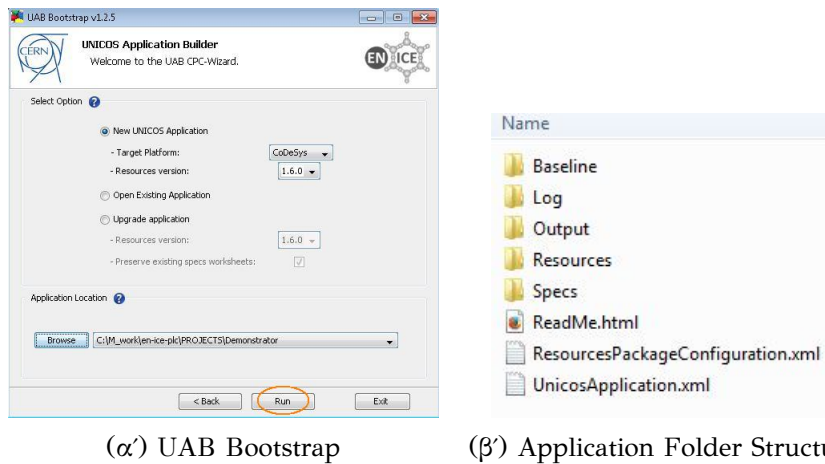
of the touchpanel with all the UNICOS variables that are exchanged with the PLC.

A tool called *UAB Bootstrap* is managing the different software versions of the UAB components and their Resource packages. It checks for updates and offers to download new components. To achieve this, the Bootstrap uses a repository manager (Sonatype Nexus) where the different versions of the Components, the Core and the Resource Package are released. The bootstrap guides the user during the installation, update and execution of the UAB components. For creating a CPC application the CPC Wizard option should be selected as shown in figure 12.



Figure 12: The UAB Bootstrap

In the next panel of the Bootstrap the user can select the development platform (Siemens, Schneider or CoDeSys) and specify the UAB application folder for the creation of the new application, as shown in figure 13α'. By pressing the *Run* button, the folder structure of the application is automatically created. The application folder has the following contents: (1) the Baseline folder with all the UNICOS baselines for control and supervision, (2) the Log folder which will contain the log files that are created during the generation process with information and errors that might occur, (3) the Output folder which will contain the output files after the generation, (4) the Resources folder with the contents of the Resource



(α') UAB Bootstrap

(β') Application Folder Structure

Figure 13: Creation of a UAB Application

Package, (5) the Specs folder with an empty Specifications file ready to be filled by the user, (6) the ResourcesPackageConfiguration.xml file which defines the list of objects that exist in the current Resource Package and (7) the UnicosApplication.xml file. This last file is built when the user creates the UAB application. Its contents come from the Core, the plug-in configuration parameters, the PLC parameters files for each platform and the ResourcesPackageConfiguration.xml. In the next steps of the Wizard the file will be written with parameters that the user will specify. The application's folder structure can be seen in figure 13β'.

The CPC Wizard is a plug-in of the CPC component, dedicated to provide the user with a friendly GUI, which will guide him through the generation process. In the first panel of the Wizard the Spec file should be selected and a name to the application should be given. Then, in the next panel the predefined PLC parameters are loaded from the the UnicosApplication.xml and the user can edit them through the graphical interface of the Wizard. These parameters have been described above as *the Wizard parameters*, which are overwriting the predefined parameters of the UnicosApplication.xml. The following panel includes a navigation menu for plug-in generators. For each generator the user has to select the objects for generation (usually all) and if other complementary files are going to be generated. Then by pressing the *Generate* button the output file(s) are generated and placed in the *Output* folder. The sequence of the panels that the user has to follow in the CPC Wizard and were described above, are presented in figure 14. In this figure the CPC Wizard panels, in case of a Schneider application, is shown as an example. The same concept applies for the other two PLC platforms.

The UAB Core, the Bootstrap and the Plug-ins are developed in *Java*. The plug-in configuration parameters use XML language and the tech-

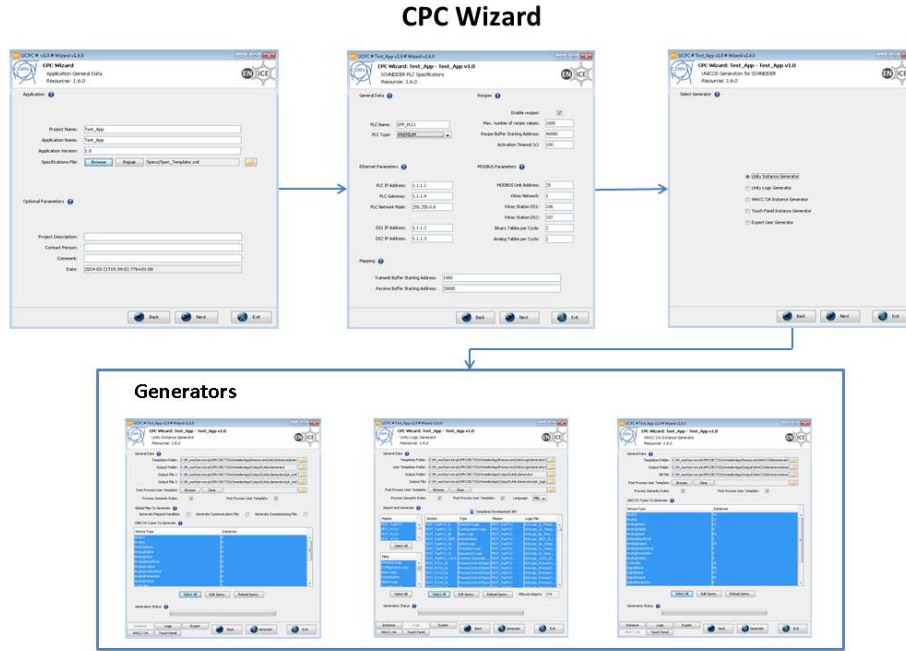


Figure 14: CPC Wizard panels

nologies used to process these files are JAXB (Java XML Binding) and JXPath (XML Path Language for Java). *Apache Maven* is the technology used to build and manage the different software modules. The templates are developed in *Jython* (Python for Java) which combines Python with Java functionality that is useful for the interaction with the plug-in. The CPC object definitions (DeviceTypeDefinitions) are also XML files, which comfort with an XML scheme called *UNICOS Metamodel* [5].

3.2.1 The Core

The UAB Core is the generic part of this software. It discovers and calls dynamically the different plug-ins and provides the services that are common to all the plug-ins, e.g. the *User Report (Log)*. It also loads the UAB project data and provides them to the plug-ins. Moreover, the semantic check rules that are defined in Jython Templates in the Resource Package are executed in the Core.

3.2.2 The Plug-ins

The CPC plug-ins generate the output files for the control and supervision layers based on the inputs of the UAB tool. They have a common base in the Core but each one is platform dependent (e.g. Siemens, Schneider).

The plug-ins load the parameters from the `UnicosApplication.xml` file, define the methods that are called in the `Jython` Templates and generate the output files. The CPC plug-ins are the following:

Siemens Instance and Logic Generator for PLC code.

Schneider Instance and Logic Generator for PLC code.

CoDeSys Instance and Logic Generator for PLC code.

WinCC OA Instance Generator Generator of the WinCC OA database.

Touchpanel Generator Generator of the database files for three different touchpanels :Magelis Vijeo, WinCC Flexible TIA Portal and WinCC Flexible 2008.

Expert User Generator Generator for files based on user scripts for purposes that are not covered by the previous plug-ins.

CPC Wizard As mentioned above this plug-in provides a GUI for guiding the user through the generation.

3.2.3 The Resource Package

The Resource Package includes all the required input files for the UAB tool that are not provided and don't have to be modified by the user. It is developed and released by the PLC section and has the following contents:

- **Templates:** They are *Jython* scripts for the: (1) three different PLC platforms, (2) SCADA WinCC OA, (3) Touchpanels, (4) IO Commissioning, (5) Expert User Generator, (6) Reverse Engineering, (7) Upgrade and (8) Shared templates. The first type of templates is divided into two other categories, the Instance and Logic Templates in correspondence to the plug-ins.
- **PLC Parameters:** They are XML files defining specific parameters for each of the three PLC platforms. These are then used to compose the `UnicosApplication.xml`.
- **Device Type Definitions:** They are XML files with the definition of the CPC objects. They are also generated by the TCT tool according to UNICOS model(xml) and validated with the *UNICOS Metamodel* scheme.
- **Semantic Check Rules:** They are *Jython* templates used for checking the spec file according to the Device Type Definitions.

- **Baseline:** Baseline is a dependance of the Resource Package handled by *Apach Maven*. The dependent files are downloaded from Nexus repository to the UAB application folder and include the files required for creating the control application. Among them are: PLC applications which contain the CPC library and maybe a hardware configuration (one for each supplier), also UNICOS-CPC library components for the WinCC OA and touchpanel applications containing the CPC library.

4 CODESYS plug-ins and Resource Package

4.1 Modification of the UAB CODESYS plug-ins

The main goal of the Plug-ins is to be able to automatic generate the code for the IEC 61131-3 PLC program, in the form of PLCOpen XML files, for the CODESYS-based platforms SoMachine and TwinCAT, according to UNICOS-CPC framework. This implied changes to the CODESYS Plug-ins and creation of new Templates that give the right output, according to the CPC logic. The development environment used for the plug-ins was *Eclipse IDE*. The first release of the CODESYS plug-ins with PLCOpen XML output was for the Resource Package 1.5.0.

Figure 15 shows the Plug-ins needed for creating a full CPC-CODESYS application. These are the CODESYS Instance and Logic plug-ins for the PLC program and the WinCC OA plug-in for the supervision. The Touch-panel plug-in could be optionally used. In the picture are also shown: (1) the inputs of the tool, which are the *Jython* Templates and the Specifications file, (2) the XML files that interact with UAB, which are the Device Type Definitions and the UnicosApplication.xml, and (3) the outputs, which are the generated PLCOpen XML files for the SoMachine and TwinCAT development software and the WinCC OA database.

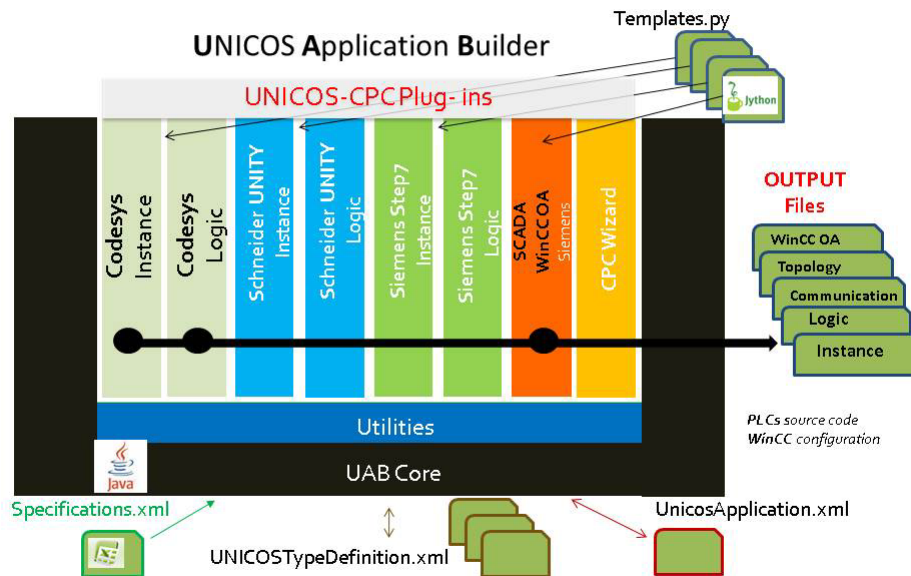


Figure 15: Plug-ins used for creating a CPC CODESYS application [16]

The output files of the CODESYS plug-ins are:

- From the Instance plug-in:

Instances file It contains all the instantiations of the CPC objects that are defined in the Specifications file and a GVL (Global Variables List) with all the variables that need to be accessed by all the programs of the application. Each CPC object type has a dedicated POU (Program Organization Unit) which contains all of the object's instances e.g. the Analog POU is a program which successively calls all the CPC_Analog Function Block's instances, defining its input and output variables. Exception to this, are the Controller and PCO objects, which have a different POU per instantiation (call). This is an enhancement from the previous implementation for reasons of right order of execution in the Task Configuration. Each instance is a global variable of a CPC Object Data Type. This, enables to access the input and output pins of an instance just by using a dot next to its name e.g. PCO_Instance.RunOSt. That is a main difference from the previous implementation, which was based on declaration of global variables, that were assigned to each pin, and reduces a lot the GVL's length.

Communication file This file contains the instantiations of the TSPP Communication FBs with all the necessary parameters (e.g. IP address...) required from the PLC/IPC to establish the communication with the SCADA. These parameters are given by the user in the UAB Wizard. The spec file does not define any communication objects.

- From the Logic plug-in:

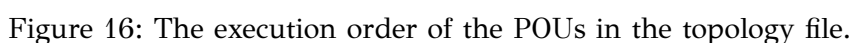
Logic file It contains the logic for the Field, Controller and PCO objects and a GVL with all the variables that need to be accessed from all the programs of the application. The logic is divided in different POUs, which define a logic section according to UNICOS CPC framework. The PCO has eight different logic sections and the Field objects and the Controller have two logic sections. Some of these sections can be modified by the user by creating the *Logic User Templates* and defining them in the spec file. A more detailed description of the logic sections will be given in paragraph 4.2.2.

Topology file The topology file defines the execution order of the programs of the application for each PLC cycle. There are two different implementations for the topology file, for SoMachine

and TwinCAT. The first one defines a program, which successively calls all the application's programs with the appropriate order. Then, this program has to be added in the Task Configuration of the PLC application. The rest Task Configuration parameters (e.g. cycle time) should be manually defined. In the other implementation the topology file defines an IEC Task Configuration, meaning that except of the execution order it defines parameters such as the task's name and type (e.g. "cyclic"), the cycle time and its unit, the priority and the enable of Watchdog. TwinCAT supports the importation of Task Configuration, as it is defined in the PLCOpen XML scheme but SoMachine does not. Consequently, for SoMachine, Task execution order had to be defined with the first way. To avoid differences the same solution could have been applied for TwinCAT. Nevertheless, since PLCOpen XML is a standard format, it is expected that SoMachine will gradually adopt this feature of the scheme. Finally, both solutions were implemented, expecting that in the future only the second, standard one will be used for both systems. The execution order of the programs in a CPC PLC application which was implemented in the topology file is shown in figure 16.

For the supervision with WinCC OA SCADA, a plug-in is shared between CODESYS, Siemens and Schneider. CODESYS and Schneider PLCs use Modbus protocol for their communication with WinCC OA and they also use the same memory mapping conventions for the variables that are exchanged between the PLC and the SCADA. For the local supervision with touchpanels a plug-in is also common for all the PLC platforms, supporting the Siemens and Schneider touchpanels. Only Schneider Magelis Vijeo touchpanels, with Modbus communication protocol, were used with the CODESYS compatible PLCs of Beckhoff and Schneider.

During the development of the UAB solution for CODESYS V3 the output files had to be validated for their consistency with the PLCOpen XML scheme, which is provided for free by the *PLCOpen* organization. For this reason, the *Altova XMLSpy* software was used. The UAB output file was opened in the editor and then the PLCOpen XML scheme *tc6_xml_v201.xsd* was assigned to it. The software gives the option to quickly validate the well-formedness of the XML and also the consistency with the scheme. In the output window, it gives a message of the place of the malformation, if it exists, or a message that the file is well-formed XML and valid according to the assigned scheme. All UAB output files were tested and are valid PLCOpen XML files.



The CODESYS instance plug-in is the part of the CPC Component of UAB tool, that is responsible for the generation of the Instance, the Communication and the IOCommissioning files. The plug-in, as part of the UAB, is also developed in Java programming language. The plug-in defines a Java Class called *CodesysCodeGenerator*. This class extends a class from the Core, called *AInstanceGeneration* and is extended by all the instance plug-ins of the three PLC platforms to offer common functionality. The *AInstanceGeneration* class extends the *AGenerationPlugin* class, which extends the *Aplugin* class. These later two classes, are extended by all the plug-ins providing common services, as for example, the log messages for the User Report Window and the execution of the Semantic Check Rules for the spec file. The relations with the classes of the Core, the class hierarchy and the purpose of each class can be seen in the figure 17.

Master Thesis

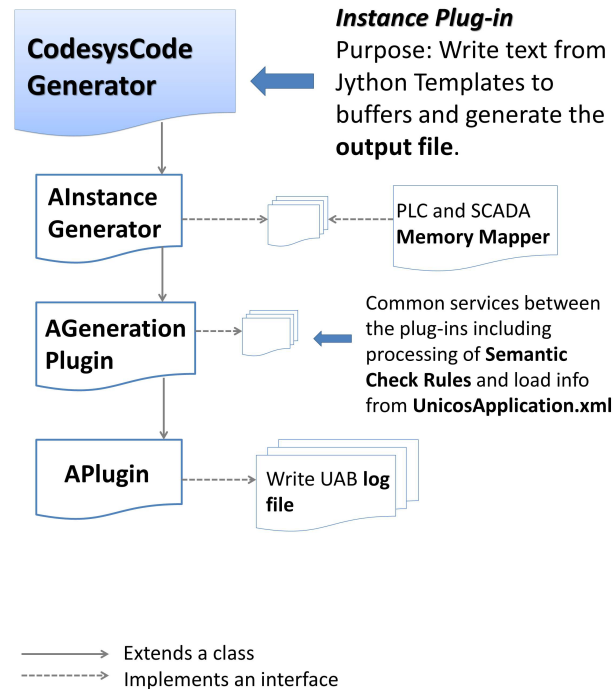


Figure 17: Java Classes for Instance generation

the GVL of the PLC program, and they have a dedicated Jython Template. They are used by SCADA operators in order to send a big number of parameters to the PLC simultaneously. Then, methods are successively called for generating the Instance file, the Communication file, the IOCommissioning file and executing the Post Process Templates. The IOCommissioning is an xls/xml file which is used during the commissioning phase of the application and is produced from a shared template for all the PLC platforms. The Post Process templates are user Jython scripts, which generate additional files based on the plug-in methods.

In the plug-in, String and Vector buffers are used for writing the output files. Some of the plug-in's methods use String buffers as arguments. These methods are called in the Jython Templates, for each object. Since the output is an XML file it has a header, a footer and other common elements. For example, the method responsible for the header of the file is called in every Instance Template, providing always the same text. In cases when one object is selected for generation, the header of the file is written with the call of this method once. In cases when many objects are selected for generation, the method is repeatedly called but only the text of the buffer

from the last call is finally written in the output file. The Vector buffers are used as method arguments in cases where the repetition of the call of the method, should add new text each time it is called.

The methods that are defined in the plug-in are called inside the Instance Jython Templates, as it was mentioned above. These methods are responsible for filling the buffers with the text that is specified in the Templates. For the generation of the instance file, the buffers are processed in the right order inside the plug-in, in order to give the right output. The output is written as an XML file, by using a method called *WriteXmlFile*, which is defined in the Core and is used by all the plug-ins. The same rules are followed for the generation of the Communication file. If no devices are selected in the Wizard the instance file will not be generated.

For the modification of the plug-in new methods and buffers had to be added. In this case changes had to be done in three different parts of the plug-in. First, new buffers of the appropriate type are declared. The new buffers are appended in the right order for the generation of the output file. Also new methods, using as arguments the new buffers, are added. These methods, are then called in the Templates. An example of how new buffers and methods were added in the plug-in, is given in figure 18.

Finally, in the XML configuration file of the plug-in the extension of the output files had to be changed to .xml instead of .xst and .export which was used for the CODESYS importation files. This file, is one of the files that are used for the creation of the UnicosApplication.xml file.

4.1.2 Logic Plug-in

The CODESYS logic plug-in is the part of the CPC Component of UAB tool, that is responsible for the generation of the Logic and the Topology files. Logic plug-in defines a Java Class called *CodesysLogicGenerator*. In the same way as the Instance Generator, the class has dependencies from classes of the Core, that provide common services.

When the user is pressing the *Generate* button in the UAB Wizard the *generate()* method of the plug-in is called. This method is responsible for generating the output files. First the PCODEclarationScript Template is processed. Contrary to the Instance plug-in, in the Logic plug-in only Vector buffers are used (although some of them are written only once). In the PCODEclarationScript.py the header and footer of the XML output file and the variable footer and header buffers are written. Since this template is always processed, the call of the methods in the Logic Templates don't have to be repeated, as in the Instance Templates. Then, the VersionNumbers Template is processed. This template creates variables, which contain the version of the resource package used for the generation and the ver-

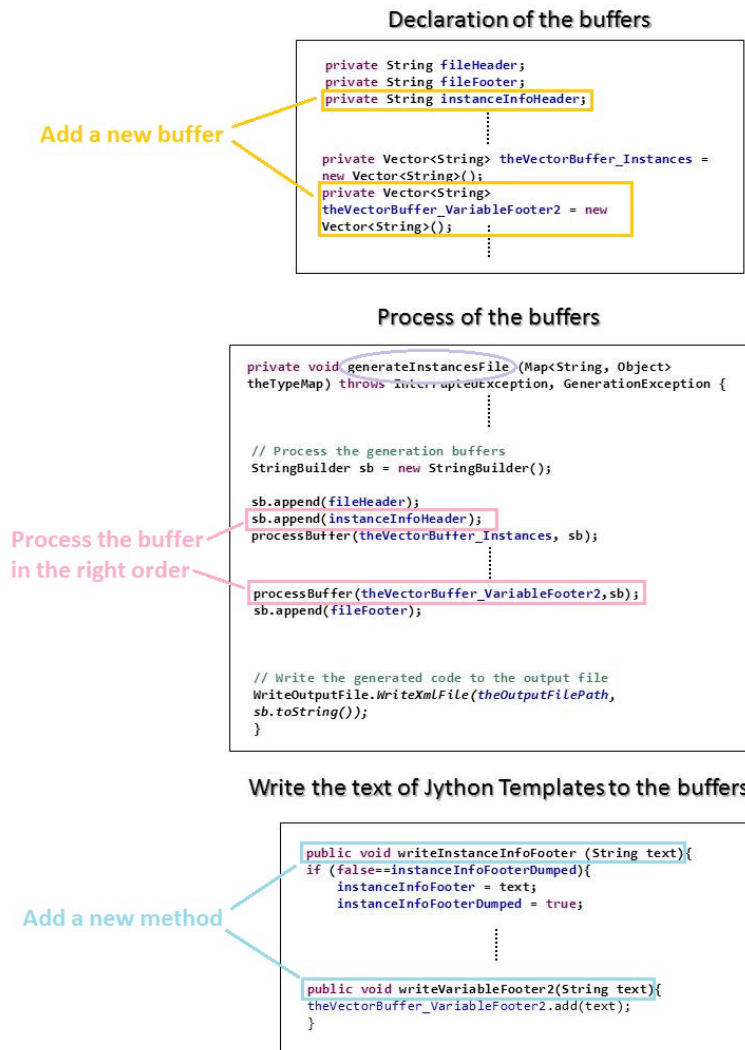


Figure 18: Modification of the Instance Plug-in

sion of the PLC application⁹. After, a method is called for the generation of the Topology file and then one for the processing of the dependency tree. The dependency tree processes each PCO Section (IL, CL, BL, ...) and for each dependent device, it processes the dependent sections. Comparing to the previous implementation the dependency tree is not processed, in order to have the ability to generate only one logic section e.g. in case of a change in the code of one User Template. Finally, a method is called for writing all the logic sections to the output file and the execution of the Post Process Templates is performed (if there are any defined by the

⁹In the Baseline they are written in mapped variables, which are sent to SCADA.

user).

The existing buffers and the methods that write to them, were adequate for the generation of the logic and topology file in PLCOpen XML. Changes were made only in the processing order of the buffers and in the Logic Templates. For the topology file due to the distinction mentioned in paragraph 4.1 two different Templates were created and some code was added in order to process the right Template, according to the CODESYS-based platform that is chosen in the Wizard, either TwinCAT or SoMachine. This information is extracted by the *UnicosApplication* xml attribute *PLCEnvironment*, which is taken by using the method *getPLCParameter*. Also an attribute parameter was added in the XML configuration file of the plug-in for the Topology Template. The part of the code added for the Topology file generation in the plug-in is displayed below.

```

/* Generates the topology file. */

private void generateTopologyFile() {
    if (!generateTopolFile)
        return;

    // Gets the absolute path of the topology template and
    // choose which template will be used according to the
    // PLCEnvironment
    String theTopologyTemplate = null;
    boolean showErrorMessage=true;
    String getPLCEnvironment =
        theXMLConfigMapper.getPLCParameter
        ("GeneralParameters:PLCEnvironment", showErrorMessage);
    theTopologyTemplate = templatesFolder +
        theXMLConfigMapper.getTechnicalParameter(getId() +
            ":TopologyTemplates:" + getPLCEnvironment +
            "Template");

```

Finally, in the XML configuration file of the plug-in the extension of the output files had to be changed to .xml instead of .xlm and .export, which was used for the CODESYS importation files. This file, is one of the files that are used for the creation of the *UnicosApplication.xml* file.

4.2 Creation of the CODESYS Templates

Since new methods were introduced in UAB, CPC logic in the Templates has evolved and the output XML text is very different from the previous implementation of CODESYS¹⁰, the Templates for the CODESYS plug-ins had to be re-created. The same UNICOS principles are used for

¹⁰An example of the two formats can be seen in figure A.1 in the Appendix.

the logic of all the three PLC platforms for the generation of the code but the implementation differs according to the particularities of each system. A first release of the new Templates was in Resource Package 1.5.0, followed by a second one after some completions in 1.6.0.

4.2.1 Instance Templates

The Instance Templates package for the Instance plug-in includes twenty-one Instance Templates, one for each object, one Template for the Recipe Buffers and one for the Communication. The general structure of an Instance Template is the following. Java classes and interfaces (from the Core and the plug-in) are imported first in the template. A class is defined for the specific object, for implementing the interfaces and using the methods of the classes. All the plug-in's methods mentioned above are called in the right order and also methods from the Core are called e.g. `writeInUABLog()`, which writes a message in the Log window of UAB. In the Template, the attributes from the Specifications (spec) file are loaded and are used in order to make calculations and compute values for the PLC code. This, is the implementation of the CPC logic, according to the UAB inputs. Also, the calculation of the addresses of the PLC variables is done by calling a method from the Core. The variables mapping is based on rules that can be seen in the Appendix B'. The structure of On-Off Template is presented as an example of the structure of an Instance Template in figure 19.

OnOff Template Structure

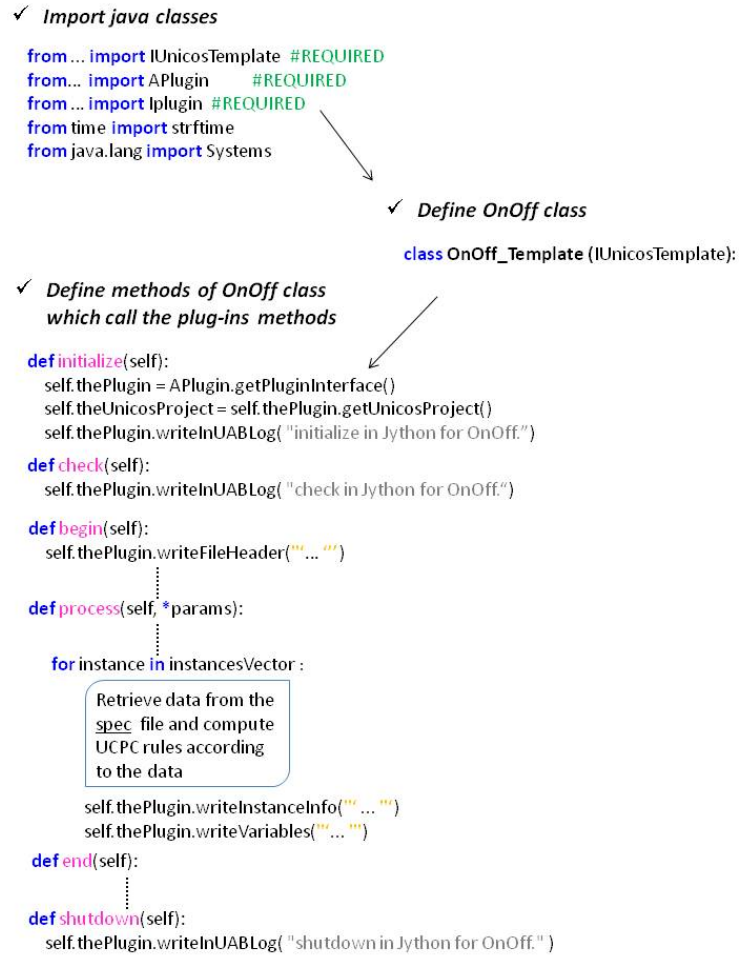


Figure 19: Instance Template Structure

The new methods that were added in the plug-in for this implementation are now called in the templates. The already existing methods are also called but the input text is now of the PLCOpen XML format. A representation of the procedure for the generation of the PLC code is shown in figure 20.

Also, two new Instance Templates were added since the previous implementation, one for the instantiation of the AnaDO and one of the MFC objects.

A main aim during the development of the Templates was to have shared Templates for both SoMachine and TwinCAT. The only difference

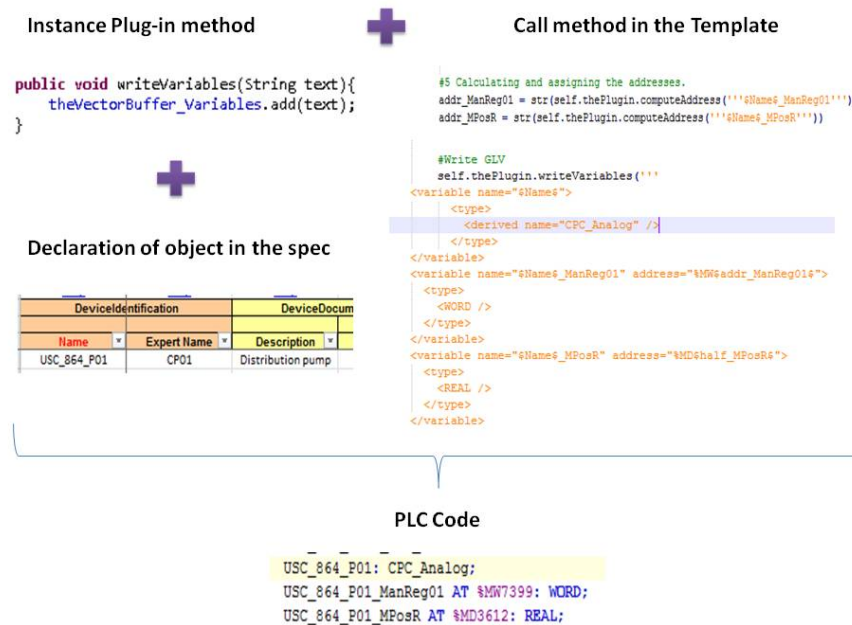


Figure 20: Creation of Instances file

between the two systems is the way the variables are mapped to the inputs and outputs of the PLC.

In SoMachine environment, the pins of every new I/O module that is added in the configuration have a predefined input or output address, according to IEC61131-3 [8] e.g. %IW6, %QW2 and this address can be accessed directly in the program.

In TwinCAT environment, a variable has to be declared to any input or output address, as it is defined in IEC61131-3 standard. After compilation, the mapped variable appears as an input or output variable, which can then be mapped to a channel of the hardware I/O module.

Due to this difference, input and output variables needed to be declared for the CPC Analog/Digital Input/Output objects. This was done in the corresponding Templates. Since this change introduces a small difference in the I/O object's Template, there was no reason for creation of a second Template for the same object, as it was done in the case of the Topology Template. As an example, the code that was added in the Analog Input Template can be seen below. An extra I/O variable is added or not based on the PLC Environment (TwinCAT/SoMachine), that is specified by the user in the UAB Wizard. The same principle was followed for the other three I/O objects.

```

config = self.thePlugin.getXMLConfig()
plcType = config.getPLCDeclarations().get(0).getPLCType().getValue()
environment = config.getPLCParameter("GeneralParameters:PLCEnvironment")

for instance in instancesVector :
    instanceCounter = str(int(instanceCounter) + 1)

    S_HFPos = "%IW" + S_Slot

    if S_HFPos[0:2].lower() == "%i" and environment == "TwinCAT":
        self.thePlugin.writeVariables('''
            <variable name="$Name$_AI" address="$S_HFPos$">
                <type>
                    <WORD />
                </type>
            </variable>
        ''')

```

4.2.2 Logic Templates

Below are listed all the logic sections defined by CPC. PCO has all the eight sections, while Field Objects and Controller have only DL and BL sections. The first two sections of the list below, are written automatically by the standard Templates of the generator. The rest, have also the possibility to use the standard Templates but can also use Templates defined by the user according to the application's logic (it is recommended).

1. BL: Basic Logic, contains the basic logic of the object. It is not necessary to fill it.
2. CDOL: Common Dependent Object Logic calculates the auto requests of all dependent objects and the alarm acknowledgment. It is not necessary to fill it.
3. GL: Global Logic, allows defining the global logic of the PCO.
4. IL: Interlock Logic, contains the alarm instantiation and calculation of the Interlocks of the PCO (Start, Temporal, Stop Interlock).
5. SL: Sequencer Logic, sequential behavior of the PCO (generally in SFC language).
6. TL: Transition Logic, contains all the calculations of the transitions between the steps in the SL.
7. CL: Configuration Logic, is used for the calculation of the On and Off status conditions of the PCO and is used mainly for animation purposes. In addition, it computes the controlled stop finished conditions.

8. DL: Dependent Logic contains the behavior of an object, linked with the PCO, according to a status (e.g. stepper states). The objects can be: PCO, Controller, On/Off, Analog, AnaDig, and AnaDO. All these objects have a Dependent Logic section associated to their PCO master. Note that a PCO can have a PCO master.

The Logic Templates package is divided in four categories, as it can be seen in figure 21. Common, ST, User Specific and Topology. The three first are used for the creation of the Logic file and the last one for the Topology file.

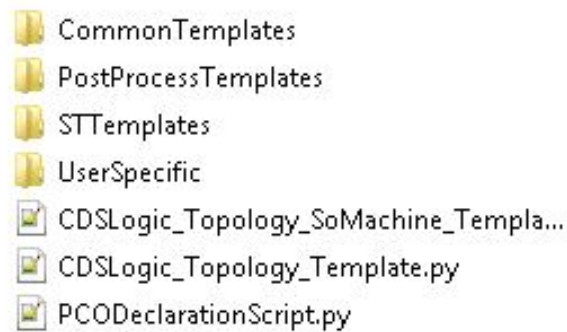


Figure 21: Logic Templates categories

Common templates are responsible for the BL, CDOL, SL logic sections of the PCO and for the Default Alarms. Default Alarms provide common methods to handle the Analog and Digital alarms of the application. These methods are called in the IL section of the PCO and in the DL section of the Field objects Logic templates and write the PLC code concerning the alarms. These methods can be also called in the User Specific Templates.

The ST Templates refer to the CL, GL, IL, TL and DL sections of the PCO, the Controller and the Field objects DL and BL sections. Both Common and ST templates have a Main part, which is always called, from which they call either a Standard template or a User Specific Template. If nothing is defined in the spec the Standard Template is called. If the name of a User Template is defined in the spec file for a specific object, under the attribute *CustomLogicSections*, then this template is called instead for the Standard Template. The template is written by the user and should be placed under the *User Specific* folder of the UAB application. The same applies for the SL Template of the Common Templates.

As mentioned above there are also two Topology Templates for So-Machine and TwinCAT. In each generation only one template is called according to the PLC environment. The two Templates generate two different topology files, as the one defines a program with the call execution

order and the other defines the whole Task Configuration of the PLC program.

New Logic Templates were also added for the AnaDO and the MFC objects, since the previous implementation.

For the moment PLCOpen XML importation for SFC language is not supported by SoMachine. So the SL (Sequencer Logic) section can be successfully imported only in TwinCAT. When SL section is imported into SoMachine it will give the following message: "Implementation of SFC is currently not supported for <body> of <pou> element. The implementation part will be empty ST." and import an empty ST program. Since PLCOpen XML is a standard format and CODESYS since Version 3.5 SP1 Patch4 supports the PLCOpen XML scheme for the SFC language, it is expected that SoMachine will adopt soon this feature of the scheme.

The logic plug-in methods are called inside the Logic Templates in the same way as it was described for the Instance Templates and the text that is written to the plug-in's buffers is of PLCOpen XML format.

4.3 Creation of CODESYS Baseline

The Baseline mainly consists of a library of CPC Objects and a library of FBs for the TSPP communication with the WinCC OA Modbus driver, as they were described in paragraph 3.1.3. The goal also here was to make a common library for both systems. This led to a common library of all the CPC objects. Inevitable differences in the Modbus communication and the Time functions between TwinCAT and SoMachine platforms led to a separate communication library for each system.

In order to facilitate the user, the PLCOpen XML library that was created for the Baseline, is imported in both software. The provided Baseline by UAB, includes one SoMachine and one TwinCAT application, which contain the Baseline library.

The Baseline library can be divided in five categories: (1) the CPC objects, (2) the TSPP Communication (3) the Global Variables, (4) the Parameters and (5) the Recipes. The CPC objects and the TSPP communication are FBs, developed in ST language. The Global Variables are declarations of variables in GVLs (Global Variable Lists). The Parameters consist of Functions and FBs, that are used by the CPC objects and Structures, which are inputs of the CPC objects. The Recipes consist of a Program and two Structures.

The software versions of the Baseline applications are for SoMachine 3.0.14.5 and TwinCAT 3.1.4012.0. The Baseline of CODESYS first release, was done in the UAB Resource Package release 1.6.0.

4.3.1 CPC Objects library

The library of the CPC objects was created according to the CPC principles. Each Function Block contains the declaration part of the Inputs, Outputs and local variables and the ST code.

In order to implement the logic of each object, some functions or function blocks had to be used in the code. The aim was to use libraries that are not vendor specific and are supported by both SoMachine and Twin-CAT (mainly CODESYS libraries), so that we have a common library of CPC objects, which is easier to maintain and to exchange between the two systems. In cases that a common library function didn't exist, instead of using the vendor specific, a new function or FB was created to cover the needs of the object. Some functions that were present from the previous implementation had to be partially modified.

In the table 1 the list of the created Functions and FBs for CODESYS Baseline, can be seen, together with a description and the object in which they are used.

Library	Description	Use
FB_LAG_FILTER	It represents a first order delay. The output is following the input after a delay but there is also the option for a tracking mode.	Controller
FB_SAMPLE	It sets the output to true after a time interval which is given as input.	Controller
FB_MFC_TOTALIZER	It integrates the value of the input (typically a flow volume) over time, until an adjustable limit is reached (typically a volume).	MFC
FUNC_PWM1	It implements a Pulse Width Modulation.	AnalogDigital
FUNC_Scaling	It performs scaling of the input within a min and a max limit.	Field Objects Controller
OR_BOOL	It implements a Boolean OR between the inputs and gives one output.	Field Objects

Table 1: Functions and FBs used by CPC objects

4.3.2 Modbus TSPP Communication library

All UNICOS instances have to exchange a set of information between the SCADA and the PLC. This is achieved by a constant transfer of data buffers between the PLC and the SCADA, also known as middleware. As it was mentioned before the PLC is sending the data to SCADA by using Modbus function 16 for writing multiple registers and TSPP protocol for the content of the frame that is sent. Each supplier has its own Modbus function for this. In TwinCAT the function is called `FB_MBWriteRegs()` and in SoMachine `WRITE_VAR()`.

The PLC data buffers which are forwarded to SCADA are sorted in three groups.

1. The Binary Status buffer: It contains all the Boolean statuses grouped into 16 bit words (each bit corresponds to a different Boolean variable).
2. The Analog Status buffer: This buffer contains all the other data values (32 bit real) to be sent to the SCADA.
3. The Event buffer: It contains the changes in the Binary status between two consecutive PLC cycles (Events).

The SCADA data to be forwarded to the PLC are also divided in three categories.

1. The Binary request: It contains all binary requests by the SCADA operators. They are grouped into 16 bits word formats.
2. The Analog request: It contains all the analog information that are provided by the SCADA to the PLC. They are mainly real numbers coded into 2 16 bit words.
3. The Recipes: They contain parameters that need to be changed simultaneously from SCADA operators and they are handled separately in `CPC_Recipes`.

Each group of the above statuses and requests is assigned to a specific memory area in the PLC and in the SCADA.

To be authorized to send (push data) data to SCADA, the PLC has first to detect it. For that, the SCADA sends continuously at a three second rate a 32 bit word containing in alternation the value Zero or its own encoded IP number. The PLC will declare the connection valid if it receives a four consecutive alternation with the same pattern. Another check is also requested. It is based on the matching of the received IP with a locally referenced IP number provided by the communication file.

Accordingly, the SCADA must detect the aliveness of the PLC. For that, the PLC sends a counter that is incremented at the rate of 1 second. This counter is taken by the middleware as a Binary Status Register, and is written at the fifth element of the first binary table which is sent every 1 second to the SCADA.

Independently of the change of any data value in Binary or Analog buffers, it has to be guaranteed that regularly, all data information are send from the PLC to SCADA. Two options are available. 1.A default one, check that any block of data has been sent within a certain delay. If not this data block is requested to be sent again. 2.The user may request to have all buffers sent at a regular rate independently of the data change. (Continuous push).

A summarized view of the above is given in figure 22.

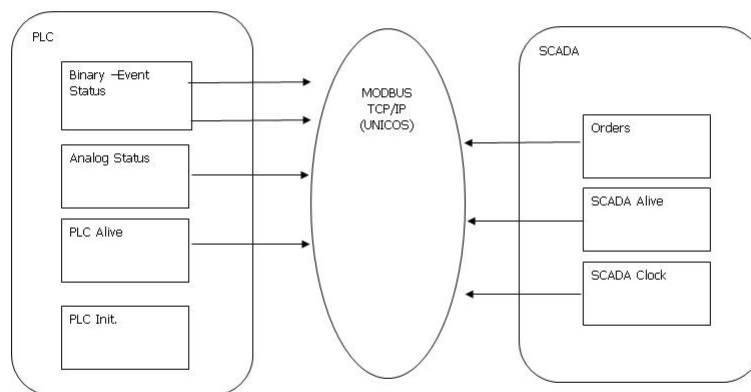


Figure 22: Middleware Communication

The communication for CODESYS consists of four function blocks:

- **CPC_CommSystem:** For getting PLC time, detect first cycle, calculate cycle period.
- **CPC_CommStartUp:** For alive signal from and to SCADA, continuous/safety push, calculation of UNICOS timers and setting the PLC time according to SCADA time.

- **CPC_CommData:** For Binary and Analog tables.
 1. **Binary data:** For the Binary data the first table the first five words are always reserved for the timestamp (UNICOSYear, UNICOSTimeUpper, UNICOSTimeLower), the offset address of the table and the counter which is sent to SCADA. Also two words are used for the Application and the Resource Package version and the rest is empty. This table is treated differently from the other binary tables and it is sent every one second to SCADA. The rest of the tables are checked and if even one word in the table is different from the previous cycle the table is scheduled to be sent. (This normally creates an event, but they are not handled here).
 2. **Analog data:** In Analog data there is not this particularity with the first table so we treat the data as the Binary data for all the tables. Here, a change in the data doesn't create an Event. For both cases if there is no change in data, there is a regular safety push.
- **CPC_CommEvent:** For Event tables. There is an array of 100 Event tables (CPC_CommEvtBufferStruct), which is a structure that contains the UNICOSYear (different from the Binary data UNICOSYear), an array of 19 events (CPC_CommEvtStruct) and a counter of the events that are stored in the events array. Each event (CPC_CommEvtStruct) contains UNICOSTimeUpper, UNICOSTimeLower, the Offset address, the new and the old value. Here, all Binary tables are checked and if even one value is different an event is created. The Event table is sent either when it is full or every 10 seconds.

six structures that are used by the above FB:

- CPC_CommBufferStruct
- CPC_CommEvtBufferStruct
- CPC_CommEvtStruct
- CPC_CommGlobalStatsStruct
- CPC_CommParamStruct
- CPC_CommStatsStruct

one global variable list:

- GVL_Comm

and one program that calls the above function blocks and is generated by UAB (Communication file):

- CPC_Com_DS

Although both CODESYS and Schneider should communicate with the same way with the driver in WinCC OA, their implementations in the PLC are different. The main difference is that while in Schneider there are two FBs, one for Binary statuses and Events and one for Analog statuses, in CODESYS there is one FB which deals with Analog and Binary statuses and one for the Events. The reason for this choice was that in small control systems maybe an Event functionality is not needed and could be easily disabled.

A Communication Baseline for SoMachine and TwinCAT systems had been developed for the two previous implementations of CODESYS. More features needed to be added to complete the baselines so that they function properly and according to the principles listed above and in paragraph 3.1.3.2. A description of the completions and the changes done in the communication baselines is shown below.

- Since Beckhoff IPCs can achieve cycle times of μs the unit of cycle period is now expressed at μs instead of ms for both systems.
- The time stamp of the PLC has to be synchronized with a unique clock. The time provided by the SCADA system is updated via its own mechanism. The reference time used is the UTC (Coordinated Universal Time) time and not the local time. The SCADA sends regularly its own time value to the PLC at the first five memory addresses and the PLC resynchronizes its local clock with the received time stamp. Then the PLC has to send its time to the SCADA and is checked for its accuracy with the SCADA time. Whenever SCADA sends a new timestamp the PLC time has to be reset. For SoMachine the Real-time clock (RTC time) of the PLC is set. For TwinCAT this feature had to be added. There are 5 timers in TwinCAT:
 1. BIOS Motherboard: RTC, battery-operated on the motherboard.
 2. CPU time: CPU counter from the hardware of the controller, not regulated, initialized with the RTC.
 3. Windows/NT time: Local system time of the Windows operating system (NT), initialized by the RTC.
 4. TwinCAT/TC time: Continuous TwinCAT clock, initialized by Windows. It is the local time in consideration of the adjusted time zone, usually UTC.

5. Distributed clocks/DC time: The system returns the start time of the current task cycle, initialized by TwinCAT. It is the local time in consideration of the adjusted time zone, usually UTC.

Since TwinCAT doesn't allow to set the RTC time, the closest choice was to set the Windows time. Consequently, also getting the PLC time is implemented differently in the two systems.

- For the measurement of the cycle period an average cycle time is calculated every 10 seconds and also the current cycle period of every cycle is calculated for SoMachine. For TwinCAT, since it gives only the option of a cycle task, the cycle time doesn't need to be calculated as average so this part was removed. Instead, an internal system variable is used now for the measurement of the cycle period.
- A distinction between Binary and Analog tables didn't exist. Binary and Analog tables are handled normally in the same way (each table is chosen to be sent if at least one bit of a word has changed from the previous cycle) and the only reason for distinction is that the first Binary table, which contains the timestamp and the alive counter, should be sent every one second. So a flag was put to indicate Analog or Binary data.
- Every change of the Binary status needs to be recorded with the timestamp of its occurrence. It has to be forwarded to SCADA, where it is saved in an archive. This is important because it gives a complete image of all events that occurred within one PLC cycle. To achieve this, the middleware saves at each PLC cycle the the Binary Status buffer and in the next cycle, it compares the two Binary buffers (new and image, old). For each word, when at least one bit differs, the two words (old and new) are saved in an Event buffer with the PLC timestamp and sent to SCADA. The timestamp header of event tables should be different from the Binary/Analog. The reason is that SCADA needs to distinguish event frames from status frames. So now the UNICOS year is calculated for both cases.
- Event tables are written to only one table which is sent whenever is full or within a safety push. A safety push was also implemented every 10s.
- In the Modbus writing function the hardware address where the message is going needs to be specified. For connection with a PLC the hardware is the number of the memory address e.g.1000. For communication with WinCC OA instead it is the reference number 0xFFFF (65535 or -1).

- In order to have a Modbus communication with the TwinCAT PLC, a TwinCAT Modbus function needed to be installed on the PLC.

Things that still need to be checked are the functionality of the Recipes, the Continuous push of data to the SCADA and the performance of the communication.

5 Demonstration and testing of CODESYS Plug-ins and Resource package

In order to demonstrate and test the functionality, of the solution that was given for the integration of the two CODESYS-based PLC platforms in UNICOS-CPC, a control application was developed for both Schneider SoMachine and Beckhoff TwinCAT. Also, in order to validate the Baseline library, an automated test focused on the CPC Field Objects was performed. Finally, a control application was developed and run on a conventional PC, in order to show the openness of the solution, also in PC-based systems.

5.1 An example of platform independence on a real control application

In figure 23, the procedure for creating a CPC application for CODESYS-based controllers with CPC is presented. First, the inputs of UAB tool (Spec file, User Templates, communication parameters) have to be specified in the UAB Wizard. CODESYS plug-ins generate the PLCOpen XML output files, according to these inputs. The generated files should be then imported in the development tools, in this case either SoMachine or TwinCAT, which have CODESYS environment as a kernel of their software. More specifically the UAB outputs are imported to the Baseline project. The Baseline PLCOpen XML files of the CPC objects are already imported and provided in the Baseline application. After compilation, native code is produced and can be downloaded and run in the programmable controller. The controller can be an IPC, EPC, PAC or even a PC. In this application it was used an EPC and an IPC from Beckhoff and a PAC from Schneider. The controller can be linked to the hardware I/Os via fieldbuses.

5.1.1 Description of the control application

A real control application was developed for testing purposes in the PLC's section lab. The application had been also developed for a Schneider PLC in the past, so the instrumentation hardware as well as the functional analysis of the application already existed. The goal of this application is to prove the functionality of the solution that was given and its independence from hardware between the two different supplier platforms, Schneider SoMachine and Beckhoff TwinCAT.

The considered CPC application includes all the three layers of a control system, as mentioned in paragraph 3.1.1. For the Supervision layer, the WinCC OA SCADA for remote control and the Magelis Vijeo Touch-panel for local control, were used. For the Control layer, three controllers were separately used: the Schneider's M258 controller and the Beckhoff's

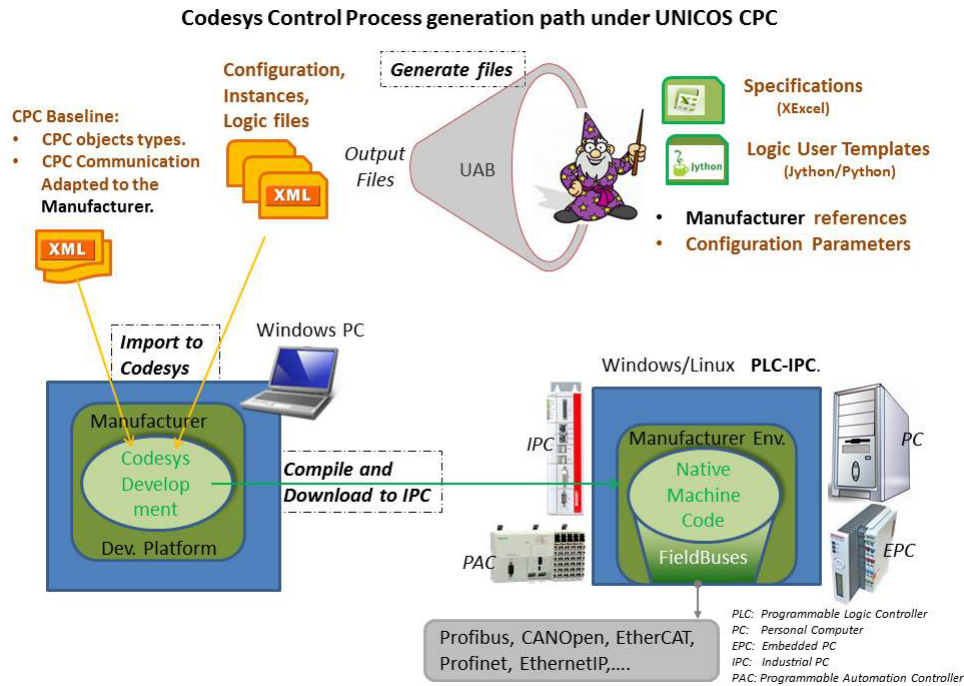


Figure 23: Creating a CODESYS application with UAB [16]

CX5020 EPC and the C6920 IPC. For the Field layer, since it is a lab application and the equipment is close, I/O cards were attached to the PLC and the EPC, and EtherCAT fieldbus was used for the connection of the I/O cards to the IPC, as the last one does not give the possibility to directly attach I/O cards on it.

A representation of the control system can be seen in figure 24. The system consists of three water tanks: tank 1 (the one in the middle), tank 2 (the one at the bottom) and tank 3 (the top one). The sensors and the actuators of the control system, with their names, are also presented in the figure. The application controls the distribution of water between the three tanks and the heating, cooling, PID control of the water distribution and PID control of the temperature, in tank 1. The equipment can be controlled by manual commands or by running automated processes, by selecting the different option modes of the PCO (Process Control Object) objects of the application. Both can be done from the WinCC OA SCADA application of the system.

In figure 25, the hierarchy of the PCO and the Field objects of the application is shown. In blue color, are the three PCO objects of the application. The TestBench PCO, that is called Top PCO, controls the other two PCO (that are called children of the Top PCO). In yellow color, are

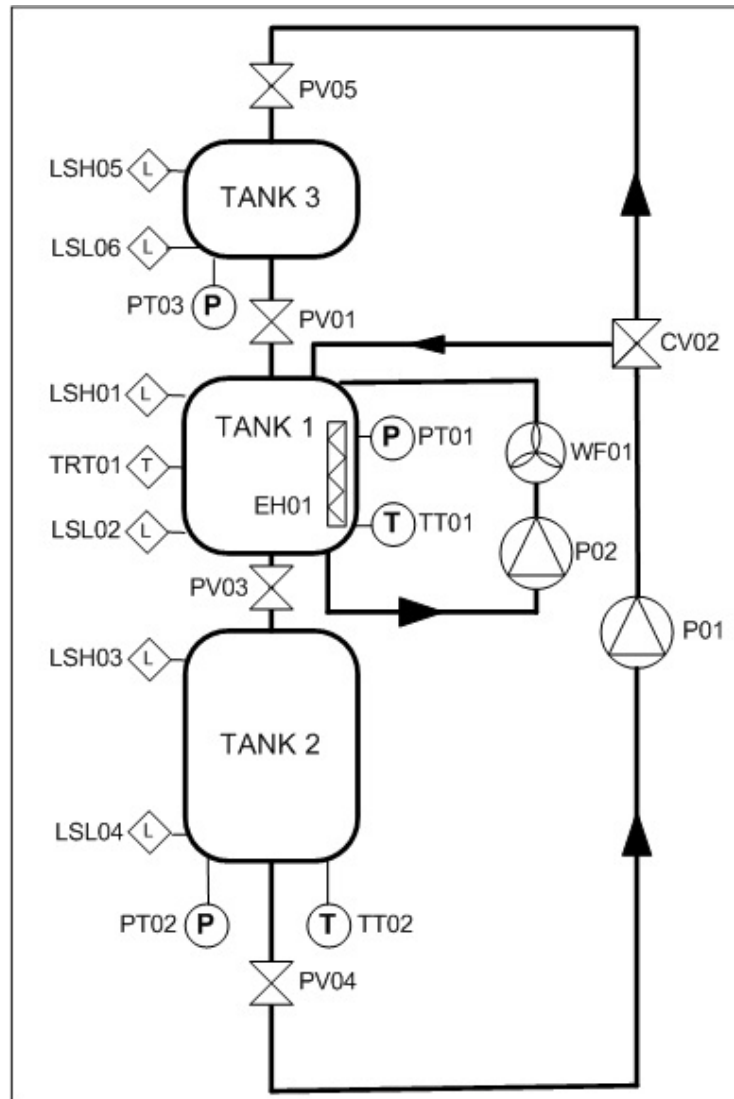


Figure 24: Overview of the process

the Field objects (actuators) that are controlled by the two children PCO. In the application, there also two Controller objects that control the analog pump P01 and the analog-digital heater EH01, Analog and Digital Alarm, I/O and Interface objects.

The Top PCO (TestBench PCO) option modes can be controlled by SCADA operator commands. This PCO controls only its two children PCOs the DISTRIBUTION and the TEMPERATURE PCO but it doesn't control directly any actuator. It has the following six option modes and each of them activates one or more option modes of the children PCOs. The children option modes that are used by the Top PCO's option modes are

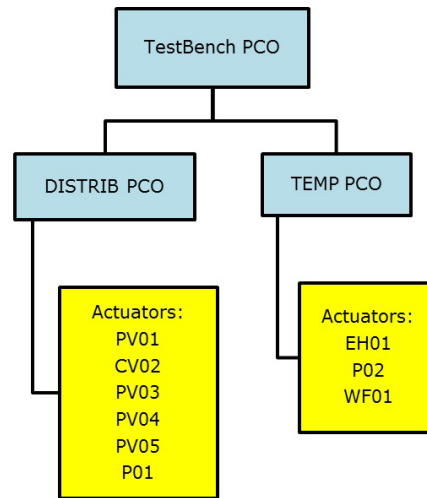


Figure 25: Process decomposition

put in parenthesis.

- The *PIDLevel* mode empties tank 3, fills tank 2 (Purge) and then regulates tank 1 level (PIDLevel).
- The *PIDTemperature* mode empties tank 3 and fills tank 2 (Purge), then it fills tank 1 with the required level (DISTtoT1) and finally regulates tank 1 temperature (PIDTemperature).
- The *CoolingControl* mode empties tank 3 and fills tank 2 (Purge), then fills tank 1 with the required level (DISTtoT1), and finally cools down tank 1 water (Cooling).
- The *HeatingControl* mode empties tank 3 and fills tank 2 (Purge), then fills tank 1 with the required level (DISTtoT1) and last warms up tank 1 water (Heating).
- The *Reserve* mode empties tank 3 and fills tank 2 (Purge), then fills tank 1 with the required level (DISTtoT1) and last fills tank 3 with the required level (DISTtoT3).
- The *SafeStop* mode fills tank 3 and empties tank 1 (SafeStop).

DISTRIBUTION PCO (child of TestBench PCO) is a liquid distributor between the three tanks by using a pump and digital valves. It controls the following Field objects-actuators:

- PV01: output valve of tank 3
- CV02: three ways valve

- PV03: input valve of tank 2
- PV04: output valve of tank 2
- PV05: input valve of tank 3
- P01: distribution pump via the Controller object PC01

And it has five option modes:

- The *Purge* mode empties tank 3 and fills tank 2.
- The *Distribution to tank 1* mode fills tank 2 to a desired level.
- The *Distribution to tank 3* mode fills tank 3 to a desired level.
- The *SafeStop* mode fills tank 3 and empties tank 1.
- The *LevelReg* mode uses PID control of the analog pump to maintain the level of tank 1 at a certain value while it is continuously emptying.

TEMPERATURE PCO (child of TestBench PCO) runs processes to cool down, warm up the water and does temperature PID regulation in tank 1. It controls the following Field objects-actuators:

1. P02: Analog (cooling) pump
2. WF01: Onoff, heat exchanger fan
3. EH01 Analog-digital heater via the Controller object TC01

And it has three option modes:

- The *Cooling* mode cools down the water of tank 1 to a desired temperature.
- The *Heating* mode warms up the water of tank 1 to a desired temperature.
- *PIDTemperature* mode uses PID control of the analog-digital heater to maintain the temperature of tank 1 at a certain value, while it is continuously cooling.

5.1.2 UAB generation procedure

5.1.2.1 UAB Inputs The inputs that should be specified by the user for the UAB application are the User Logic Templates, the Specifications file and the parameters that are filled in the Wizard.

The User Logic Templates are complementary to the Resource Package's Templates, that contain the logic of the application. They call Core and Plug-in methods in order to write text (PLC code in PLCOpen XML format) to the output file. The PLC code in the User Logic Templates (as in all the Templates) is based on IEC 61131-3 programming rules.

User Templates have been created for all the logic sections of the PCO and the Dependent Logic (DL) of the Controller and Field Objects, as they had been described in paragraph 4.2.2. The User Templates for the application are exactly the same for both development software, SoMachine and TwinCAT. No vendor-specific language extensions and programming techniques interfere in the programming of the application.

The User Specific Templates should be placed in the *UserSpecific* folder, under the UAB application's directory, so that they can be processed by the Logic plug-in. Moreover, in the spec file a reference should be done, in order to call the User Template instead of the Standard one. For each specific object and logic section, the name of the Template that should be used instead, has to be given in the spec file. In figure 26 is shown the case of defining the User Template, of the Transition Logic (TL) section of the USC_864_DISTIB PCO object . This is done in the spec file under *CustomLogicSections* attribute. The same procedure was followed for all the User Templates.

1	50	51	52	53
Name				
Object Type Family				
Description				
Version				
Version Comments				
Status				
Help	Help	Help	Help	Help
DeviceIdentification	LogicDeviceDefinitions			
	CustomLogicSections			
Name	GL User Template	TL User Template	SL User Template	CDOL User Template
USC_864_TestBench		TestBench_TL_Template.py	TestBench_SL_Template.py	
USC_864_DISTIB	DISTRIB_GL_Template.py	DISTRIB_TL_Template.py	DISTRIB_SL_Template.py	
USC_864_TEMP	TEMP_GL_Template.py	TEMP_TL_Template.py	TEMP_SL_Template.py	

Figure 26: User Template definition in the Spec file

The second thing that is needed for the application is the filling of the Specifications file. An empty spec template is provided automatically, when a new UAB application is created. Then, it should be filled with the CPC objects of the application and their specific attributes, according to the functional analysis. The CPC objects that are needed in order to program the control application, can be seen in the table 2. The spec file also offers an excel sheet *UCPCSspecDoc* with descriptions of the objects' attributes in order to facilitate the filling of the spec file. The object at-

tributes are related to ranges, option modes, alarms, SCADA parameters, User Templates etc. The values of these parameters are used by the logic in the Jython Templates in order to write the output file. An example of the Spec file for the Analog Input objects and their parameters can be seen in figure 27.

Object	Quantity	Description
Digital Input	7	Level sensors and a thermostat
Analog Input	5	Pressure and Temperature sensors
Digital Output	7	Valves, a cooling fan and a heater
Analog Output	2	Pumps
Digital Parameter	7	Boolean Parameters that can be modified by SCADA operators
Analog Parameter	58	Analog Parameters that can be modified by SCADA operators
Word Status	6	Word statuses of the PLC that are sent to SCADA
Analog Status	3	Analog statuses of the PLC that are sent to SCADA
OnOff	6	Valves and a cooling fan
Analog	2	Pumps
Analog Digital	1	Heater
Controller	2	PID control of the temperature and the level of tank 1
PCO	3	One for distribution processes, one for temperature processes and one for controlling those two for combined processes
Digital Alarms	12	Alarms based on Boolean conditions
Analog Alarms	6	Alarms based on non-Boolean conditions

Table 2: The CPC objects of the Demonstrator application

Finally, the parameters concerning the PLC and its communication with the SCADA should be filled in the UAB Wizard by the user. These parameters are the following:

- the PLC Name, Environment(TwinCAT/SoMachine), Type(EPC, IPC/M258)
- the PLC IP address, gateway and network mask
- the IP of the SCADA Data Server

1	2	3	6	7	8	9	10	11	12
Name	AnalogInput					TRUE	FALSE	alogInput_SmIogInput_SciOrfRecs	
Object Type Family	IDObjectFamily								
Description	Analog Input Device								
Version	changedRevision: 00000 \$								
Version Comments									
Status									
Help	Help	Help	Help	Help	Help	Help	Help	Help	Help
DeviceIdentification	DeviceDocumentation	FEDeviceParameters							
Name	Expert Name	Description	Range Min	Range Max	Raw Min	Raw Max	Deadband (%)	Encoding	InterfaceParam
USC_864_PT02	CXPT02	Tank 2 pressure sensor	0	32.3	1980	11282	0.1	1	%IW9
USC_864_PT03	CXPT03	Tank 3 pressure sensor	0	12.2	1660	4730	0.1	1	%IW11
USC_864_TT02	CXTT02	Tank 2 temperature sensor	-200	850	-2000	8500	0.095	1	%IW6
USC_864_PT01	CXPT01	Tank 1 pressure sensor	0	32.3	930	10810	0.1	1	%IW8
USC_864_TT01	CXTT01	Tank 1 temperature sensor	-200	850	-2000	8500	0.095	1	%IW5

Figure 27: Specifications file-Analog Input objects

- the Netlink, for Ethernet communication with the SCADA (is used in only by SoMachine)
- the number of Analog and Binary tables sent to the SCADA per PLC cycle
- the Modbus Unit(should be unique in a WinCC OA application)
- the PLC and SCADA starting address for the mapping of the Recipes as well as their length and if or not they will be generated
- the PLC and SCADA starting mapping address for the PLC statuses (Binary/Analog) and the starting mapping address of the SCADA requests (Binary/Analog).

5.1.2.2 Generation of the output files In order to generate the importation files for SoMachine and TwinCAT development software the UAB tool is used. First, a new UAB application should be created. For each of the three controllers of the Demonstrator application, a different UAB application was created. The procedure for the generation of the output files through the UAB Wizard, is the same for the three systems and only the parameters that are filled are specific to each controller.

In the UAB Bootstrap, the creation of a CoDeSys application and the version of the Resources Package, that will be used for the generation, are chosen from the menu and the application's folder is specified (Figure 28). By clicking on the *Run* button, the folder structure of the application is created and a window for the UAB Wizard is opening in order to guide the user through the generation procedure.

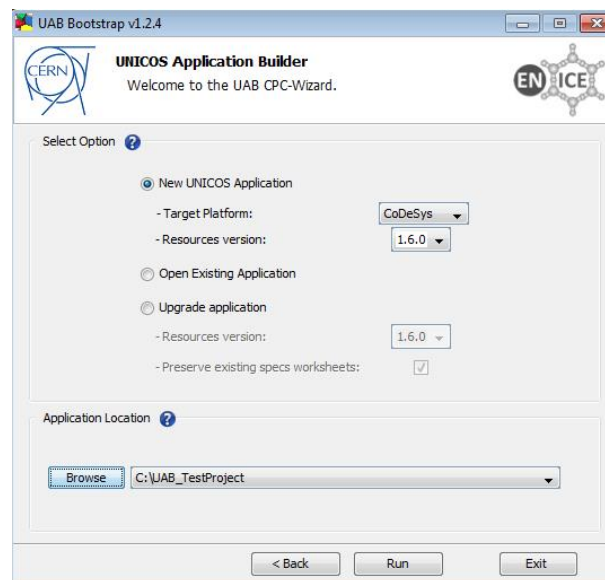


Figure 28: Create a new UAB application

The first panel of the Wizard is shown in figure 29. In this panel the Application and the Project's names are given and the directory of Specifications file is specified. Optionally, extra information to the application can be introduced, like a description, a comment or a contact person.

Next, is the *PLC Specifications* panel (Figure 30), where the Wizard parameters that were described in the above paragraph are filled. As it was mentioned, for each of the three controllers of the Demonstrator application, a different UAB application was created, so that the specific PLC parameters, as the Name, the type, the IP address etc. , are taken into account in the generation. In the picture the parameters for the Schneider M258 PLC are shown.

UCPC # v1.0 # Wizard v1.6.0

CPC Wizard
Application General Data
Resources: 1.6.0

Application ?

Project Name: Demonstrator

Application Name: Demonstrator

Application Version: 1.0

Specifications File: Specs/Spec_Demon_CDS.xml

Optional Parameters ?

Project Description: Lab Project

Contact Person: mkoutli

Comment:

Date: 2013-12-12T18:07:24.485+01:00

Figure 29: Application General Data

The next panel, in figure 31, shows the CoDeSys generators (plug-ins) that should be used for the generation of the application. The first two concern the PLC program and the other two the supervision.

For the Instance files generation, the option CoDeSys Instance Generator is chosen and the corresponding panel appears, Figure 32. The location of the templates, as well as the location of the output files is available and can be accessed directly through the panel by the *Open Folder* button next to each directory. Under those directories, there is a checkbox, giving the option for processing or not the Semantic Check Rules during the generation. A step lower in the panel, are located two checkboxes that provide the option to generate or not the Communication and/or the I/O Commissioning file. The communication file is used for the communication between the PLC and the SCADA and should always be generated since in the topology file there is reference to this program. The Post Process User Templates were not needed for this application.

In the panel there is the list of all the CPC objects this application consists of, as well as their quantity (information coming from the spec file). All the objects are selected for the first generation. Only in case of a change in a specific object(s), generation can be done by selecting only

UCPC # Demonstrator v1.0 # Wizard v1.6.0

CPC Wizard: Demonstrator - Demonstrator v1.0
CoDeSys PLC Specifications
Resources: 1.6.0

General Data

PLC Name:
 PLC Env.:
 PLC Type:

Recipes

Enable recipes: ☒
 Max. number of recipe values:
 Recipe Buffer Starting Address:
 Activation Timeout (s):

Ethernet Parameters

PLC IP Address:
 PLC Gateway:
 PLC Network Mask:
 DS1 IP Address:
 DS2 IP Address:

MODBUS Parameters

MODBUS Unit Address:
 NetLink:
 Binary Tables per Cycle:
 Analog Tables per Cycle:

Mapping

Transmit Buffer Starting Address:
 Receive Buffer Starting Address:

Back Next Exit

Figure 30: PLC Specifications



Figure 31: CoDeSys Generators

the desired object(s) that needs to be regenerated. It is important to notice also, that in the output file are also included the global variables definition. So a selection of only some of the objects for generation, will give to the output only the global variables of those objects.

After selecting the objects and the files to be generated the *Generate*

button is pressed. The generated files are located under the directory ...Output/CodesysCodeGenerator/, of the UAB application folder. The files in this folder are the plc_instances_ file.xml, the communication_DS.xml and the IOCommissioning.xml, if its generation is chosen to be performed. In the generators panels there are buttons at the bottom of the window for navigating between generators.



Figure 32: Instance generator

For the generation of the Logic code, the CoDeSys Logic Generator is selected next. Figure 33 shows the panel dedicated to the logic generation. All the objects are selected for generation the first time. After selecting the objects for generation the *Generate* button is pressed. It is mandatory for the first generation to generate all the logic (object types, masters and the logic sections), since inside the topology file there are references to these logic sections. One or more objects, which will maybe be modified in the future (e.g. with the User Templates) can be generated separately.

The *Templates Development API* button of the panel is opening a pdf file that gives guidelines for the creation of the User Specific Templates.

The output files can be found in the directory ...Output/CodesysLogic-

Generator/, which includes the `plc_logic_file.xml` and the `topology_file.xml` files.

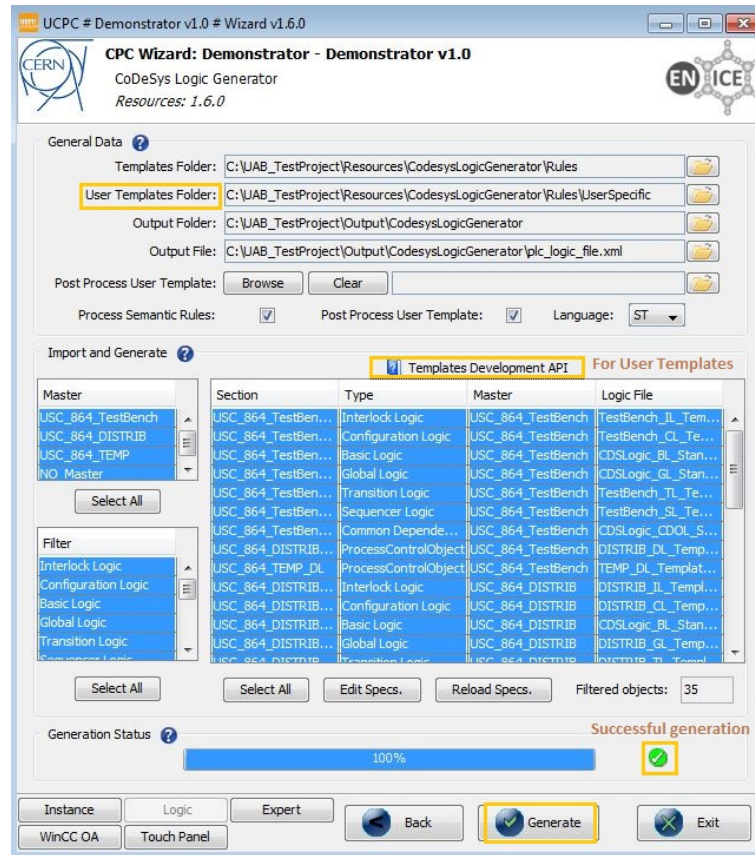


Figure 33: Logic Generator

The third generator, the WinCC OA Instance Generator, is displayed below in figure 34. Again, all the CPC objects are selected to be generated. The generated file for the WinCCOA importation is the `wincc_oa_db_file.txt` and is located in the `...Output/WinCCOAInstanceGenerator/` directory.

In figure 34 can be seen a warning triangle. This appears when there is a change in the specifications file since the previous generation. The specifications file can be modified by the *Edit specs* button and reloaded with *Reload Specs* button, in order to take into account the new changes. This applies for all the generators.

The forth generator is also used (Figure 35) , since in this application the Magelis Vijeo touchpanel was used. In the panel the Magelis touchpanel is chosen from the menu and all the objects are selected for generation. The output file `vijeo_variable.xml` is located under the `...Output/TouchPanelGenerator/` directory.

The UAB User Report window, that shown in figure 36, is a very



Figure 34: WinCC OA Generator

helpful tool to show information and errors that are detected during the generation and the reason why they occurred. The various messages can be hidden or displayed according to the user's choice for a specific category of messages. The categories are: Severe, Warning, Information, Configuration, Fine and Debug. Additionally, there is a button for clearing the report data and one for copying them to clipboard. In figure 36 for example you can see a Warning message that there are no AnalogInputReal Instances in the spec so they will not be generated. When the generation finishes with no severe errors then in the User Report Window will appear a message that the exit status of the Generator plug-in is a success.

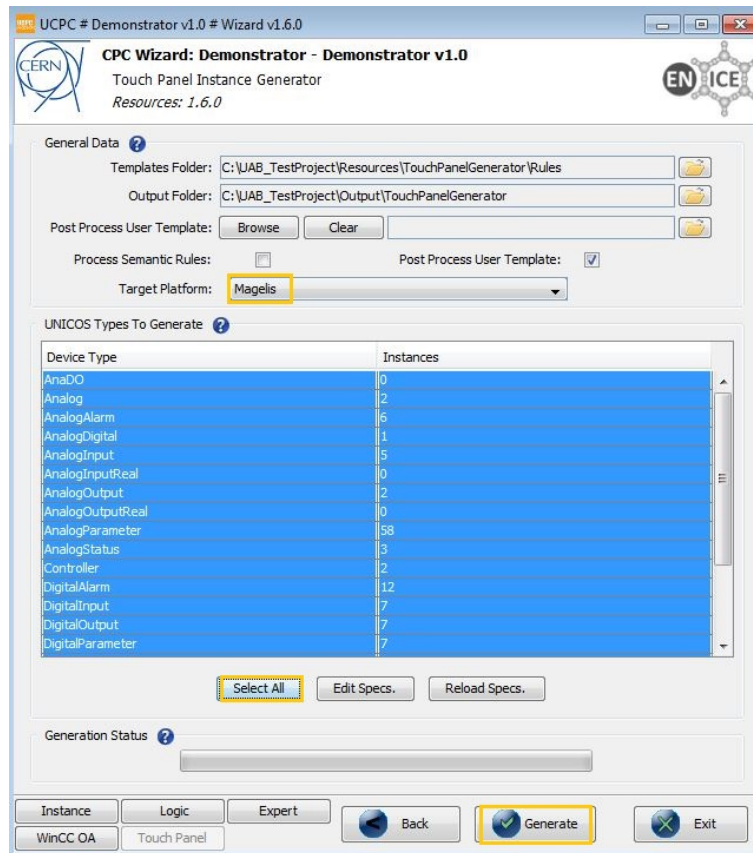


Figure 35: Touchpanel Generator

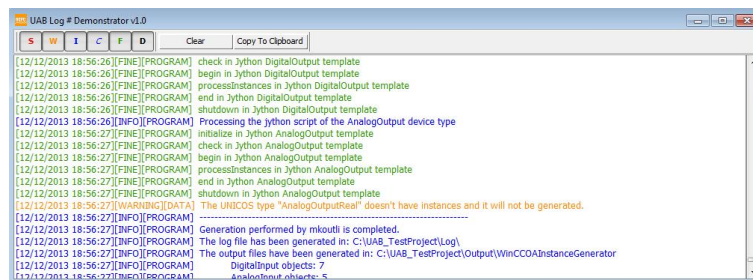


Figure 36: Log Window

5.1.3 Importation to Somachine

The four plug-ins (generators) that were described above, have generated the importation files for the Control and the Supervision layer. For the Schneider controller M258, the control application was created with the SoMachine development tool. The four generated PLCOpen XML files, from the CODESYS Instance and Logic plug-ins, should be imported in

the SoMachine project.

First, the Baseline SoMachine project has to be opened. The project contains all the CPC objects and the communication FBs of the Baseline library and is placed under the UAB application folder.

The Baseline SoMachine project is predefined for the Controller TM258 LF42DT, which is the one that is used for this application. More I/O modules needed to be added to the four existing ones, that were already attached to this controller in the configuration, so that the PLC can connect to the sensors and the actuators of the control system.

The I/O modules that were used for the connection to the sensors and the actuators of the application are shown in table 3, with their quantity and their description.

I/O Module	Quantity	Description
DI12DE	1	12 channels Digital Input 24 VDC
DO12TE	1	12 channels Digital Output 24 VDC
SAI2PH	1	2 channels Temperature Input
SAO2L	1	2 channels Analog Output $\pm 10V$
SAI2L	2	2 channels Analog Input $\pm 10V$

Table 3: Schneider's M258 I/O modules for Demonstrator

The adding of the modules is done by selecting in SoMachine the Tab *Configuration* and then *Add Expansion Module*, as shown in figure 37 α '. In the same Tab by double-clicking on the controller a new panel appears. There, by selecting Communication-> Ethernet-> Physical Settings, in the *MyController* list that appears in the left side of the editor, the IP address of the controller can be set, figure 37 β '. This IP is given to the PLC when the program is downloaded. The parameters that were given here, should match with the Ethernet parameters in the PLC Specifications panel, in figure 30.

Also, in the *Configuration* Tab, by selecting the *Parameters* option, a new panel appears. In *Communication settings* tab, the connection of the development application to the PLC via the CoDeSys Gateway can be established. First, the gateway should be added and then it should be scanned for connected devices. In order to be able to connect with the PLC, the developing PC should be in the same sub-network with the PLC. This is because SoMachine is using a protocol feature (Broadcast IP) in order to detect and connect to a PLC, that is not allowed to pass through CERN backbone routers. So in this case the PC that is used for the application should be connected to the same side of the router as the M258 PLC. After the physical connection is established via Ethernet cables the PLC can be detected by Somachine (figure 38). Finally, the connection with the device that was found, should be activated .

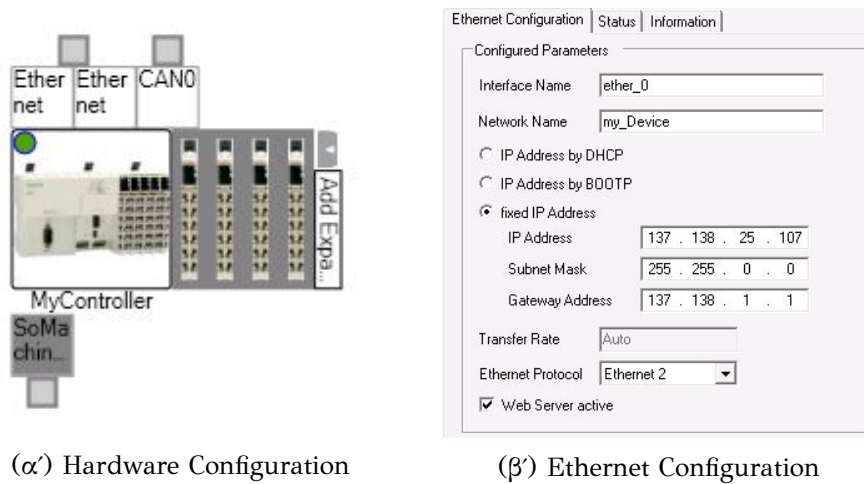


Figure 37: Demonstrator SoMachine

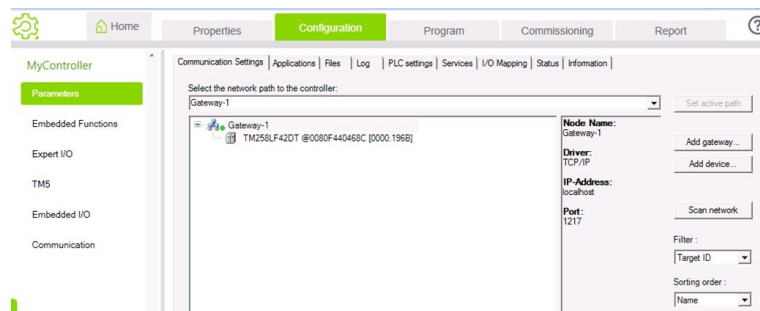


Figure 38: Connection to M258

In the *Program* Tab the application with the imported Baseline library (Functions, FBs and Structures), appears. The four generated files from UAB should be successively imported, figure 39. In order to have a better structure, two folders were created, one for the Instance and one for the Logic imported programs.

Note that only the files for the SL (Sequencer Logic) sections of the three PCO objects should be manually created in the application, since SFC language for PLCOpen XML importation is not currently supported by SoMachine.

After selecting the file for importation, a window pops-up, showing all the POUs (Programmable Organization Units) which will be imported (Programs, GVL etc.). All the files are selected to be imported the first time that the application is created (Figure 40). In the *Additional Information* tab, the information coming from the File and Content header of the XML file can be seen.

This procedure is followed for all the four generated files that come

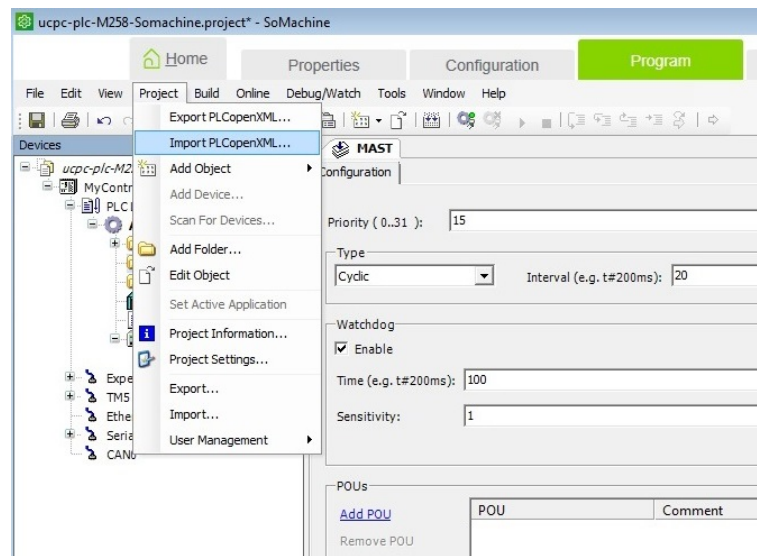


Figure 39: Import PLCOpen XML files into SoMachine

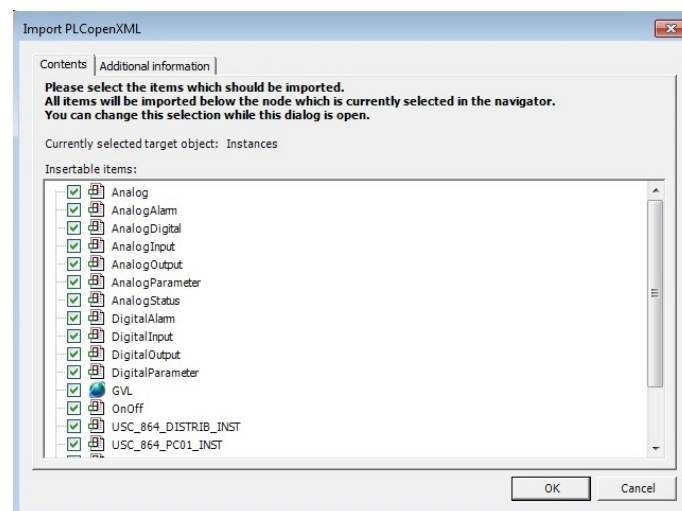


Figure 40: Choose the objects for importation

from the Instance and the Logic CODESYS plug-ins.

The Topology file imports a program, which sets the execution order of the imported programs. In order to be taken into account, this program has to be added in the *Task Configuration* of the application. There, other Task parameters, as the PLC cycle, can be set.

After the necessary files have been imported and the execution order of the imported programs has been defined, the application is compiled and then downloaded to the PLC to Run. In the Online Mode the software

offers an online view and change of the PLC variable values.

5.1.4 Importation to TwinCAT

For the Beckhoff's EPC CX5020 and IPC C6920, the control application was developed with the TwinCAT development tool. The four generated PLCOpen XML files, from the CODESYS Instance and Logic plug-ins, should be imported in the TwinCAT project.

First, the TwinCAT Baseline project is opened. This project contains all the CPC objects and the communication FBs of the Baseline library and is placed under the UAB application folder.

The application doesn't have an existing hardware configuration (as in SoMachine). The PLC target should be specified in the TwinCAT application. From the *SYSTEM* module on the left menu of the project after the *Search Ethernet..* button is pressed a dialog window opens. This window is shown in figure 41. Here, the IP of the target PLC has to be specified and then the *Enter Host Name/ IP* button is pressed to add the target. After the device is found, the option *Add Route* should be selected to connect to the device. In the next dialog panel the default Username used is *Administrator* and the Password is *1*. X in the column *Connected* indicates a successful connection to the target. This action needs to be done only the first time that this target is used for a TwinCAT application in that specific developing PC. For the Demonstrator application in TwinCAT, two different applications were created for two different Beckhoff target controllers: the CX5020 EPC and the C6920 IPC.

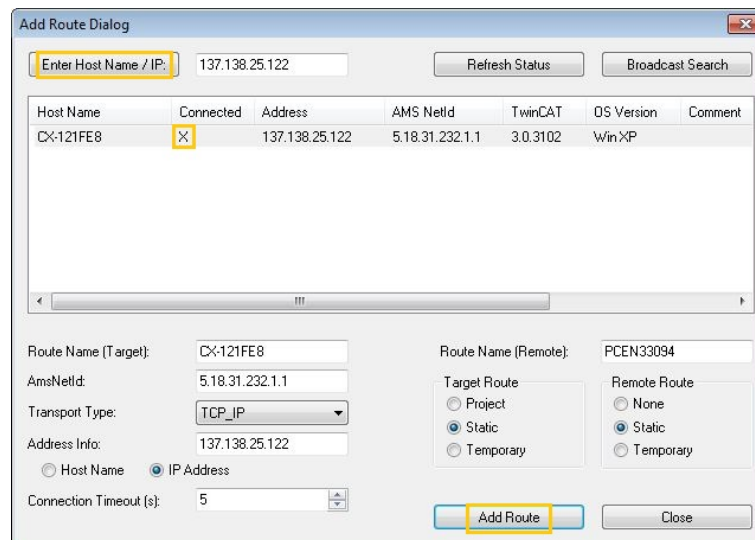


Figure 41: Connect to TwinCAT target

The UAB output files can be imported in the *PLC* module, where the

Baseline library is already imported. As it was done in SoMachine, two folders for Instance and Logic importation files can be created. The files can be imported in each folder, as it is shown in figure 42.

For the Topology file the predefined Task of the application has to be deleted first. Then, with the importation of the Topology file, a new Task is imported in the *SYSTEM* module, with a Referenced Task in the *PLC* module. The Task contains the execution order of the programs and it is a cyclic task with name Mast, cycle time 10ms, priority 9 and watchdog not enabled. These characteristics can be also modified.

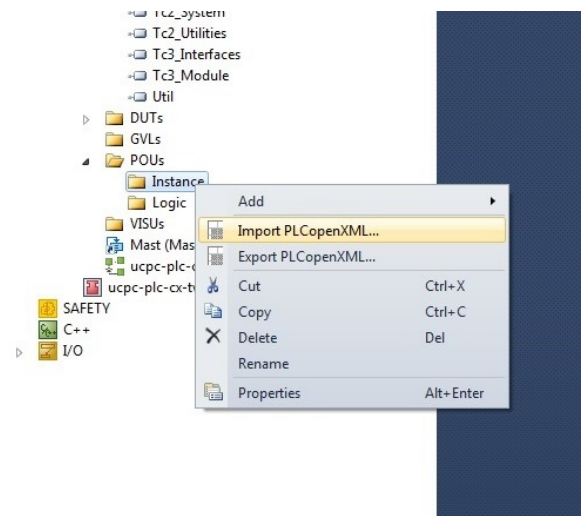


Figure 42: Import PLCOpen XML files to TwinCAT

After all the files are imported and their execution order has been set, the application can be built. After the compilation, under the project's instance the mapped input and output variables of the program are listed, figure 43. These variables will be mapped with the appropriate input/output channels of the hardware modules.

The I/O devices that are attached to the PLC should be detected. This is done in the *IO* module by right-clicking on the *Devices* and choose the *Scan* option. For the EPC the search finds both EtherCAT bus and K-bus devices. In this application K-bus was mainly used, because the EtherCAT modules were used for the IPC. Nevertheless, a configuration with I/O modules for the EtherCAT bus of the EPC was also tested, offering a more quick I/O task cycle comparing to the K-bus. The I/O modules attached to the EPC's Bus Coupler are shown in table 4 with their quantity and their description:

The scan detects the above I/O modules and the hardware configuration of the EPC for this application can be seen in figure 44α'.

For the C6920 IPC, an EtherCAT Bus is detected and the scan for

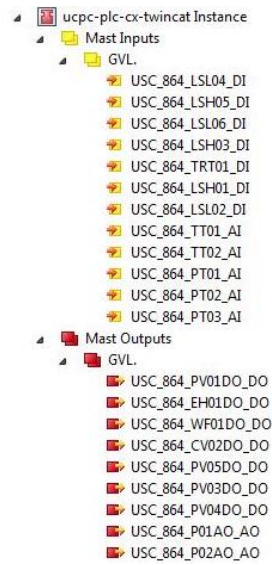


Figure 43: TwinCAT input/output variables

I/O Module	Quantity	Description
KL1408	1	8-channel Digital Input 24V DC
KL2408	1	8-channel Digital Output 24 V DC
KL3202	1	2-channel Input PT100 (RTD)
KL4002	1	2-channel Analog Output 0...10 V
KL3022	2	2-channel analog input 4...20 mA
KL9010	1	End-Terminal

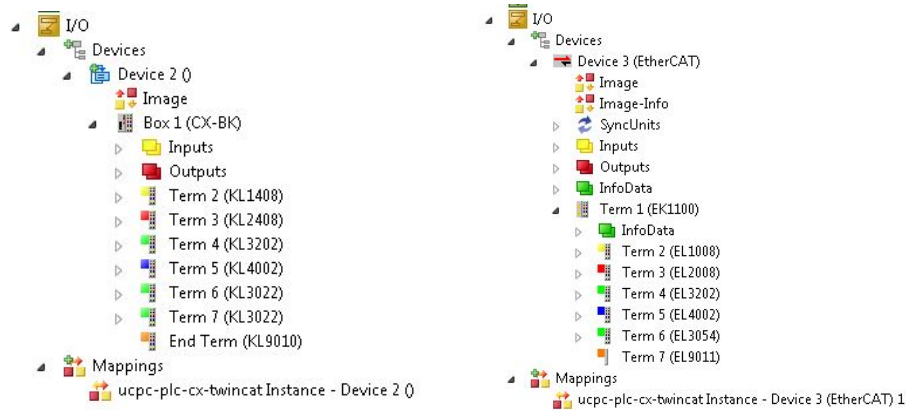
Table 4: Beckhoff's EPC I/O modules for Demonstrator

Boxes detects the EtherCAT Coupler EK1100 and the EL modules that are listed in table 5. The I/O modules are attached to the EtherCAT Coupler, which is connected to the IPC via an Ethernet cable. Figure 44β' shows the whole hardware configuration of the IPC.

I/O Module	Quantity	Description
EL1008	1	8-channel Digital Input 24V DC
EL2008	1	8-channel Digital Output 24 V DC
EL3202	1	2-channel Input PT100 (RTD)
EL4002	1	2-channel Analog Output 0...10 V
EL3054	1	4-channel analog input 4...20 mA
EL9011	1	End-Cap

Table 5: Beckhoff's IPC I/O modules for Demonstrator

After, detecting the connected I/O modules, their channels can be



(α) I/O modules of the CX5020 EPC (β) I/O modules of the C6920 IPC

Figure 44: Demonstrator-I/O modules in the TwinCAT application

mapped to the Input/Output variables that were created after the compilation and are shown in figure 43. Then, in the *IO* module, the *Generate Mappings* option should be chosen in order to create the links and the Configuration can be activated. After the activation of the Configuration the software asks to *Restart TwinCAT in Run Mode*. By selecting *Yes* we can Log in, download and start the project in the target EPC or IPC. This sequence can be performed by the buttons of the TwinCAT menu, as they are shown in figure 45. In the Online Mode the software offers an online view and change of the PLC variable values as well.

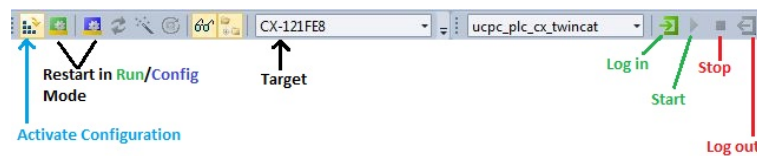


Figure 45: TwinCAT menu

5.1.5 Supervision with WinCC OA

After creating the Demonstrator application for the Schneider and Beckhoff PLCs, the procedure continues with the creation of the supervision application. The first WinCC OA application was with the Resource Package 1.5.0 and the WinCC OA 3.8 version was used. With the current Resource Package 1.6.0 the application was regenerated again, this time for WinCC OA 3.11.

In WinCC OA, first an application was chosen to be created. Then, the Baseline files for UNICOS framework needed to be imported. After the

creation of the application, the output file of the WinCC OA UAB plug-in should be imported. In the importation panel that is shown in figure 46, this database file is selected and then, while the Simulation driver is running a Check of the database is performed. If the check finishes without any errors the database can be imported to the application. After the importation the Simulation driver is stopped and the Modbus driver is started in order to begin the communication with the PLC.

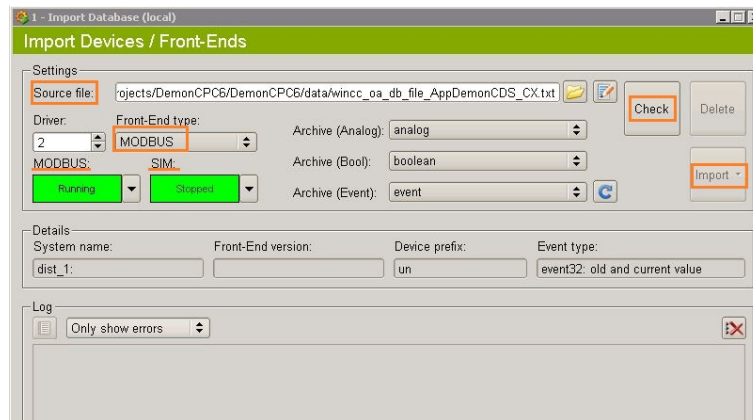


Figure 46: Import Demonstrator's database in WinCC OA 3.11

In the Front-End Diagnostic panel the communication with the PLCs can be checked. The panel is displayed in figure 47. By selecting a PLC from the *Configured Front-Ends*, its communication parameters can be seen. The *Counter* and the *Timestamp*, that were mentioned in the TSPP communication Baseline and are sent from the PLC, are shown in *Communication* Tab. In the *Specific Information* the communication parameters that were given in the UAB Wizard are displayed. The *Send IP* is not specified in the Wizard, but is calculated by the Data Server (DS) IP (the IP of the PC of the WinCC OA application), and is the one that is sent to the PLC in alternation with zero. The green color in the PLC alarms indicates a well established communication.

A main panel was made for navigating through the different PLC applications. Each application panel can be accessed by clicking on the corresponding button. A screenshot of the main panel is shown in figure 48.

For the three CODESYS applications for Demonstrator, one panel is used. The names of the objects in the panel are parametrized so for each application the same panel is imported but with different parameters. Buttons can be also defined in the down part of panel for easy navigation between the panels. All the above are UNICOS/UNICORE features. An example, is shown for SoMachine panel importation in the window tree in figure 49.

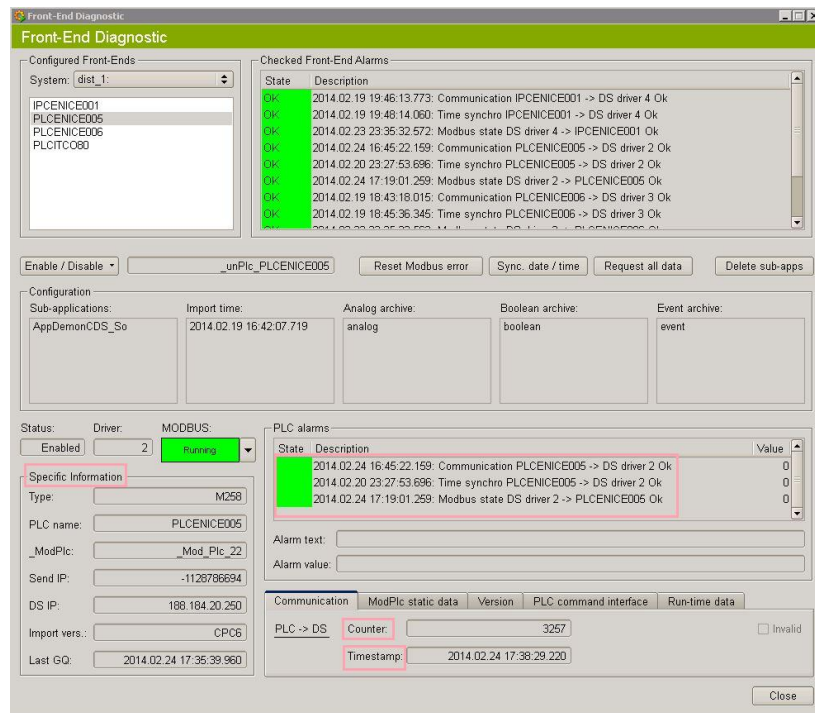


Figure 47: Front-End Diagnostics

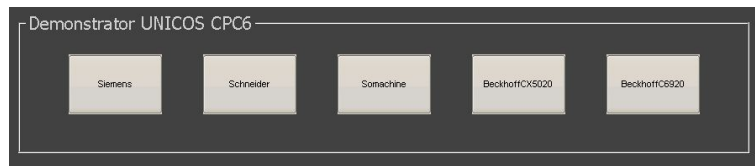


Figure 48: Navigation panel

The Demonstrator application can be remotely controlled via this application. The panel for the Demonstrator application can be seen in figure 50. In this panel the majority of the objects can be seen and easily controlled. By double-clicking on any object widget, a faceplate with command options for the object, is popping-up. The Auto, Manual and Forced Modes that were described in paragraph 3.1.3.1 are also available as buttons in the faceplate of each object.

The communication between the PLC and the SCADA is established with the use of the Modbus/TSPP protocol. Operator requests, as the change of Mode, On/Off requests, setting of parameters and others, are sent to the PLC via Modbus frames. The SCADA is receiving the value of the Binary, Analog data and the Events via TSPP frames.

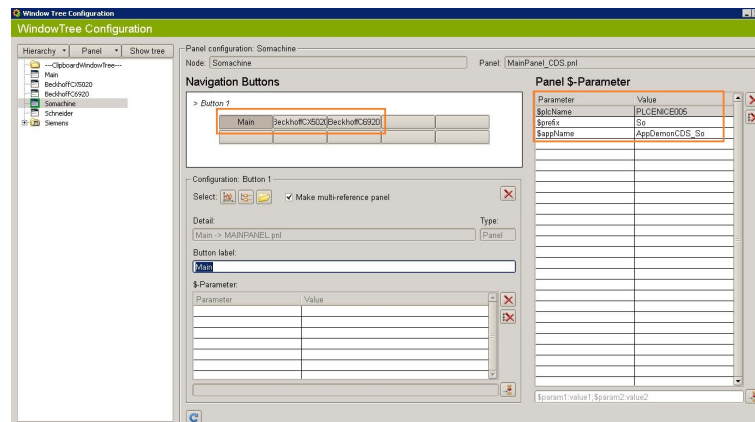


Figure 49: Parametrization of SoMachine panel in the Window tree

5.1.5.1 Control of the application In order demonstrate the behavior of the application, a set of commands was given to the PLC and the response of the process was checked.

First, the actuators were controlled by giving On-Off or Position orders in the I/O objects from SCADA, while they were in Forced Mode. Then, the Field objects were checked, with the same concept, while they operated in Manual Mode e.g. open-close a valve.

After, the manual orders, all objects were put in Auto Mode and were controlled only by the logic of the Top PCO. The different Option Modes of the Top PCO (TestBench) were performed. The Top PCO was operated in Manual Mode, since the Option Modes and the Run Order are given manually in the SCADA faceplate. When an Option Mode is executed and the automated process finishes, the PCO stops and "waits" for the next order.

Then, the children PCOs can be separately controlled. Following the same concept, they are operated in Manual Mode, while all the objects that they control are in Auto Mode. All the different Option Modes were performed.

Finally, the alarms of the system were checked. This was done by manually controlling the Field objects, so that the alarm parameters reach their upper or lower limits. When the limits were reached, an alarm secured the system, by stopping the appropriate device (pump, valve etc) with a Full or Temporary Stop or by preventing them from starting with a Start Interlock. The Digital and Analog Alarm objects are responsible for the implementation of this logic to the other objects. The same thing could happen if there was an unexpected event e.g. a tank with a higher level of water, wouldn't allow the pump and the valves related for the distribution of water to it, to run. The alarm would also prevent the PCO option mode for distribution, that is related to theses valves and pumps,



Figure 50: WinCC OA panel for Demonstrator

from running.

The three applications operate exactly the same with the hardware, as expected. The control process behaves according to the specifications of the functional analysis for the different PCO option modes, in Auto, Manual and Forced Mode and in cases of Alarms.

5.1.6 Supervision with Magelis Vijeo Touchpanel

The supervision with the touchpanel offers a local control of the application. The procedure for creating the Magelis application is the following. First, the Baseline project, that is also placed under the UAB application folder, should be opened. Then, the generated file from the Touchpanel UAB plug-in has to be imported. This file imports the variables that need to be monitored and controlled. The touchpanel is polling the variable values, while it is sending Modbus requests. Then, the panel of the application has to be made by drag and drop CPC object and assign the variables to them. In the application the IP of the touchpanel as well as the IP of the target PLC are specified. In figure 51, the touchpanel that was used for the demonstrator application is shown. The application can be controlled the same way as with the WinCC OA application but the last offers more features and functionality comparing to the touchpanel.



Figure 51: Magelis Vijeo Touchpanel

5.1.7 Hardware of the application

The Demonstrator application can be controlled by any of the three controllers: the Schneider M258 PAC, the Beckhoff CX5020 EPC and the Beckhoff C6920 IPC. The controllers that were used can be seen in figure 52β'.

The control system consists of three water tanks and a list of sensors and actuators that can be seen in table 6. The controlled hardware can be seen in figure 52α'.

Ethernet cables were used for the connection from the PLCs to the CERN General Purpose Network (GPN).

For the connection from the equipment to the PLC I/O modules, DB-25 connectors were used [17]. The wires of the sensors and the actuators were soldered to the pins of two DB-25 connectors. One connector was used for the digital and one for the analog signals. Three different pairs of DB-25 cables were used (one of them is shown in figure 52γ'). In the controller's side the cables were cut and the wires were connected to the specific channels of the I/O modules. This side remained fixed, while in the sensor and actuator's side, the DB-25 cables could be easily changed, according to which controller was going to be used.

One wire is used for the digital signals and two for the analog signals (or more in the case of the temperature signal). The I/O connections wiring is the same for all the modules of the three controllers, except of the Analog Input (AI) modules. The input signal for this module comes from the pressure sensors and it is a 4-20mA current loop.

Sensor/Actuator	Use	Number
Analog pump	Distribution of the water of tank 2 to tanks 1 and 3 and circulating the water of tank 1 when cooling	2
OnOff fan	Cooling water of tank 1	1
Analog Digital heater	Heating water of tank 1	1
Digital sensors	Level sensors	6
Digital valves	Distribution of water between the three tanks	4
Three way valve	Distribution of water between tank 2 and tank 1 or 3	1
Temperature sensors	Measurement of the temperature in tanks 1 and 2	2
Pressure sensors	Measurement of the pressure in the three tanks.	3

Table 6: Sensors and actuators of the Demonstrator application

For Schneider's M258 AI module, a 24 V DC voltage source and a resistance of 250 Ω were connected in series for each of the three signals. The current flowing through the resistance produces a voltage, that can be easily measured by the analog input module (1-5 V). The sensor (transmitter) is not the source of current, but simply regulates the flow and magnitude of the current through it. For 4-20 mA current loops with 2-wire transmitters, the most common loop power supply voltage is 24 V DC. The power supply must be set to a level that is greater than the sum of the minimum voltage required to operate the transmitter, plus the IR drop in the resistance, and for long transmission distances, the IR drop in the wire [2]. For the Beckhoff's EPC AI module, the 24 V DC voltage source needed to be connected in series with the sensor, in order to help the flow of the current. Since the module can measure 4-20 mA current, a resistance was not needed here. For the C6920 AI module, no intervention had to be done, since the 24 V power contact is internally connected to the terminal that one of the wires of the transmitter is going, in order to enable the connection of 2-wire sensors without external supply.

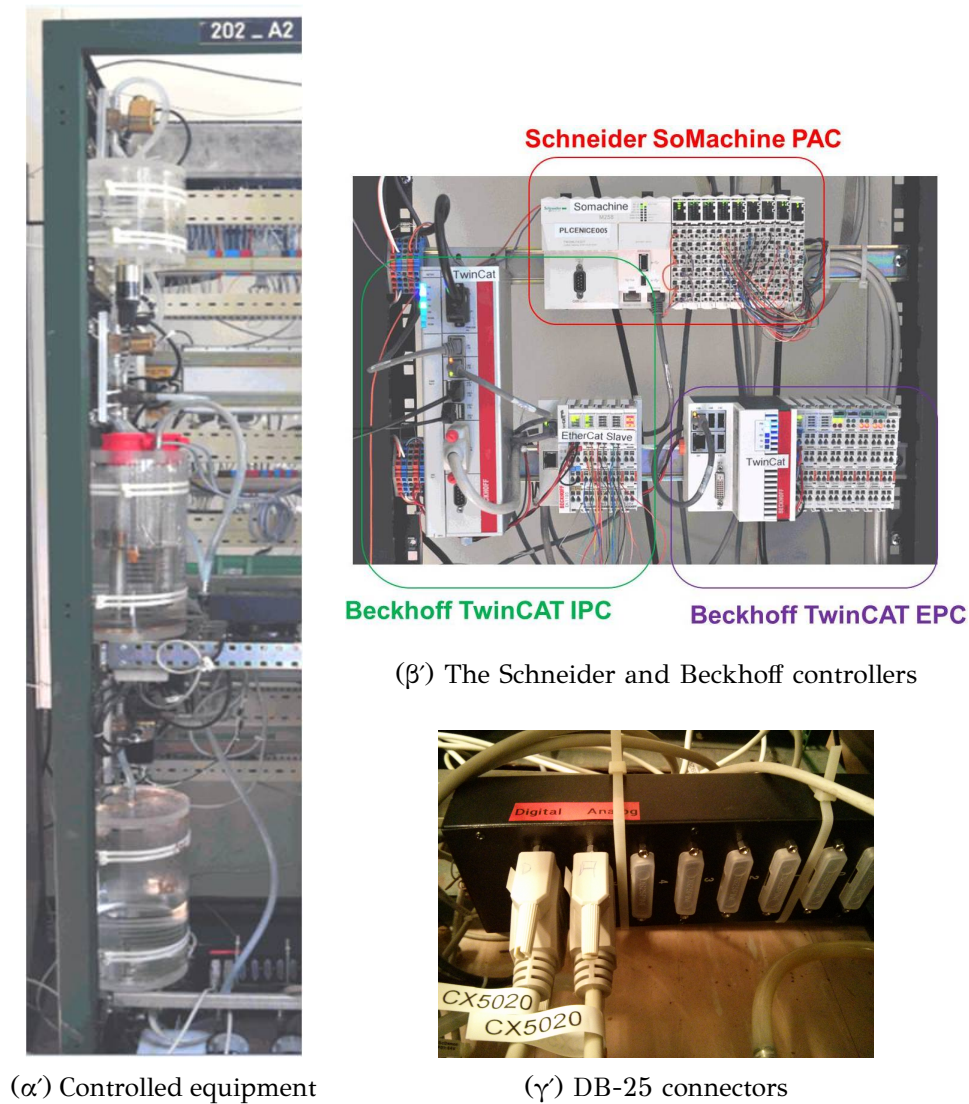


Figure 52: Hardware equipment

5.2 Validation of Field Objects

For the validation of a main part of the CPC Baseline library, a test focused on the Field objects behavior was performed. This test is also used for the validation of the two other Baselines of Schneider and Siemens. It is divided in four parts that check the following functionality:

- WinCC OA operator commands
- Full Stop Interlock
- Temporary Interlock

- Start Interlock

Each part has different steps that they are applied separately to each Field Object. For each step, a condition is set and the appropriate response from the object is expected. Every Field Object type (e.g. OnOff) has a number of different configurations, which are defined in the specification file, that is used for the UAB generation.

For this purpose, there is a dedicated WinCC OA application, where the database of the CODESYS project had to be imported. The series of tests are executed automatically by a script, which is initiated from a WinCC OA panel. In this panel, the validation procedure can be viewed. When one of the tests finishes its name and its result is displayed in the panel. When the whole procedure is finished detailed results can be obtained. If one or more errors have occurred, the step that has failed and the real and expected values of the tested attribute of the object, are provided as well. In case of failure in some tests, their steps can be executed manually, in order to reproduce the failure, while checking the PLC values, in order to detect the fault in the baseline code.

In figure 53 a part of the Full Stop Interlock tests is shown. As an example, the OnOff object's first configuration steps will be described. The behavior of the Digital Output object, that is linked to this OnOff object, is going to be checked if it responds as expected.

In step 1 the object gets an *Auto On Request* and the expected output is *On* (1). When the *Full Stop Interlock* (FS) in step 2 is triggered, the output should go to *Off* and remain even when the interlock is gone (step 3). The output should be *On* again only when the alarm is *Acknowledged* and the object is *Re-started*. Then, the OnOff object is set to *Manual Mode* (step 5). In this case if a FS occurs and then is *Acknowledged*, after the interlock is gone, the expected output is *Off*. The procedure continues to the next steps and successively to the next objects until all the tests are finished.

	FULL STOP	NB: In these tests, feedbacks are kept at FAULTY. ALLOW RESTART ONLY IF FS DISAPPEARS									
Type	Config	Comments	Param	1	2	3	4	5	6	7	8
INPUTS	LOGIC		AuOnR(OnOff+PCO+ANADO)	1	1	1	1	1	1	1	1
INPUTS	LOGIC		AuOffR (OnOff+PCo)	0	0	0	0	0	0	0	0
INPUTS	LOGIC		AuPosR (ANALOG,ANADIG,ANADO)	100	100	100	100	100	100	100	100
INPUTS	LOGIC		Interlock	x	FS	x	ack+AR	x	FS	x	ack+AR
INPUTS	MANUAL		PVSS action (OnOff+PCO+ANADO)	x	x	x	x	manual	x	x	x
INPUTS	MANUAL		PVSS action (ANALOG,ANADIG,ANADO)	x	x	x	x	manual	x	x	x
ONOFF	config1		DO On	1	0	0	1	1	0	0	0
ONOFF	config2		DO On	0	0	0	0	0	0	0	0
ONOFF	config3		DO On	0	0	0	pulse1-5s	0	0	0	0
ONOFF	config5		DO On	1	0	0	1	1	0	0	0
ONOFF	config5		DO Off	0	1	1	0	0	1	1	1
ONOFF	config6		DO On	1	1	1	1	1	1	1	1
ONOFF	config6		DO Off	0	0	0	0	0	0	0	0
ONOFF	config7		DO On	1	0	0	1	1	0	0	0

Figure 53: Full Stop Test for OnOff object

The Field objects of the CODESYS Baseline have passed successfully all the validation tests.

The all different parts of the validation procedure and their steps can be seen in the Appendix I'.

5.3 CODESYS and Process Simulation

CODESYS gives the possibility to run an application on a conventional Windows PC. For SoMachine, the so-called CODESYS Soft-PLC, can be used as a target device for downloading and running an application. SoftPLC is a virtualization of a real PLC, which offers the Runtime environment for the control application. In TwinCAT software, the Soft-PLC option is not present, since Beckhoff is not using the CODESYS Runtime to run an application. Instead, the option of the *Local* target can be chosen in order to locally run the control application.

This way, the developing PC of application, can also take the role of the controller. Of course, since the hardware was not particularly designed for industrial purposes, it has the disadvantage of connecting to industrial hardware, comparing to a PLC. EtherCAT technology could be used in order to connect to the I/O signals of powerful PCs with high real-time performance. Alternatively, the OPC Server or the C/C++ interface could be used.

In PLC section, many control applications are tested before their commissioning, by simulating the real plant. The simulation gives the capability of testing, without the danger of damaging equipment. It is also used by the SCADA operators for training purposes. In these cases a PLC is used for the logic execution and is connected via OPC Server to the simulation of the process. The software that is used for the simulation of the control system, is the *EcoSim Pro*. Libraries for Cryogenics and Cooling control applications have been developed at CERN for the *EcoSim Pro* software, so it is mainly used for this type of applications.

A conventional PC could be used, instead of a PLC for the simulation. It is also convenient, because the PC doesn't have to be connected to real hardware devices, but only interacts with the I/Os through the software. This gives also the possibility, that the PLC program and the simulation can run in the same PC, reducing the cost for resources.

5.3.1 Twincat

A real CPC application, developed for Schneider, was also created for CODESYS and more specifically for TwinCAT, in order to demonstrate the use of a conventional PC, as a PLC. The PC control application was connected via OPC Server to the *EcoSim Pro* simulation. The PC application was also communicating with WinCC OA SCADA via Modbus/TSP protocol.

The application is called QSDN and it is used at Point 6 of the LHC ring. It is composed by two nitrogen storage vessels, which can provide liquid nitrogen to cryogenic plants via two OnOff valves (xPV409). Moreover, each vessel can be filled from a nitrogen truck and the internal pressure

of vessels is regulated by an electrical heater xEH400. Each vessel has also a gas outlet xPV408 to provide warm gaseous nitrogen.

The *EcoSim Pro* simulation of the process is exported in C++ classes that are handled in a Visual Studio C++ project. The project produces an executable file for running the simulation and communicate with the PLC variables through OPC.

The PLC program and the simulation were both running on the same Windows PC.

The steps that were followed for creating and connecting the application with the simulation and the WinCC OA are listed below:

1. Generation of the project with UAB and importation of the generated files from CODESYS plug-ins to the TwinCAT Baseline.
2. Generation of the Simulator_Variables.txt from the *Expert Generator* plug-in. This file is needed for the correspondence of the simulated variables with the variables of the PLC program and is used by the EcoSim Pro simulation.
3. Built and Run of the TwinCAT project.
4. Configuration of the TwinCAT OPC Server (figure 54). For availability of all variables in the OPC server the option 7 should be selected in the *AutoCfg*. In *AutoCfg File* the .tpy file of the project should be specified. This file is updated every time the TwinCAT application is built. The name that is given to the I/O Device should be the same as the PLC name specified in the UAB Wizard, since the name is taken into account in the generation in step 2. The server mode *Inproc(Dll)* or *Out-of-Process(exe)* can be selected from I/O Devices in *OPC Server Settings* Tab. *Exe* is a better option if only one client is going to be connected.
5. Checking of the availability of the variables with the demo OPC client *Softing OPC Toolbox* demo client before connecting with the simulation.
6. Creation of the WinCC OA application and importation of the generated database file from UAB.
7. Run of the simulation from the produced executable file.

In the figure 54 is presented the WinCC OA panel of the application. From there, orders to the PLC can be given and the response from the simulated plant can be viewed. For example, in this picture an order was given to open the valve 1PV409. The Digital Output is giving an *On* signal to an internal variable that is read by the simulation, which sends back

a Digital Input feedback On. This is the expected response of the plant, that is indicated by the full green color of the valve in the panel. Also, the level of this vessel is slowly decreasing, due to the open valve, which is an Analog Input coming again from the simulated controlled equipment.

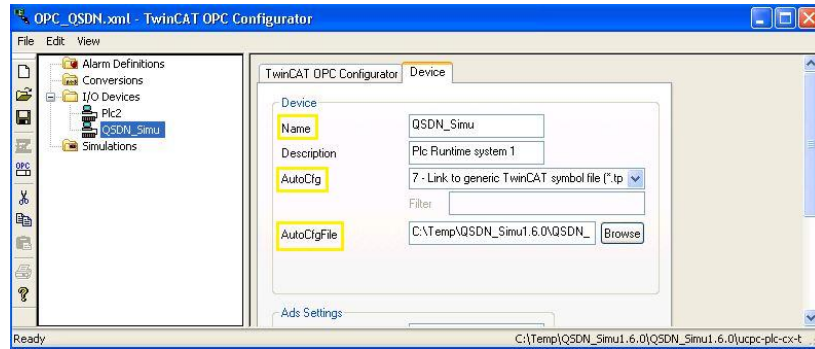


Figure 54: TwinCAT- OPC Configuration

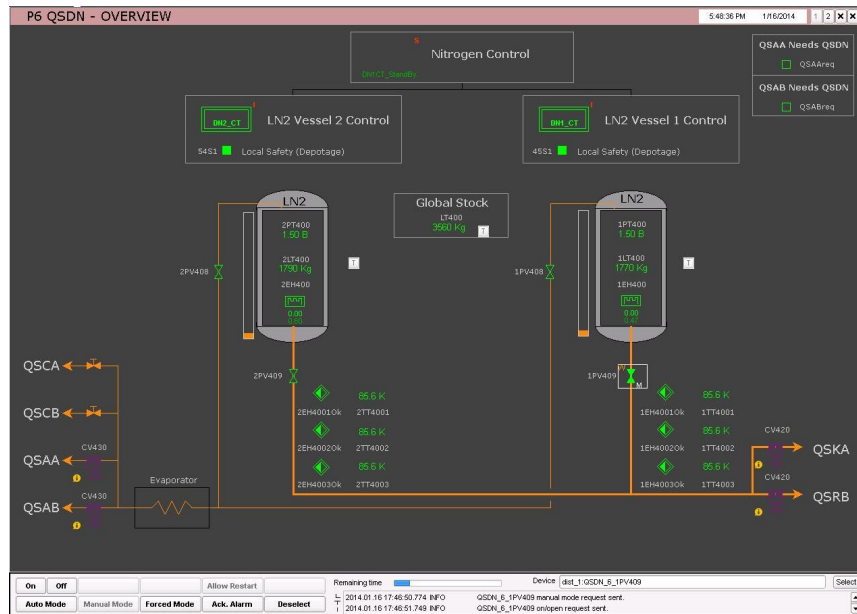


Figure 55: WinCC OA panel of the QSDN application

5.3.2 SoMachine

CODESYS and SoMachine offer a demo SoftPLC, as well as an OPC server for a Windows PC. In this case, the CPC objects Baseline is fully compatible but the Modbus and Time functions that are used for the communication with WinCC OA are not supported. The QSDN project can

be generated and imported to SoMachine software. It can be built and run, but with no connection to the SCADA, which is required for testing the simulated process. SoftPLC demo version is used for all the applications, which do not have hard real-time constraints, as the ones mentioned above. Alternatively, a real-time SoftPLC is offered by CODESYS, but it is not license free and it is mainly aimed on controller manufacturers.

6 Conclusions

6.1 General

An integration of CODESYS-based PLC platforms to UNICOS-CPC framework was performed in this project. The integration was based on IEC 61131-3 and PLCOpen XML standards. Two CODESYS-based platforms, Beckhoff-TwinCAT and Schneider-SoMachine, were used to demonstrate an example of hardware independence, that is offered by this solution.

One Instance and one Logic plug-ins for the UAB software tool are offered to the future control application developers of TwinCAT and SoMachine development environments. The plug-ins automatically generate the PLC code, based on the UNICOS-CPC framework and the IEC 61131-3 standard. Also, a common Resource Package, which contains the Jython Templates, that are used by the UAB plug-ins, is provided. The two plug-ins, that are shared by the two PLC environments, generate the PLC program in PLCOpen XML file format. The program can then be imported to the specific PLC development environment (SoMachine and TwinCAT). The library of CPC objects (Function Blocks) that is provided, is also common for both development tools.

The influence of the hardware, was present in terms of real-time functions and handling of the I/O signals. This influence was taken into account during the development of this project, so that the developer of the control application, can focus only on the programming of the application, regardless of the hardware platform that he is going to use.

6.2 Future possibilities

The development of this project was based on international standards and on the CERN defined framework for industrial applications, UNICOS. This is widening the choices for PLCs, since there are many CODESYS compatible controllers, that could be easily integrated to UNICOS, in the future.

For the existing system some suggestions for continuing the work that has been done, are the following. The performance of the PLC-SCADA communication for the Schneider and Beckhoff controllers, could be tested and improved. Also, the EtherCAT fieldbus functionality could be studied, with different configurations. EtherCAT and I/O configuration in general, could be also generated from UAB, for TwinCAT, by adding related attributes in the Specifications file and creating dedicated Templates.

High-level programming languages, as C/C++, could be introduced in the CPC control applications, since they are supported by CODESYS and also TwinCAT, which is integrated in Visual Studio 2010 and has a dedi-

cated C++ module.

Finally, the use of general-purpose computers with Windows or Linux operating systems, as CODESYS compatible controllers can be also studied in the future, bridging the gap in the industrial and PC worlds.

Appendix

A' CPC Objects

In figures A'.1, A'.2 the 21 CPC object FBs can be seen with their input and output pins.

B' Mapping of addresses

A representation of the mapping of variables in the PLC memory can be seen in figure B'.1:

There are three main register types: Status (Binary/Analog), Event and Request registers. These registers are allocated in three different areas of the PLC and SCADA memory, as shown in figure B'.2. The initial addresses to map the registers are:

Status registers: 1096

Event registers: 2192(2x 1096)

Request registers: 7000

The Status registers can be divided in:

- Binary status of Input/Output objects.
- Binary status of Other Objects (Field Objects, Control Objects..).
- Analog status of Input/Output objects.
- Analog status of Other Objects (Field Objects, Control Objects..).

All the registers are mapped in memory in blocks of 96 words (TCP communications optimization) in the order described in figure B'.3. The I/O Binary Status and OTHER Binary Status registers can be consecutive, so there is no need to have free memory between them. The OTHER Binary Status and I/O Analog Status registers can't be consecutive. If the OTHER Binary Status last register and the I/O Analog Status first register are consecutive, a spare empty block of 96 words has to be inserted between them. The starting address for the I/O Analog Status registers must be: $1096 + n \cdot 96$, where 'n' is the number of tables used to map the I/O Binary Status, OTHER Binary Status registers and the spare block, if necessary. The I/O Analog Status registers and OTHER Analog Status registers can be consecutive, so there is no need to have free memory between them. The OTHER Analog Status block is made of registers of sizes WORD and REAL. If the registers of size REAL have a start address

in odd numbers it may occur that, when the TSPP protocol split the data in several tables, the first part of the REAL is in the table N and the second part is in the table N+1. To avoid this situation the OTHER Analog Status registers are ordered to place first the REAL registers and then the WORD registers. The same principle is used for SCADA mapping.

All the event registers are mapped in memory starting at the address 2192. This is not mandatory for the PLC side but it is used for the SCADA mapping.

The request registers can be divided in:

- Analog request of Input/Output objects.
- Analog request of Other Objects (Field Objects, Control Objects...).
- Binary request of Input/Output objects.
- Binary request of Other Objects (Field Objects, Control Objects...).

These registers are mapped in memory in the order described above, starting usually at the address 7000. To separate the Analog and Binary registers in the TSPP tables, it's necessary to include some free memory between the blocks 2 and 3. The starting address for the Binary request registers of I/O objects must be: $1096 + n \cdot 96$.

All the addresses mentioned above are the recommended addresses and can be changed in UAB Wizard.

Γ' Field Object Test

In figures Γ'.1 - Γ'.4 can be seen the series of the Field Object Tests, their steps and the expected results for each state. The tests were performed for CODESYS Field Objects Baseline.

Δ' PLCOpen XML and CODESYS XML file formats

A part of a PLC program is expressed in PLCOpen XML and in CODESYS XML file formats and is shown in figure Δ'.1, as an example of the difference in the two formats.

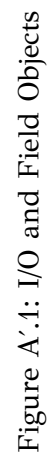


Figure A'1: I/O and Field Objects

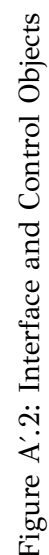


Figure A'2: Interface and Control Objects

CoDeSys	
Description	Address
Command Word Received from DS (I.e Synchronization)	0 5
Big/Little Endian conversion, Real	6 9
Free Area - Reserved	10 69
DWord Received from DS switching between 0 and DS IP coded on 2 Words	70 71
Free Area - Reserved	72 99
Free Area	100 999
1st Status Frame reserved for information exchange between PLC and SCADA	1000 1095
Binary and Analog Status of PLC Objects	1096 6999
Binary and Analog Request from SCADA	7000 9999
Recipe Buffer	ManRegAddr 1000 ManRgeVal 1000 ReqaAddr 1000 ReqVal_Real 2000 ReqVal_UINT 2000
	10000 11000 12000 13000 13000 14999
Free Area	15000

Figure B'.1: PLC variables mapping

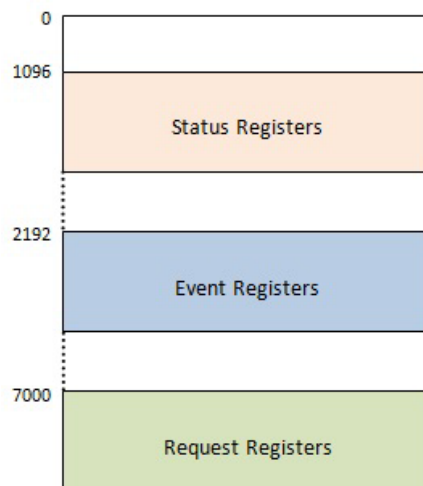


Figure B'.2: Status,Event and Request Registers

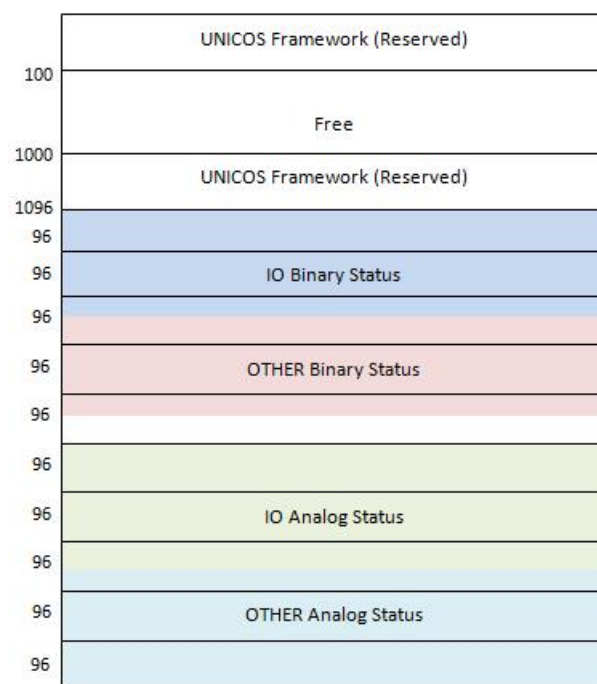


Figure B'.3: Binary and Analog Status Registers

ORDERS		NB: In these tests, feedbacks are kept at FALSE											
Type	Config	Comments	Param	1	2	3	4	5	6	7	8	9	10
INPUTS	LOGIC		AuOnR(OnOff+PCO+ANADO)	1	1	1	1	1	1	0	0	0	1
INPUTS	LOGIC		AuOffR (OnOff+PCO)	0	0	0	0	0	0	0	1	0	0
INPUTS	LOGIC		AuPosR (ANALOG,ANADIG,ANADO)	100	100	100	100	100	100	100	0	0	100
INPUTS	MANUAL		PVSS action (OnOff+PCO+ANADO)	x	Manual	Off	On	Off	Auto	x	x	x	x
INPUTS	MANUAL		PVSS action (ANALOG,ANADIG,ANADO)	x	Manual	MPosR=0	MPosR=100	MPosR=0	Auto	x	x	x	x
ONOFF	config1		DO On	1	1	0	1	0	1	1	0	0	1
ONOFF	config2		DO On	0	0	1	0	1	0	0	1	1	0
ONOFF	config3		DO On	0	0	0	pulse1-5s	0	pulse1-5s	0	0	0	pulse1-5s
ONOFF	config5-7		DO On	1	1	0	1	0	1	1	0	0	1
ONOFF	config5-7		DO Off	0	0	1	0	1	0	0	1	1	0
ONOFF	config9-11		DO On	0	0	0	pulse1-5s	0	pulse1-5s	0	0	0	pulse1-5s
ONOFF	config9-11		DO Off	0	0	pulse1-5s	0	pulse1-5s	0	0	pulse1-5s	0	0
PCO	config7		RunOst	1	1	0	1	0	1	1	0	0	1
ANALOG	config19	FS Off	AO	100	100	0	100	0	100	100	0	0	100
ANALOG	config20	FS On	AO	100	100	0	100	0	100	100	0	0	100
ANALOG	config29	FS Inv	AO	0	0	100	0	100	0	0	100	100	0
ANADIG	config23	FS Off	PosSt	100	100	0	100	0	100	100	0	0	100
ANADO	config9-	FS Off	DO	1	1	0	1	0	1	1	0	0	1
ANADO	config9-	FS Off	AO	100	100	0	100	0	100	100	0	0	100
ANADO	config10-	FS On	DO	0	0	1	0	1	0	0	1	1	0
ANADO	config10-	FS On	AO	0	0	100	0	100	0	0	100	100	0

Figure Γ.1: Test WinCC OA operation orders

FULL STOP Config	Comments	Param	ONLY IF FS DISAPPEARS																						NEW	NEW	NEW	NEW
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22				
LOGIC		AutOn(OnOff+PCO+ANADO)	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0			
LOGIC		AutOff(OnOff+PCO)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0			
LOGIC		AutPost (ANALOG+ANADO)	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0	100	100	100	100			
LOGIC		Interlock	x	x	x	x	FS	x	act+AR	x	FS	x	act+AR	x	x	FS	x	act+AR	x	x	x	x	x	x	act+AR			
MANUAL		PVSS action (OnOff+PCO+ANADO)	x	x	x	x	FS	x	x	x	Off	x	x	x	x	Auto	x	x	x	x	x	x	x	x				
MANUAL		PVSS action (ANALOG+ANADO)	x	x	x	x	FS	x	x	x	MposAd	x	x	x	x	Auto	x	x	x	x	x	x	x	x				
config1		DO On	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0			
config2		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config3		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config4		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config5		DO On	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1			
config6		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config7		DO On	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0			
config8		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config9		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config10		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config11		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config12		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config13		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config14		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config15		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config16		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config17		DO On	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0			
config18		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config19		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config20		DO On	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0	0			
config21		DO On	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0	0			
config22		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config23		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config24		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config25		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config26		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config27		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config28		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config29		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config30		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config31		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config32		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config33		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config34		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config35		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config36		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config37		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config38		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config39		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config40		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config41		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config42		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config43		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config44		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config45		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config46		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config47		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config48		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config49		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config50		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config51		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config52		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config53		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config54		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config55		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config56		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config57		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config58		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config59		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config60		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config61		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config62		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config63		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config64		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0			
config65		DO On	0	0	0	0	0	0	0	0	0	0</																

Type	Temporary Stop Config	Fail Safe	NB: In these tests, feedback Pulse Duration = 5sec										Warning Time Delay = 10sec										NEW	NEW	NEW	NEW			
			Param	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17									
INPUTS	LOGIC		AutOnR(OnOff+PCO+ANA)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0
INPUTS	LOGIC		AutOffR(OnOff+PCO)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
INPUTS	LOGIC		AutPosR (ANALOG, ANADI)	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0	0
INPUTS	LOGIC		Interlock	x	TS	x	x	x	TS	x	x	TS	x	x	TS	x	x	x	x	TS	x	x	TS	x	x	TS	x	TS	x
INPUTS	MANUAL		PVSS action (OnOff+PCO)	x	x	x	x	manual	x	x	x	Off	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
INPUTS	MANUAL		PVSS action (ANALOG, AN	x	x	x	x	manual	x	x	x	MPosR=0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
ONOFF	config1	Off	DO On	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config2	On	DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config3	Off	DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config5	Off	DO On	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config6	On	DO Off	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ONOFF	config6	On	DO On	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ONOFF	config6	Off	DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config7	2DO Off	DO On	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config7	Off	DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config9	Off	DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config9	Off	DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config10	On	DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config10	Off	DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config11	2DO Off	DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ONOFF	config11	Off	DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PCO	config7	Off	RunOst	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANALOG	config19	Off	AO	100	0	100	100	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANALOG	config20	On	AO	100	100	100	100	100	100	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANALOG	config29	Inv	AO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANADIG	config23	Off	PosSt	100	0	100	100	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANADO	config9	Off	DO	1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANADO	config9	Off	AO	100	0	100	100	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANADO	config10	On	DO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ANADO	config10	On	AO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7.3: Test Temporary Stop behavior

START INTERLOCK		NB: In these tests, feedbacks are kept at FALSE/0																		
Type	Config	Comments	Param	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
INPUT/TS	LOGIC		AuOnR(OnOff+PCO+ANADO)	1	1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	
INPUT/TS	LOGIC			0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0
INPUT/TS	LOGIC		AuPosR (ANALOG,ANADIG,ANADO)	100	100	100	0	0	100	100	100	100	0	100	100	100	100	100	100	
INPUT/TS	LOGIC	Interlock		x	SI	x	x	SI	SI	x	SI	SI	SI	x	x	SI	SI	SI	x	
INPUT/TS	MANUAL		PVSS action (OnOff+PCO+ANADO)	x	x	x	x	x	x	x	x	x	x	x	x	x	Manual	Off	On	x
INPUT/TS	MANUAL		PVSS action (ANALOG,ANADIG,ANADO)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	MposR=0	MposR=100	x
ONOFF	config1		DO On	1	1	1	0	0	0	0	0	1	1	0	0	1	1	0	0	1
ONOFF	config2		DO On	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config3		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config5		DO On	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	
ONOFF	config5-		DO Off	0	0	0	1	1	1	0	0	1	1	0	0	0	1	1	0	
ONOFF	config6-		DO On	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	
ONOFF	config6-		DO Off	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config8-		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config9-		DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config10-		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config10-		DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config11-		DO On	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
ONOFF	config11-		DO Off	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
PCO	config7		RunOst	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	
ANALOG	config19	FS Off	AO	100	100	100	0	0	0	100	100	0	0	100	100	100	0	0	100	
ANALOG	config20	FS On	AO	100	100	100	0	0	100	100	100	100	100	100	100	100	100	100	100	
ANALOG	config29	FS Inv	AO	0	0	0	0	100	100	0	0	0	0	0	0	0	0	0	0	
ANADIG	config23	FS Off	Posst	100	100	100	0	0	0	100	100	0	0	100	100	100	0	0	100	
ANADO	config9-	FS Off	DO	1	1	1	0	0	0	1	1	0	0	1	1	1	0	0	1	
ANADO	config9-	FS Off	AO	100	100	100	0	0	0	100	100	0	0	100	100	100	0	0	100	
ANADO	config10-	FS On	DO	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	
ANADO	config10-	FS On	AO	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 1'4: Test Start Interlock behavior

```

1  PROGRAM MAIN
2  VAR
3      counter: INT;
4      Mode: BOOL;
5  END_VAR
6
7  counter:=counter+1;
8
9  IF counter=1000 THEN
10     Mode:=TRUE;
11     counter:=0;
12 ELSE
13     Mode:=FALSE;
14 END_IF

```

(α') Program

```

<pou name="MAIN" pouType="program">
  <interface>
    <localVars>
      <variable name="counter">
        <type>
          <INT />
        </type>
      </variable>
      <variable name="Mode">
        <type>
          <BOOL />
        </type>
      </variable>
    </localVars>
  </interface>
  <body>
    <ST>
      <xhtml xmlns="http://www.w3.org/1999/xhtml">
        counter:=counter+1;
        IF counter=1000 THEN
          Mode:=TRUE;
          counter:=0;
        ELSE
          Mode:=FALSE;
        END_IF
      </xhtml>
    </ST>
  </body>
  <addData />
</pou>

```

(β') PLCOpenXML


```

    <Single Name="Text" Type="string">counter:=counter+1;</Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">4</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string"></Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">5</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string">IF counter=1000 THEN</Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">6</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string">Mode:=TRUE;</Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">7</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string">counter:=0;</Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">8</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string">ELSE</Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">9</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string">Mode:=FALSE;</Single>
  </Single>
  <Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
    <Single Name="Id" Type="long">2</Single>
    <Null Name="Tag" />
    <Single Name="Text" Type="string">END_IF</Single>
  </Single>

```

(γ) CODESYS XML (Code)

```

  <Single Name="Text" Type="string">PROGRAM MAIN</Single>
</Single>
<Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
  <Single Name="Id" Type="long">11</Single>
  <Null Name="Tag" />
  <Single Name="Text" Type="string">VAR</Single>
</Single>
<Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
  <Single Name="Id" Type="long">12</Single>
  <Null Name="Tag" />
  <Single Name="Text" Type="string">counter: INT;</Single>
</Single>
<Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
  <Single Name="Id" Type="long">13</Single>
  <Null Name="Tag" />
  <Single Name="Text" Type="string">Mode: BOOL;</Single>
</Single>
<Single Type="{a5de0b0b-1cb5-4913-ac21-9d70293ec00d}" Method="IArchivable">
  <Single Name="Id" Type="long">14</Single>
  <Null Name="Tag" />
  <Single Name="Text" Type="string">END_VAR</Single>
</Single>

```

(δ) CODESYS XML 2 (Declarations)

Figure A.1: A PLC program in PLCOpen XML and in CODESYS XML file format

References

- [1] 3S-Smart Software Solutions GmbH. CODESYS–industrial IEC 61131-3 PLC programming. [Online]. Available: <http://www.codesys.com>
- [2] Acromag Inc. Introduction to the two-wire transmitter and the 4-20mA current loop. White Paper. [Online]. Available: http://www.acromag.com/sites/default/files/Acromag_Intro_TwoWire_Transmitters_4_20mA_Current_Loop_904A.pdf
- [3] B. Bradu, “UNICOS – CPC: Basic Functionalities,” CERN Internal Paper.
- [4] J. Brahy, “TSPP UNICOS Manager,” Geneva 2005, CERN Internal Paper.
- [5] B. Copy, R. Barillere, E. Blanco, B. F. Adiego, R. N. Fernandes, and I. P. Barreiro, “Model oriented application generation for industrial control systems,” in *ICALEPCS11, Grenoble, France, 2011*.
- [6] M. Dutour, “Software Factory techniques applied to Process Control at CERN,” in *ICALEPCS07, Knoxville, Tennessee, USA, 2007*.
- [7] P. Gayet and R. Barillère, “UNICOS a framework to build industry like control systems: Principles & methodology,” in *ICALEPCS05, Genève, Suisse, 2005*.
- [8] *Programmable Controllers, Part 3: Programming Languages*, IEC International Standard IEC 61131-3 Std., 2003.
- [9] H. Milcent, E. B. Viñuela, F. Bernard, and P. Gayet, “UNICOS: An open framework,” in *ICALEPCS09, Kobe, Japan, 2009*.
- [10] A. Otto and K. Hellmann, “IEC 61131: A general overview and emerging trends,” *Industrial Electronics Magazine, IEEE*, vol. 3, pp. 27 – 31, 2009.
- [11] E. Parr, *Programmable Controllers: An engineer’s guide*. Newnes, 2003.
- [12] T. Petrou, “CoDeSys Implementation to UNICOS CPC6,” Master’s thesis, TEI Piraeus, Faculty of Technological Applications, Automation Department, 2012.
- [13] PLCOpen. Welcome to PLCOpen. [Online]. Available: <http://www.plcopen.org/>
- [14] PLCOpen Technical Committee 6, “XML Formats for IEC 61131-3,” PLCOpen Organization, Tech. Rep., 2009.

-
- [15] E. B. Viñuela, J. Beckers, B. Bradu, P. Durand, B. F. Adiego, S. I. Rosas, A. Merezhin, J. O. Vidal, J. Rochez, and D. Willeman, “UNICOS Evolution: CPC Version 6,” in *ICALEPCS11, Grenoble, France, 2011*.
 - [16] E. B. Viñuela, M. Koutli, T. Petrou, and J. Rochez, “Opening the floor to PLCs and IPCs: CoDeSys in UNICOS,” in *ICALEPCS13, San Francisco, USA, 2013*.
 - [17] Wikipedia, the free encyclopedia. D-subminiature. [Online]. Available: <http://en.wikipedia.org/wiki/D-subminiature>