Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Advanced PLC programming methods

Miroslav Hanák

Supervisor: Ing. Petr Kadera, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2017

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Miroslav H a n á k
Study programme:	Cybernetics and Robotics
Specialisation:	Robotics
Title of Bachelor Project:	Advanced PLC Programming Methods

Guidelines:

1. Study the PLCopen XML standard.
2. Study the Model-driven PLC programming approach.
3. Develop a method based on object-oriented programming principles for generating PLCopen XML compliant code from reusable templates.
4. Implement a tool set for practical evaluation of the method from the step 3.

Bibliography/Sources:

[1] Blewitt Alex - Eclipse 4 Plug-in Development by Example Beginner's Guide - Birmingham, 2013
[2] Blewitt Alex - Mastering Eclipse Plug-in Development - Birmingham, 2014
[3] Bolton William - Programmable logic controllers - Oxford, 2009

Bachelor Project Supervisor: Ing. Petr Kadera, Ph.D.

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2017

Acknowledgements

I would like to thank my family for all their support through my studies. Also, I would like to thank my supervisor for offering an interesting thesis topic and for his advice.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....

signature ....

Abstract

The main goal of this thesis is to use Model-Driven Design approach and Object-Oriented Programming features to create a software tool for programming PLC control system.

The primary function of the instrument of development is the automated generating of proper PLC code according to IEC 61131-3 from templates which represent models of real components of automated plant and let us use some of the Object-Oriented Programming features.

PLCopen XML standard based on extended markup language technology, is used, as it represents a vendor and hardware platform independent way for storing generated PLC code.

The development tool is implemented in Java language with using Eclipse 4 Rich Client Platform development framework.

Keywords: PLC, Model-Driven Desing, Object-Oriented Programming, PLCopen XML, Java, Eclipse 4 Rich Client Platform

Supervisor: Ing. Petr Kadera, Ph.D.
CTU in Prague, CIIRC
Jugoslávských partyzánů 1580/3
160 00 Prague 6
Czech Republic

Abstrakt

Hlavním cílem této práce je využít přístup Modelově řízeného vývoje a Objektově orientovaného programování k vytvoření softwarového nástroje určeného k programování PLC.

Hlavní funkcí vývojového nástroje je generování PLC kódu dle normy IEC 61131-3 z šablon, které představují reálné komponenty automatizované soustavy a dovolí nám využít vlastnosti Objektově orientovaného programování.

Standard PLCopen XML, založený na technologii rozšiřitelného značkovacího jazyka, je použit jako na platformě a výrobci nezávislý způsob uchování generovaného PLC kódu.

Vývojový nástroj je implementován v jazyce Java s využitím vývojového rámce Eclipse 4 Rich Client Platform.

Klíčová slova: PLC, modelově řízený návrh, objektově orientované programování, PLCopen XML, Java, Eclipse 4 Rich Client Platform

Překlad názvu: Pokročilé metody programování PLC

Contents

Figures

Tables

Chapter 1
Introduction

1.1 Motivation

Throughout a development process of control systems, control engineers deal with challenges like shorter development time, higher quality and flexibility requirements and reusability of the control code. Since existing technologies and approaches are limited by their effectiveness, new approaches are needed [2].

A programmable logic controller (PLC) is a digital computer designed to resist hard conditions of the industrial environment. It has been an integral part of factory automation and industrial process control for decades. They control a wide sort of applications from simple lighting functions to chemical processing plants.

The IEC 61131-3 standard defines a software model and provides a set of programming languages for the development of PLC control systems. It is widely accepted in the industrial automation domain, and most of the commercial tool vendors proclaim compliance with it [1].

Last years the standard received criticism for its noncompliance with state of the art software engineering trends and concepts such as object-oriented programming, distribution of the application software and the model-driven development paradigm [3]. Several research groups work in this direction [1].

1.2 Aim

The primary focus of the thesis is to create a PLC software development tool using Model-Driven development approach.

The secondary aim of this thesis is to introduce the state-of-the-art PLC software development methods briefly.

1.3 Thesis structure

In the next chapter, we introduce PLC and International Electrotechnical Commission standard for PLCs 61131 especially part 3. Also, the PLCopen and XML concepts will be presented. Chapter 3 gives us a brief overview and explanation of state-of-the-art PLC programming methods including Model-Driven development. Chapter 4 focuses on the development tool design. In Chapter 5 the implementation of the instrument of development is covered. In Chapter 6 the evaluation of the development process and the implemented development instrument is done. In the last chapter, we discuss reached results.

Chapter 2

PLC and its control code

2.1 Programable logic controller

A programmable logic controller (PLC) is a digital computer applied for automation of electromechanical processes. Almost any production line, machine function, or process can be greatly enhanced using this type of control system. It continuously monitors the state of input devices and makes decisions based upon a custom program to control the state of output devices.

PLCs are demanded to work flawlessly for years in industrial environments that are hazardous to the electronic components that modern PLCs are made from. PLCs must be robust and designed for immunity to electrical noise, resistance to vibration, impact, extended temperature ranges and moisture.

A PLC is a hard real-time system. Output must be produced in response to input conditions within a limited time. Otherwise, an unintended operation will result. It can cause many damages [18, 20].

2.1.1 Formation

PLC invention was in response to the requirements of the American automotive manufacturing industry. Before the PLC, control, sequencing, and safety interlock logic for manufacturing lines relied on hundreds or, in some cases, thousands of relays, cam timers, and drum sequencers and dedicated closed-loop controllers.

Relay systems at the time tended to fail and create delays. Engineers then had to troubleshoot a whole wall of relays to find and fix the problem. Also, a process of updating such facilities was very time to consume and expensive, as technicians needed to separately and manually rewire every relay.

The purpose of a PLC was to directly replace electromechanical parts as logic elements, substituting it by a solid-state digital computer with a saved program, able to imitate the interconnection of many relays to perform several logical tasks. In the 1960s and the 1970s, with the discovery of the microprocessor, the device that was first used as a relay replacement device only, evolved into the advanced PLC of today [18, 20, 21].

2.1.2 Hardware

The basic PLC must be adequately flexible and configurable to meet the diverse needs of different applications. All PLCs have the same essential components. These elements work together to bring input information into the PLC from the plant, process that information, and send output information back out to a plant.

The components are input and output modules (I/O), central processing unit (CPU or processor), co-processor modules, power supply and peripheral devices [18].

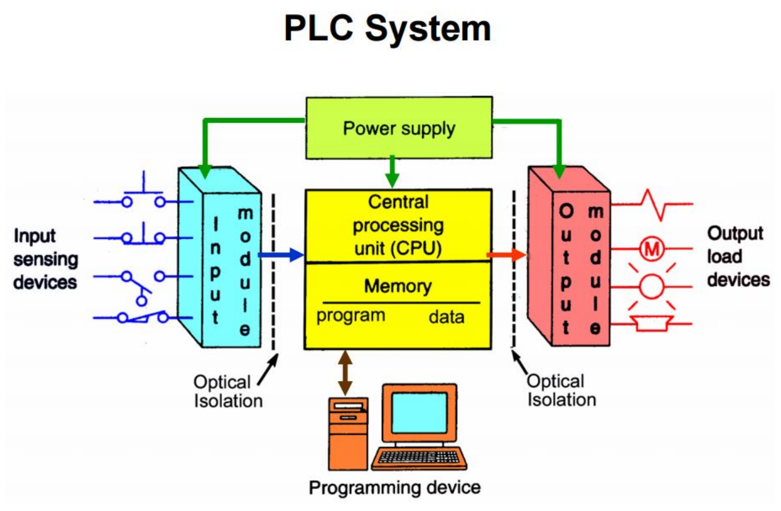


Figure 2.1: PLC hardware depiction from [20]

CPU

A CPU, working as the brain of the PLC, is a microprocessor consisting of a memory and integrated circuits performing control logic, monitoring and communicating.

The CPU executes control instructions stored in a user's programs, communicates with other devices, carries out logic and arithmetic operations, and performs housekeeping activities such as communications, internal diagnostics. It runs memory routines, continually checking the PLC to avoid programming errors and secure the memory is undamaged [20].

Memory

Memory provides storage to the operating system and firmware of the processor and modules, and the program and data. The program memory consists of the lists of instructions, which are sent to the processor. The program is downloaded to a PLCs program memory by a programming device, computer or console. The data memory stores the input and output image tables as well as numerical and boolean data [17, 20].

Peripherals

PLCs read signals from and write signals to different devices through I/O modules. The I/O modules can be single I/O cards or complex decentralised I/O peripherals. The input devices can be sensors, keyboards, and switches. Output devices can be power lights, solenoids, contactors, small motors, pneumatic or hydraulic cylinders.

PLC can either communicate with different intelligent devices like robots controllers, frequency inverters, locks, cameras, database servers, etc. PLCs cannot interact with human well, so external monitors and HMI (Human Machine Interface) operator panels are required. All these devices communicate through various types od field buses [21].

2.1.3 Scan cycle

When PLC is set in run mode, it executes an initialization step. If there are no problems, then the PLC repeatedly executes scan cycle sequence. Scan cycle usually takes a few milliseconds. Scan cycle consists of four main steps [16, 19].

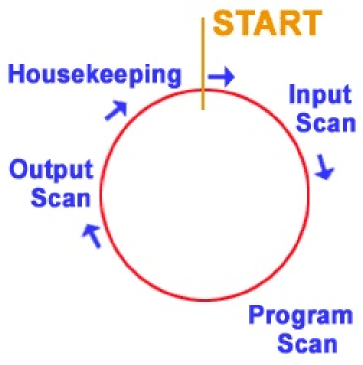


Figure 2.2: PLC scan cycle depiction from [19]

Input scan

The status of all input modules is copied to the memory area called input image table. It avoids cases where an input changes from the start to the end of the program.

Program scan

Data in input image memory area is applied to the user program. The user program is performed, and output image memory area is updated.

Output scan

Data, taken from the output image area, are sent to all output modules in the system.

Housekeeping

There are also other steps like systems checks and updating the current internal counter and timer values [16, 19].

2.2 IEC 61131

In the past, many of various vendors developed PLCs with different run-time, operating systems, and programming languages. To enhance compatibility and interoperability among the various products, the International Electrotechnical Commission (IEC) elaborated the standard IEC 61131. Part 3 of this standard specifies the syntax and semantics of a unified suite of programming languages for PLCs [4, 5].

2.2.1 Software model

Configuration

At the highest level, the software of the particular PLC control application is enclosed in a configuration. It consists of several resources and it groups them together within the PLC control system. It also provides means for data exchange between them.

Resource

A resource is central processing units (CPUs), able to execute assigned task or several tasks. Configurations and resources can be started and stopped via the operator interface or operating system functions.

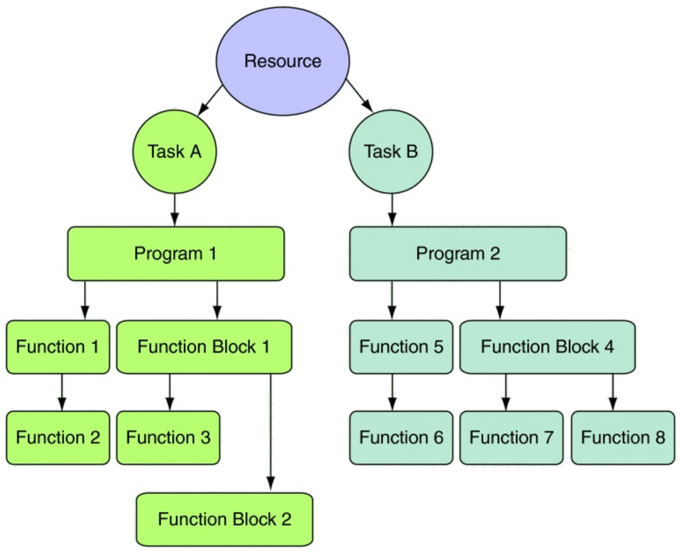


Figure 2.3: Resource depiction from [5](modified)

Task

The task is performed according to its specifications and priority, usually cyclically with cycle times from few to several hundred milliseconds or it's an event triggered. The task call one or several assigned POUs.

Program Organisation Units

Program Organisation Units (POUs) defined by the IEC 61131-3 are Function, Function Block and Program. A POU contains a declaration part and body.

Each declaration of a POU contains at least one declaration part specifying the types and the physical or logical address of the variables used in it.

The body of POU defines POUs algorithmic behaviour in one of the five languages specified by the standard: ST (Structured Text), IL (Instruction List) and SFC (Sequential Function Chart), FBD (Function Block Diagram), LD (Ladder Diagram).

A PLC program is the composition of all the programming language elements and constructs, functions(without internal memory) and function blocks (with internal memory) [5, 22].

2.2.2 Programming languages

There are two textual languages: Instruction List (IL) and Structured Text (ST) and two graphical programming languages: Ladder Diagram, Function Blocks. There is also fifth programming language called Sequential Function Chart (SFC), it can be assumed to be a graphical language.

Instruction List

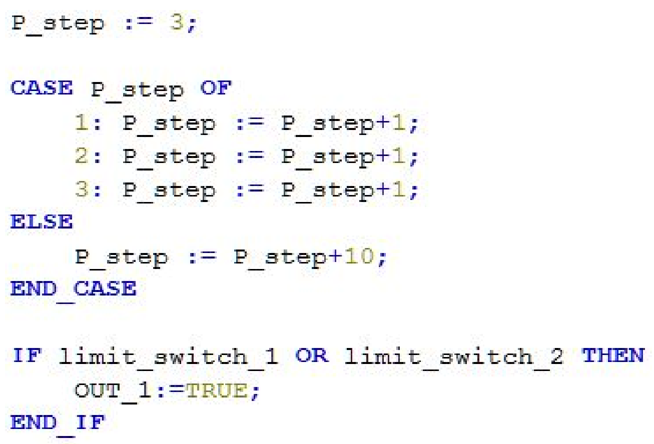
Instruction List is similar to low-level assembly language. A POU consists of the sequence of instructions. Each instruction is situated on separated line. Instructions can be modified by a set of modifiers e.g., negation, conditionality, jumpings, returns and priority.

```
LD var1
AND var2
ANDN var3
OR var4
ST var5
```

Figure 2.4: Instruction List language depiction

Structured Text

Structured Texts syntax seems like a high-level procedural programming language (C or PASCAL). It is defined by commands and expressions. The semicolon divides commands. There can be more commands on the same line.



```
P_step := 3;

CASE P_step OF
    1: P_step := P_step+1;
    2: P_step := P_step+1;
    3: P_step := P_step+1;
ELSE
    P_step := P_step+10;
END_CASE

IF limit_switch_1 OR limit_switch_2 THEN
    OUT_1:=TRUE;
END_IF
```

Figure 2.5: Structured Text language depiction

Ladder Diagram

Ladder Diagram also known as Ladder Logic is the oldest programming language for PLCs. It is based on a graphic representation of relay contact logic, so it is well suited to express the Boolean circuits. Elemnts of Function Block Diagram language are allowed too.

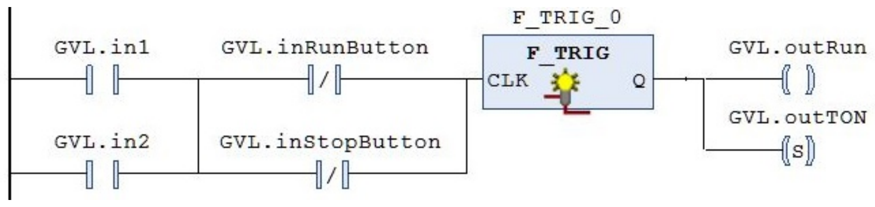


Figure 2.6: Ladder Diagram language depiction

Function Block Diagram

Function Block language is analogous to the electrical and block diagrams of the analogue and digital technique. There are included standard logical functions, timers, counters, communication functions and special functions.

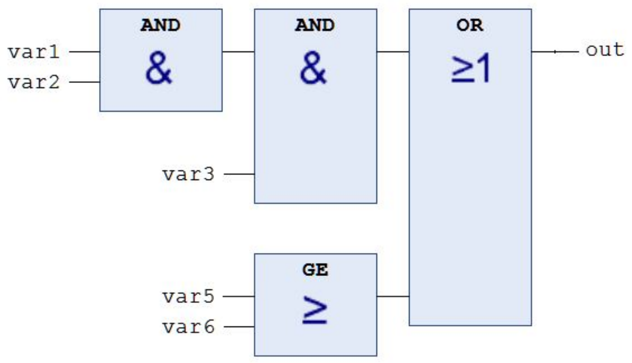


Figure 2.7: Function Block Diagram language depiction

Sequential Function Chart

SFC is derived from Petri Nets. It consists of steps, transitions and actions. The step represents a state of a controlled system and has assigned action block. Transition express condition to deactivate the previous step and activate next step. These are drawn graphically to describe a sequence of interactions [23, 24].

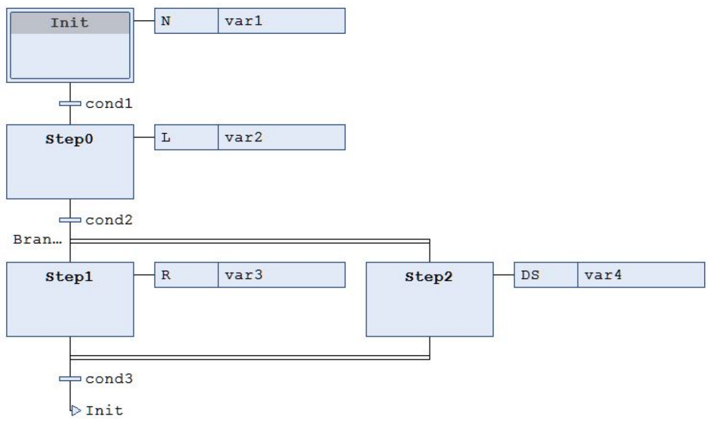


Figure 2.8: Sequential Function Chart language depiction

2.3 PLCopen XML

2.3.1 Purpose

Since the release of the IEC 61131-3, a lot of development environments, used for editing PLC control code according to the standard, have been created by a broad sort of vendors. Users - software developers have wanted to be able to exchange their programs, libraries and projects between these development environments. The IEC 61131-3 compliant programming tools are only one part of complex set of development instruments. The another parts are network, debug and documentation tools and simulators [25].

2.3.2 Introduction

PLCopen, an independent organisation, had decided to develop interfaces towards all these tools. It means interfaces between producers and consumers of logical and graphical information. The primary goal is still to transfer a control project according to IEC 61131-3 standard without much additional work, from one development tool to another without loosing information even if it's incomplete or with errors. The design of the transferred software project unit has to remain the same after the transfer [25].

It allows migration of a software project between different hardware platforms. It causes the losing of the information about hardware. So after the migration of a software project to another vendor development environment

- another hardware platform the hardware configuration with appropriate hardware is needed [25].

Extensible markup language (XML) is the right technology for this task. From the moment that this format is available, it is just more than a transfer tool. XML file can be generated by other tools like simulation and modelling tools and consumed by verification, documentation, and version control tools. [25]

2.3.3 Testing

We tested portability of software project with help development environment CODESYS V3.5 SP9 Patch 5 [26] and e!COCKPIT [27]. We ported project with basic fragments of programs in ladder diagram language from one development environment to another development environment and otherwise to check the PLCopen XML function. We also searched for another development environments which support function of importing/exporting project using by PLCopen XML. Only TwinCAT 3 - XAE [28] was found.

2.3.4 Extensible markup language

XML is a markup language similar to HTML (Hypertext Markup Language), designed to store data in the plain text format and to be both human- and machine-readable. It provides a software- and hardware-independent method of storing, transporting, and sharing data [29, 30].

Structure

The XML document is formed as a tree. The tree starts at a root element and branches from the root to child elements. There can be empty elements or elements can include text. XML elements can have attributes, just like HTML. Attributes serve to contain data related to a specific element [30].

Syntax

An XML document with correct syntax is called "Well Formed". The syntax rules are very simple, logical, easy to learn and easy to use. Each element is defined by a beginning and an ending tag. XML tags are case sensitive. The elements must be properly nested. The attribute values must always be quoted [30].

```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!
</body>
</note>
```

Figure 2.9: XML file syntax depiction from [30]

Design

The tags and attributes are designed by the author of the XML document. So the author defines the document structure. Element names and their attributes describe the content of the document's data, and the tree structure describes the relationship between the data [30].

Schema

The data can be checked for consistency with the provided scheme. Different schemes provide a possibility to check the various incompatibilities. An XML document validated toward an XML Schema is both "Well Formed" and "Valid". A schema is defined as a formal specification of the element and attribute names, that indicates which of those are allowed in an XML document, and in what combinations. It also determines the structure of the document: which elements are child elements of others, the order in which the child elements can be, and the number of child elements. XML Schemas are itself written in XML. [30]

2.3.5 File structure

The following figure depicts the basic structure of PLCopen XML file. The project element is root.

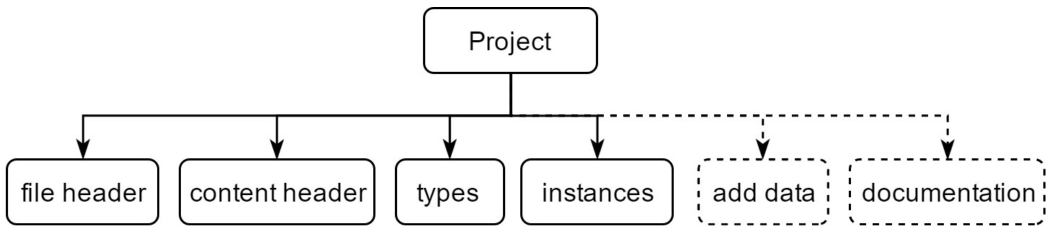


Figure 2.10: PLCopen XML file structure

The file consists of a mandatory and optional elements.

File header

The file header element provides information regarding the creation of the file. Its obligatory attributes are the company name, with a product name, release information, version, and the date and time of the file creation.

Content header

The content header element gives overview information concerning the actual content of the file. The only name attribute is required [25].

Types

The types element gives us an overview of user data types derived from defined data types according to the standard. Also, POUs declaration section and code for both the graphical and textual languages is included [25].

Instances

The instances element can contain a configurations element. The configuration element represents a group of resources. The resource element describes a group of programs, tasks and global variables.

The task element depicts a periodic or triggered task, defined by a required priority and an optional interval time, and consists of a group of program and/or function block instances.

The pouInstance element represents a program or function block instance either running with or without a task [25].

Add data

The vendor can include additional data in the XML file. These optional vendor-specific data are certain in the add data object and must be uniquely identifiable by URI (uniform resource identifier) in an addDataInfo element [25].

Chapter 3

State-of-the-art PLC programming methods

3.1 Model-Driven development

The first model in Model Driven Development (MDD) process is the platform-independent model (PIM). A PIM describes the software in a higher level of abstraction that is independent of any implementation technology. The main idea is to use a model transformation language (MTL) to transform a Platform-independent model into a Platform-specific model. The platform-specific model (PSM) is linked to a specific platform and implementation technology [3].

MDD is a quite successful paradigm in general-purpose computing. This leads to the exploiting the benefits of MDD in the industrial automation domain. It uses the application-centric approach, which is not much supported yet, instead of a classic device-centric approach. The key point in MDD is that models have become the primary artefact of software design and have increased the level of abstraction in the development process. Models are next used for the automatic or semi-automatic generation of program code. [1, 3].

3.1.1 Conclusion

MDD development of PLC code is based on the IEC 61131, its extensions and IEC 61499. It utilises piping and instrumentation diagrams (P&IDs) and modelling languages (e.g. UML, SysML) to generating PLC code [1, 3, 6].

3.2 Object Oriented extension of IEC 61131

The IEC 61131 standard, first published in 1992 is widely accepted and has been used for many years in the industrial automation domain. This is faced with several challenges today's complex systems. After a long period of success and many commercial implementations of the standard, it is the time for new features to address these challenges.

Object-oriented programming (OOP) is conventional in desktop application development and an integral part of technical university education. It's great in handling with complex software development tasks, producing flexible, reusable software components and reducing the development time of new software.

To meet the demands of modern industrial automation, the extension of the IEC 61131-3 that supports the Object Oriented paradigm was considered as the most promising and successful solution. In 2013 the third edition of the IEC 61131-3 was established. It brought significant technical changes, mainly the mentioned Object Oriented (OO) features of classes and function blocks. The approach used in the OO extension is a mix of procedural and OO paradigms e.g. like in C++ [4, 7, 31].

The standard already covers a simple OO class concept, the function block. A function block has an internal state, a routine manipulating this state, and can be instantiated. So, the enlargement of the current function block by object-oriented features is a natural way of adding these features to the standard. Primary points of the OO extension are:

- INTERFACE support
- extending the declaration of the FB type with METHODS
- defining the CLASS construct
- inheritance support with the EXTENDS keyword.

The new OO features are more detailed described and discussed in [4, 7], of course, the exact description can be found in the standard release.

3.2.1 Conclusion

The third edition of IEC 61131-3 introduced important changes to support the new object-oriented syntax and now it is starting to be adopted by some PLC vendors. However, there are some inconsistencies between the proposed features and the formal specification of those features [8].

These differences may become sources of different interpretations, and hence different implementations of the standard, which may lead to non-portability of code, which is contrary to the purposes of the norm. The main source of the inconsistencies is the support both the procedural as well as the object-oriented paradigms [4].

Solutions have been suggested for all the identified issues with the objective to have a more robust extension that is required to avoid problematic implementations of the new version of the IEC 61131 [4].

3.3 IEC 61149

IEC 61499 is based on the function block model of currently dominating standard IEC 61131. It meets the current industry requirements for intelligent, portable, reliable, distributed, flexible automation systems. It shifts from the cyclic scan execution to the event-driven execution, as well it shifts from the device-centric approach to the application-centric approach. It assumes basic concepts of object-oriented approach, and it defines a generic architectural model for the distributed automation applications [10, 11].

3.3.1 Architecture

The standard defines architecture consisting of application, system, device, resource, and the FB model - that allows the development of distributed control applications in an intuitive and graphical manner [9].

3.3.2 Application

An application is described as an aggregation of interconnected FBs instances. Applications are in general composed without any device or infrastructure in mind. After the modelling process, they are mapped to devices, which can execute them [9].

3.3.3 Device

A device represents any control equipment, capable hold resources and execute applications. It consists of a communication interface, providing communication services for the device and the application parts mapped to this device, a process interface, providing the services for communication with sensors/actuators monitoring and controlling the process, the device management, and can contain zero or more resources. In diversity to the resources defined in the IEC 61131-3, IEC 61499 not necessarily associates a resource to one computational unit (e. g. one CPU) [9].

3.3.4 Resource

A resource is a containment unit for applications or application parts living on the specific device, and it has independent control of applications operations. Within a device, resources can be created, deleted, configured, etc [9].

3.3.5 Function Block

An FB is the elemental functional unit for composing IEC 61499 applications. It specifies three main FB types [9].

Basic Function Block

The Basic Function Block (FB) is determined as an entity to encapsulate algorithms and the data flow that these algorithms run on. The IEC 61499 FB enters the target state, executes the associated algorithms, and outcomes the corresponding events.

FB consists of a head and a body. The head, connected to the event flows, accepts event inputs and generates event outputs. The head of the FB type is used to capture dynamics in terms of ECC. The body, connected to the data flows, capturing the functionality terms of algorithms accepts data inputs and generates data outputs. The sequencing of algorithm invocations is defined using a statechart called execution control chart (ECC). An ECC consists of EC states, EC transitions and EC action[10, 11].

Composite Function Block

Composite FB (CFB) contains a BFB network similar to a resource. Incoming events are passed on to the internal FBs [9, 32].

Service Interface Function Block

Service Interface FB (SIFB) provides a mapping from device specific services to IEC 61499 FBs. It represents the interface to low-level services provided by the operating system or hardware of the embedded device. In general, there are two different types of SIFBs: the requester and the responder type[9, 32].

3.3.6 Conclusion

IEC 61499 has been formed to enable intelligent automation. Intelligence is decentralised and embedded into software components, which can be openly distributed over network devices [12]. With a recent increasing emergence of commercially supported, freeware and open source software tools (the overview can be found in [13]) and dozens of hardware platforms, IEC 61499 is getting adopted from academic and research sphere to industry domain. To meet the open system requirements on portability, reconfigurability, and interoperability the using of Software tools requirements (defined in Part 2 of the standard) and Compliance Profiles (defined in Part 4 of the norm) is indispensable [12, 13].

The number and the complexity of the example applications and examinations are not sufficient to demonstrate the maturity of the new technology. The high number of publications, studies, reviews with many contradicting remarks on the standard FB models ambiguities and open problems make very difficult to clearly understand the technology, identify its advantages and it complicates the process of the standard assimilation by industry [10].

Also, the standard efforts to introduce at the same time two paradigm shifts, the one from the procedural to the object-based, and the other from the device-centric to the application-centric, makes the adoption procces slower [10].

Chapter 4

Development instrument design

The main function of the development tool is an automated generating of proper PLC code according to the IEC 61131-3 from reusable templates.

4.1 Task description

In general, the point is to connect and bind the control system with its control code. The goal is the use application-centric approach, instead of classical device-centric approach. So finally we assign the control systems hardware to applications components, representing the control code instead of standard procedure when a user first creates a hardware configuration and then write code for each controller in hardware configuration. We need to represent the control system (CPUs), which executes the control code and components of the automated plant - application and its behaviour.

4.2 Control system

Although using PLCopen XML, we lost information about particular hardware configuration some minimal control system description is necessary. The software model within IEC 61131 includes resources, representing the CPUs of given configuration. The control system consists of a set of controllers (CPUs) which execute the control code. CPUs communicate among themselves by some field bus type.

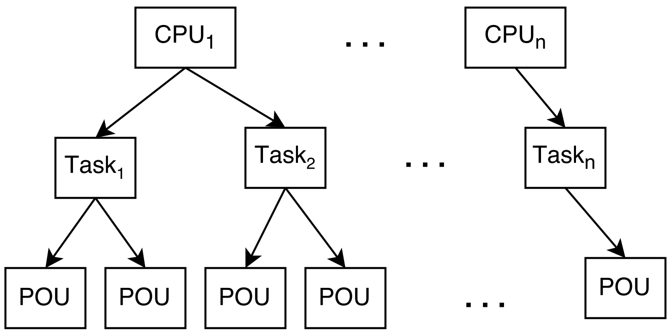


Figure 4.1: Control system view

4.3 Application

The given application (automated plant, e.g. a manufacturing line) consists of a hierarchy of components (e.g. belt-conveyors, rotating tables, machining stations and many others). Each component can include subcomponents (e.g. belt conveyor consists of motors).

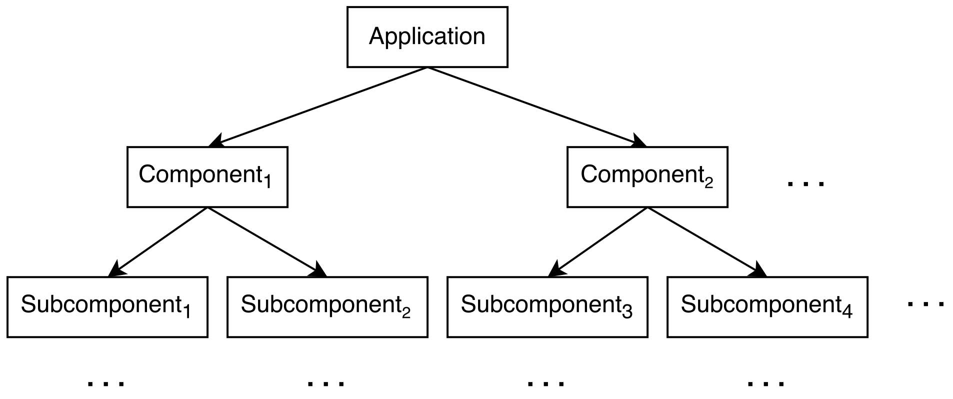


Figure 4.2: Application view

Control code of each component defines its behaviour. We describe the components and hierarchy in software (PLC control code) using object-oriented design.

4.4 Templates

The application is described by a hierarchy of components. To represent the components we use universal Object-Oriented templates. A template represents a general type of a component with general properties and a behaviour e.g. belt-conveyor with n motors.

To express the behaviour of general OO templates we need to enrich proper PLC programming language according to IEC 61131-3 with OO features. A major issue is that the control code cannot be dynamically created during its execution by a CPU of a PLC. All the instances have to be created before downloading into the PLC. This complicates the using of the template.

For example, we need to iterate over a set of instances although we do not know how many of them will be created. Attributes of the templates instances are stored in PLC tags, each of them must have a unique name within the PLC. Again, the names of instances aren't known at design time. To solve these issues we use some enhancements defined in IEC 61131 taken from [14].

Indirect references

It allows the access to attributes-variables of the template . Character $ represents the notation "this" known from OO programming which represents the current object - class instance.

Within enhanced LD code it allows expressing general and not fully named variable e.g. $.run in a belt-conveyor template. "this " construct is later replaced by the name of concrete component - template "instance".

Containment

It allows a user to specify that component has the sub-component(s) e.g. conveyor contains motors. Dot notation realises the referencing to the sub-components attribute e.g. motor.run.

It allows us to access to run variable in subcomponent motor. Hat notation realises the referencing to the parents component attribute e.g. ^.ready means ready variable in parent subcomponent.

Macro instructions

It gives a user the ability to specify basic operations like AND, OR over the set of components either the number of components an their names aren't known yet. The macro instructions operator consists of a type of operation (e.g. AND, OR), a definition of collection type (e.g. motor, cylinder) and set of subinstructions containing character # which is step by step substituted by each element of the collection. It is very useful for collecting general information from the same type of components although the elements of the collection are unknown.

4.5 PLC code creation

The process of the creation of PLC code starts with the creating of the object-oriented template representing general functionality of selected device in the automated field. Then the user creates a concrete application, which consists of a hierarchy of components. Specific components and their subcomponents are created ("instatiated") from object-oriented templates. At this point, all components of the application are known. Now software developer defines CPU representing the minimal hardware configuration. Then he or she assigns using the editor a CPU which will execute the components control code to each component.

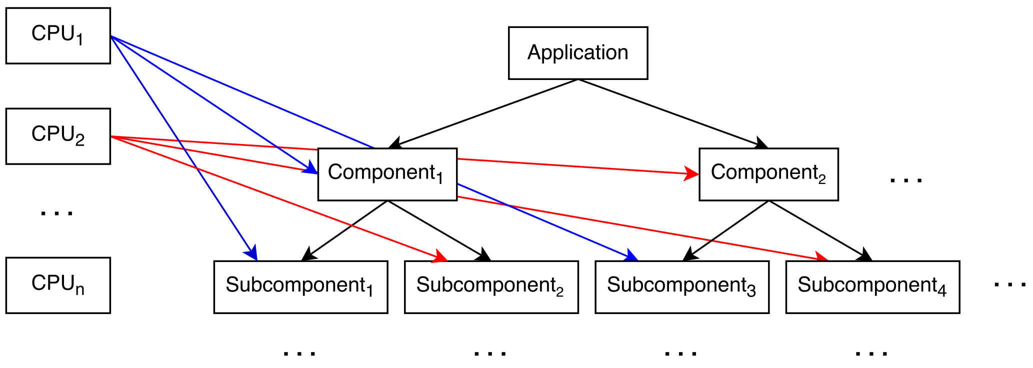


Figure 4.3: CPU assignment

In the editor for the code generation the developer can select for which CPUs the code will be generated. For the chosen CPUs will be generated proper PLC code according to IEC 61131-3 represented by PLCopen XML file.

During the process of generation, all references and object-oriented notation mentioned above are substituted by specific PLC tag which is unique within the PLC. To simplify and test the process of the generation of the control code during the development, the control code will have some restrictions. PLC code will be written only in Ladder diagram language. The generated control code of each PLC will be represented by single POU type Program. All variables all will be stored in one global variable list. For each controller will be generated one global variable list.

4.6 Code deployment

Using the Object-Oriented templates and then the automated generating of the code can simplify the process of the development of the PLC control code.

The advantage of PLCopen XML is its vendor and hardware platform independency but on the other hand it causes the losing of the information about hardware. So after the migration of a software project to another vendor development environment - another hardware platform the hardware configuration with appropriate hardware is needed.

After the importing project to the vendor-specific development environment and then the exact hardware configuration, we are allowed to map the tags to real hardware I/Os. The code of different components can be executed in different CPUs. In that situation, maybe it will be necessary to set tag sharing between these CPUs. Its solution depends on a vendor of the PLC control system - on a particular hardware platform.

4.7 Tools

To ensure all needed functionalities, we divide development instrument to different tools.

- Control system editor - A control system is represented by the CPUs executing the control code. User creates CPUs and edit its properties.
- Template editor - User creates a reusable templetes representing a different types of facilities.
- Application editor - User creates a specific application from templates of facilities.
- Assignment editor - User assigns CPU to each component. Component's code will be executed in an assigned CPU.
- Generation editor - User can choose the CPU for which the PLC code will be generated. The PLC code has to be generated for an each CPU of the control system.

4.8 Example

For better illustrativeness and explanation, we introduce the whole process by example. We are going to use all enhancements introduced above. Let us consider that there is a simple production line consisting of pneumatic cylinders, their limit switches, part presence switches, control desks and belt conveyor. Belt conveyor consists of motors.

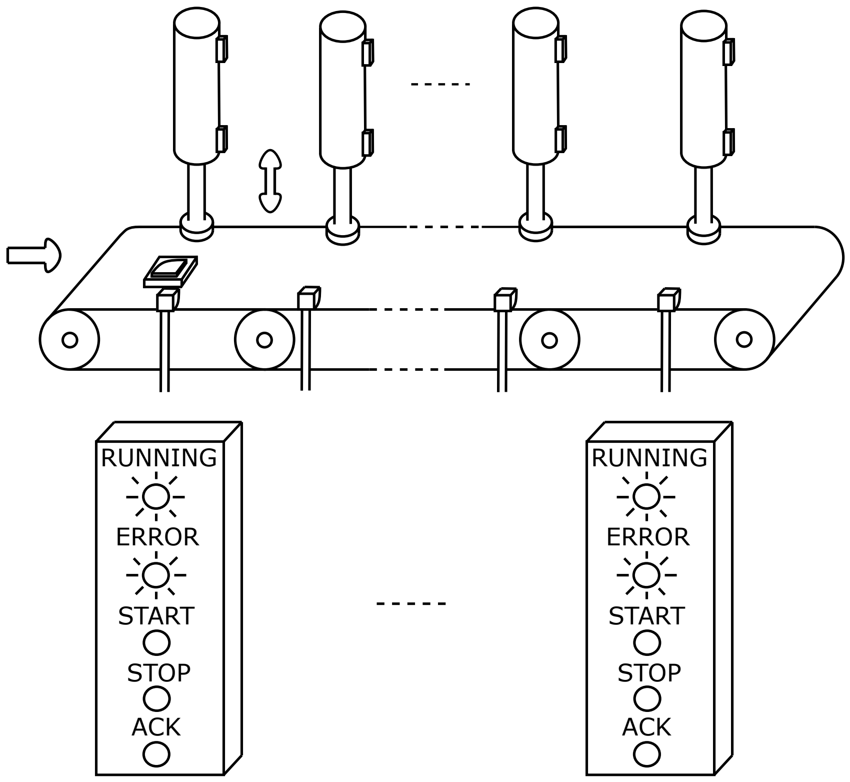


Figure 4.4: Sketch of the model situation

First of all, we need to create object-oriented templates representing general parts of the production line and implement their PLC code to describe parts behaviour. Secondly, we create the particular application consisting from a specific hierarchy of components. Next, we create the CPUs representing the control system and assign to each component CPU which will be executing its control code. Finally, the proper PLC code according to IEC 61131-3 represented by PLCopen XML file for both CPUs can be generated.

4.8.1 Templates

The templates are production line control desk, cylinder, belt conveyor, and motor. We suppose the following behaviour of general devices - components.

Production line

The top level entity is a production line template. It collects all information from its subcomponent. It realises the controlling of all its subcomponents.

Control desk

Control desk contains three push buttons - run, stop, acknowledge and two indicator lights, error and run light. After pressing start pushbutton production line starts to do its task. The belt conveyor runs, and cylinder can eject. After pressing stop, it stops the task. The belt conveyor stops, and cylinders can't eject.

If there is an error, the line is stopped, and error acknowledgement and again start are required. The acknowledge push button acknowledges the error state. If the line is running, the run indicator lights. During the error state the error indicator blinks.

Cylinder

The single acting pneumatic cylinder represents the e.g. pressing device. Each cylinder is equipped with limit switches and part presence sensor. If part presence sensor detects the part, the cylinder ejects. After ejecting it inserts again to its default state.

We want to detect some error states. For example, if the cylinder doesn't eject in five seconds since supplying the compressed air. Also, there is the error if both limit switches are active at the same time.

Belt conveyor

Belt conveyor transports the parts through the assembly process. It consists of motors. Belt conveyor mustn't run if one or more cylinders are air supplied or if there is an error state.

Motor

Motor is just simple devices with on/off function.

4.8.2 Templates code

Control desk

Variable	Data type	Comment	Type
startButton	BOOL	start	input
stopButton	BOOL	stop	input
ackButton	BOOL	error acknowledgement	input
errLight	BOOL	error state indication	output
runLight	BOOL	run state indication	ouput
timer_0	BOOL	timer output	internal
timer_1	BOOL	timer output	internal
TON_0	TON	on-delay timer	internal
TON_1	TON	on-delay timer	internal

Table 4.1: Control desk template variables

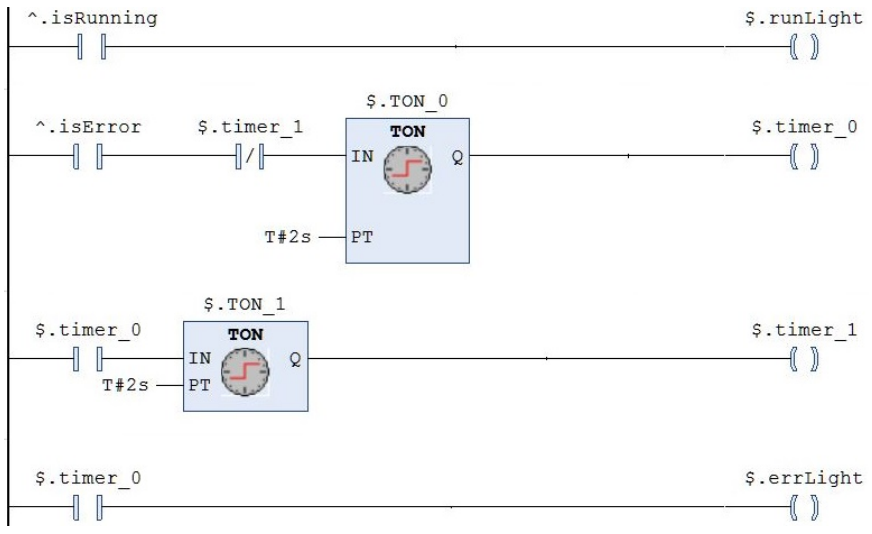


Figure 4.5: Control desk template code

Cylinder

Variable	Data type	Comment	Type
isInserted	BOOL	limit switch	input
isEjected	BOOL	limit switch	input
isPart	BOOL	part presence detection	input
air	BOOL	air supply	output
eject	BOOL	eject command	internal
insert	BOOL	insert command	internal
err1	BOOL	error state no. 1 - cylinder is not ejected in 7s after air supplying	internal
err2	BOOL	error state no. 2 - both limit switches are active at the same time	internal
R_TRIG_0	R_TRIG	rise edge detection	internal
R_TRIG_1	R_TRIG	rise edge detection	internal
TON_0	TON	on-delay timer	internal
TON_1	TON	on-delay timer	internal
RS_0	RS	RS flip-flop	internal

Table 4.2: Cylinder template variables

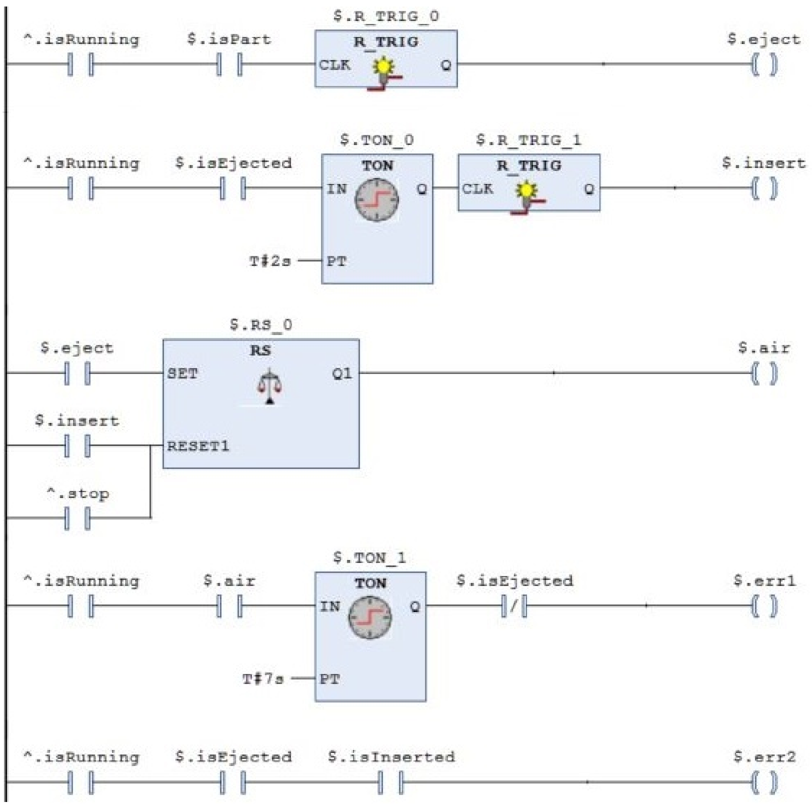


Figure 4.6: Cylinder template code

Belt conveyor

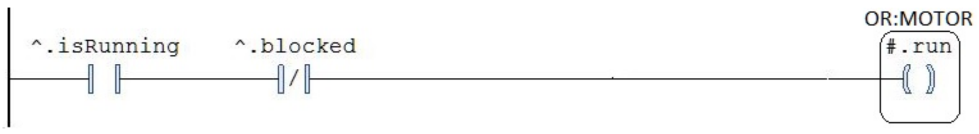


Figure 4.7: Belt conveyor template code

Motor

Variable	Data type	Comment	Type
run	BOOL	motor on/off	output

Table 4.3: Motor template variable

Production line

Variable	Data type	Comment	Type
start	BOOL	global start command	internal
stop	BOOL	global start command	internal
ack	BOOL	global ack command	internal
err	BOOL	global err command	internal
blocked	BOOL	global blocked state	internal
isError	BOOL	global error state	internal
isRunning	BOOL	global run state	internal
RS_0	RS	RS flip-flop	internal
RS_1	RS	RS flip-flop	internal
R_TRIG_0	R_TRIG	rise edge detection	internal
R_TRIG_1	R_TRIG	rise edge detection	internal
R_TRIG_2	R_TRIG	rise edge detection	internal
R_TRIG_3	R_TRIG	rise edge detection	internal

Table 4.4: Production line template variables

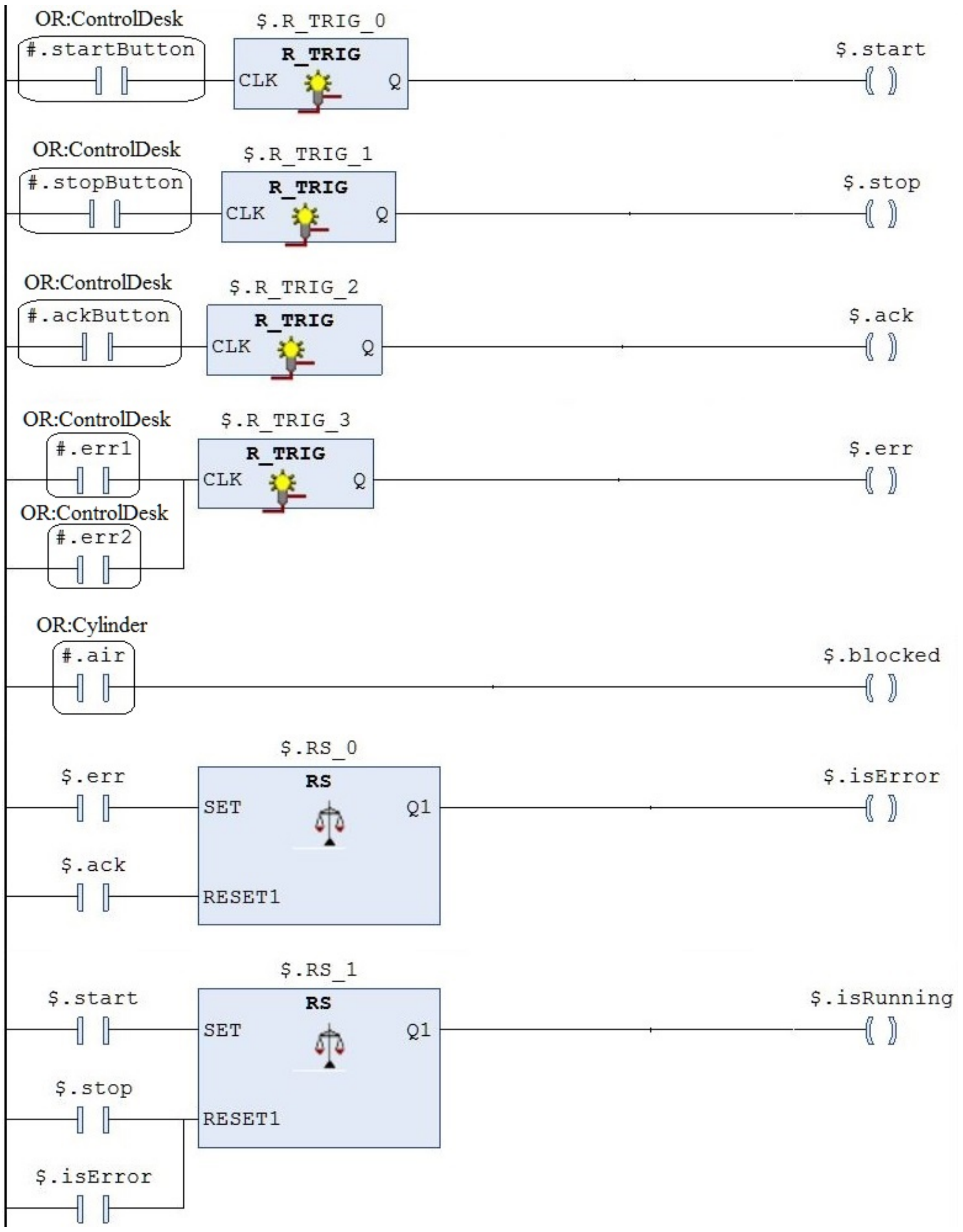


Figure 4.8: Production line template code

4.8.3 Application

Production line consists of one belt conveyor, two control desks and five pneumatic cylinders. Belt conveyor consists of ten motors. Components are created "instantiated" from templates. Now all components names and their counts are known.

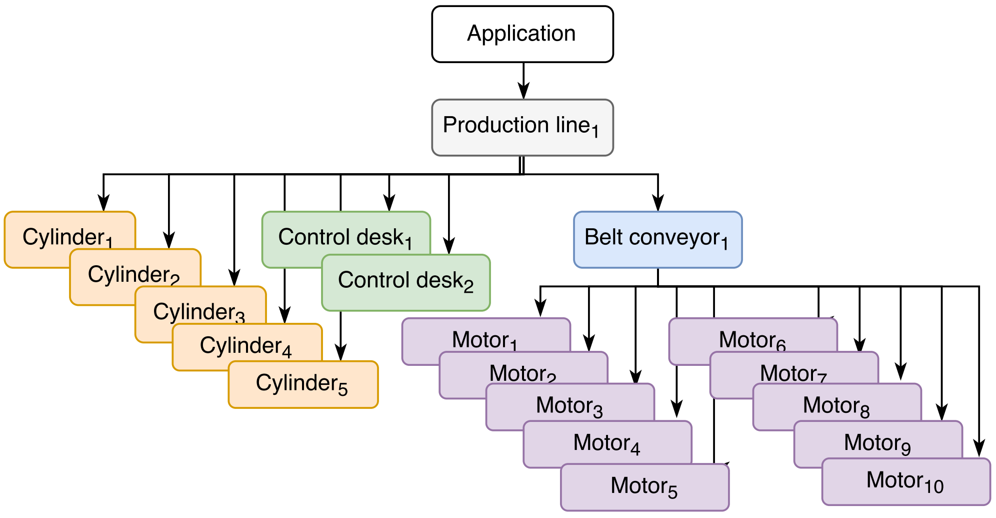


Figure 4.9: Example of the application

4.8.4 Control system

Despite the production line is not very complicated, the PLC control system includes two controllers placed in different locations in a plant connected by a bus.

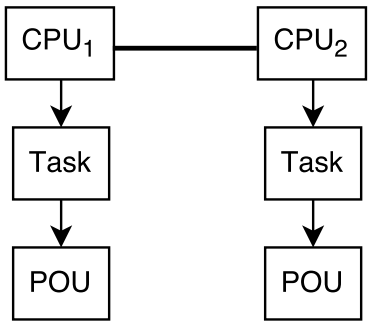


Figure 4.10: Example of the control system

4.8.5 Assignment

After creating the specific application user help by Assignment editor assign the CPU to each component.

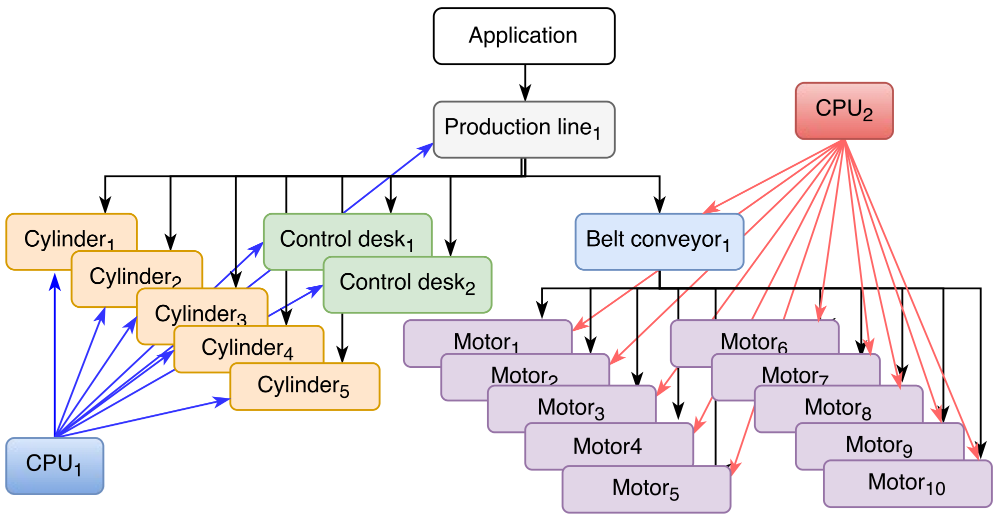


Figure 4.11: Example of the assignment

Chapter 5

Implementation

We implemented the development instrument in Java language with using the state-of-the-art Eclipse 4 Rich Client Platform (RCP) development framework. We called the RCP application representing the implementation of development instrument designed by us MDDE (simply shortcut for "Model-Driven Development Environment").

5.1 Eclipse

Eclipse is an open-source project providing a robust, full-featured, commercial-quality, and freely available development platform for the development of highly integrated tools. It provides an open, extensible integrated development environment (IDE) composed of plugins. So, its functionality can be extended by adding or changing plugins through Extension Points. It is created to run on multiple operating systems, and it provides integration with each underlying OS and fast and reliable native user interface (SWT, JFace libraries)[33].

Eclipse RCP uses Eclipse platform's components to build a fully-customizable stand-alone application easily extensible for future purposes.

Eclipse 4 is the next generation for building Eclipse-based tools and RCP applications, and it introduces a new set of technologies making the development of applications and tools easier. These are e.g.: a model-based UI, new CSS-based application styling, a new services-oriented programming model, support of Dependency injections [15].

5.2 Software architecture

MDDE's architecture is based on Model-View-Controller (MVC) pattern. It divides an application into three interconnected entities to separate internal data representation from ways the data are presented and accepted by a user.

The Model stores user data that are acquired by commands from the Controller and displayed in the View. The View (user interface) is output representation of the Model. It simultaneously serves to receive input from a user. The Controller accepts data and commands from user help by the View and updates Model's state and sends commands to actualize the View [34].

Because MDDE is implemented using the Eclipse development framework its consists of its specifics parts, and it uses its specific functionalities and services provided by Eclipse's API, but the MVC pattern is preserved.

5.3 User interface

The Eclipse application user interface (UI) consists of one or more windows. Usually, an application has only one window. Each window contains set of Parts which allow a user to navigate, create and modify data. The arrangement of the set of the Parts is called Perspective e. g. Eclipse IDE uses various perspectives for different development tasks.

Parts can be directly added to a Window or can be grouped by Part Stacks or Part Stash Containers. In Parts Stack, only one Part is visible at the same time. This Part is selected via part tabs. Parts in Part Stash Container are displayed at the same time divided horizontally or vertically.

Parts, in general, can be divided into Views and Editors. The view is typically used to operate on a set of data. The data can be hierarchically structured. Editors are typically used to modify a single data element e.g. file [15].

Without main UI parts, there can be a wide variety of secondary UI parts like buttons, popup menus, dialogs, tool panels, trim bars and many other.

5.3.1 MDDE's UI design and control

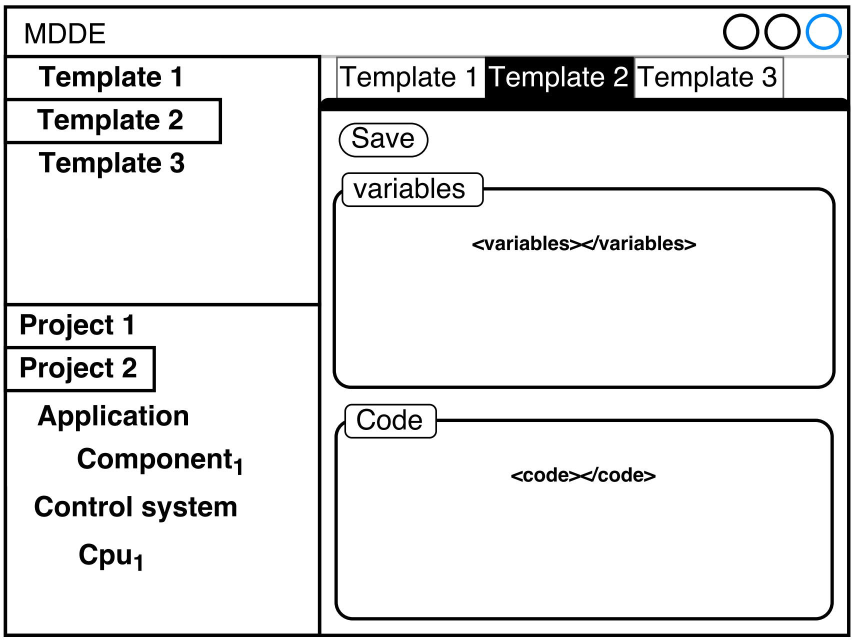


Figure 5.1: User interface design

The previous figure depicts the MDDE's user interface design. It consists of one main Window. Because our application it's not yet very complex, we need only one static perspective. The Window is vertically divided by main Part Stash container into two main parts.

The Left side is horizontally divided into two Views. Upper View serves as Template Explorer. A user can create and delete OO Templates here. Bottom View serves as Project Explorer. A user creates and deletes here Projects and their particular content - Control system and Application. The Control system consists of a list of CPUs. The Application consists of tree hierarchy of Components.

The right side of the main Part Sash container contains Part Stack collecting Editors of individual Templates. Template editor can be opened from Template Explorer View by mouse double-click on chosen Template. The Template Editor contains Save button, the Variables text area, and Code text area. Text areas are used to input and edit text representing PLC code of Editor's Template and its variables. Save button saves modifications of the Code and Variables text areas.

There are secondary user interface parts beside main parts mentioned above predominantly Popup menus and various Dialogs.

Popup menu allows a user to choose specific action after a right mouse click on a concrete UI object e.g. element in View. Popup menus are assigned to particular Views. There are two Popup menus in MDDE application. The First in assigned to Template Explorer View and the second is assigned to Project Explorer View. Through the Template Explorer View Popup menu, a user chooses Template to deletion and can create new Template.

The Project Explorer View's Popup menu is more complex. It serves for creating and deletion Projects, Components, and CPUs. Also, CPU assignment to particular Component is done through it as well as generating code action triggering.

Dialogs in MDDE are used to prompt the user for additional information like new Template, Project, Component and CPU name or provide a user with feedback like no text was inserted.

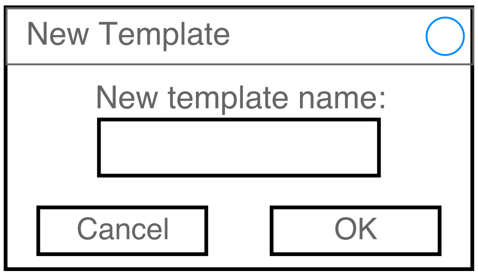


Figure 5.2: Example of Dialog - New template dialog

5.4 Model

5.4.1 PLC code

The PLC control code expressed in Ladder Diagram language is composed of Networks. Each Network is net of connected instructions (NO, NC contacts, coils, Function Blocks, etc.). Each net starts at left power rail and continues to the right power rail by a serial-parrel combination of instructions till coil ends it.

We implemented this using the similar Java class (object) structure. A list of Networks represents the PLC code. A list of Instruction represents the Network. The Instruction is a basic code element. Instructions connections are realized by Instructions references, but the situation is more complicated since Instructions could have more inputs and outputs. So each Instruction has a list of inputs and outputs then each input and output have references to connected Instructions. The list of variables can be very easily expressed by a list of Variable objects.

In our Model, we distinguish mainly between proper PLC code Instructions and Macro Instructions. Macro Instructions are in the later stage of code generating replaced by proper Instructions. OO features like indirect referencing and containment notations are only parameters of Instructions and thus can be easily replaced by appropriate Components names.

5.4.2 Template

Templates serve as code patterns for its next replacement by proper Components name and expansion of macro instructions. Template simultaneously stores code pattern's variables.

A user creates PLC code of Templates through the editor. Variables are declared help by editor too. In a fully qualified commercial development environment, the Ladder Diagram code editor is graphical. A user creates code by Drop and Drag actions and creates nets (ladder diagrams) of Instructions (contacts, coils, functions blocks). Variables are declared by text or table form in the unique part of the code editor or a separated editor.

In MDDE the PLC code editor is textual because design and implementation of the full graphical editor are very time consuming and the development of it is not the goal of this thesis. The Variable Editor is textual too.

Because we operate with XML structure due to PLCopen XML, we have decided to describe the PLC code of Templates' and their variables by XML structures too. Texts, expressing these XML structures, are stored in the String variables, which are edited trough particular Text areas in Template Editor Part as described in 5.3.1. Since we need describe only PLC code and variables, it is not necessary for the much-complicated structure as in PLCopen XML. The syntaxes of these XML structures are adapted for being easy to parse information from them and create the model expressing the PLC code and variables as described in 5.4.1.

Code XML structure

The root of the structure is Code element. Child elements of the root are Network elements. The child elements of the Network are Instruction elements. Instruction element contains elements storing information about its inputs' and outputs' connections, parameters like ID number, Instruction's type, and variable name. In the case of Macro Instruction, there are additional parameters like its type (OR, AND), Subcomponent type and its variable, type of Instruction to expand (coil, contact).

Variables XML structure

The root of the structure is Variables element. Its child elements are Variable elements. Variable element stores data about variable's name, type and comment.

5.4.3 XML handling

The Java has a rich set of libraries for XML handling. We have decided to use Document Object Model (DOM) API. A Document Object Model is a tree structure, where each element contains one of the components of an XML structure. The DOM provides functions to create and remove elements, change their contents, and traverse the hierarchy of the elements.

5.4.4 Project

The software unit representing the automation task is in MDDE expressed by Project. The Project consists of Application and Control system. The Application is tree hierarchy of Components. Each Component has reference to parent Component and a list of its Subcomponents. Each Component has a reference to Template, from which is "created." The Control system is a list of CPUs. After CPUs assignments, each Component has a reference to CPU, which will"execute" its code.

5.4.5 Code generation

The output of MDDE is single PLCopen XML file, storing one Program Organization Unit (POU) type Program and one Global Variable List (GVL), for selected CPU or CPUs in Control System. Thus, the process of code generation is separated in few stages and starts by user choice of CPU, for which PLC code and variables will be generated. The final Program is composed of PLC code fragments of Components' Templates. Similarly, the final GVL is composed of variables lists of Components' Templates.

The iterative BFS algorithm gradually searches the tree of Components. It starts at the root of the tree (Application) and adds all its children to the queue (First In First Out). Further, it iterates through the queue. If actual Component's CPU is equal to selected CPU, the following steps are executed:

- From String variable storing XML document, which represents the PLC code of Component's Template is created the Document object.
- From parsed parameters of the Document's nodes are created Instructions. During this step, the all indirect references and containment notations are replaced by proper names of given Component relating to actual Component.
- The DOM's nodes are parsed again, and Instructions created in previous steps are connected through their references based on again parsed ID numbers. This ensures that all connection will be linked because connecting Instructions through their references during the first process of parsing can cause that some Instruction can need to be connected to Instruction, which is not created yet.
- All macro instruction are expanded. Macro instruction is replaced by the set of Instructions specifically connected according to its type (OR, AND).
- All parsing, connecting, and expansion is ongoing within one network. After all these actions the Network as a list of connected Instructions is added to a list of Networks, which represents the PLC code.
- From String variable storing XML document, which represents variables of Component's Template is created the Document object.
- The Variables, created from parsed Document nodes' parameters, are added to the variable list, expressing the GVL. To all variables' names is added the particular name of actual Component.

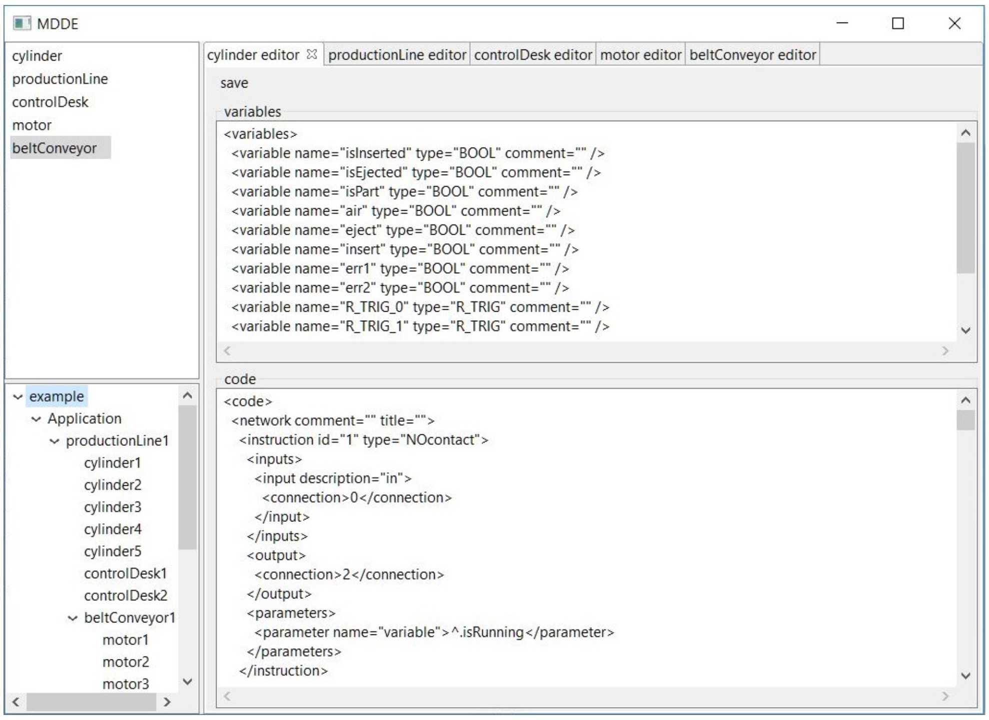
After these actions, the actual component is removed from the queue and all its children are added to the queue. When the queue is empty, BFS algorithm ends and in the list of Networks representing the PLC code and list of Variables representing the GVL are all objects, from which will be created DOM elements and these written to PLCopen XML file.

In PLCopen XML file, each tag expressing the code element has a local ID. Help by it the information about code elements' connections is stored. Before creating the DOM elements from Networks' and Instructions' objects, the IDs numbers, which will represent the local IDs, are generated and assigned to these objects.

Chapter 6

Evaluation

To evaluate the designed development process and implementation of MDDE, we implemented the example model situation from 4.8 in MDDE.



```
<variables>
  <variable name="isInserted" type="BOOL" comment="" />
  <variable name="isEjected" type="BOOL" comment="" />
  <variable name="isPart" type="BOOL" comment="" />
  <variable name="air" type="BOOL" comment="" />
  <variable name="eject" type="BOOL" comment="" />
  <variable name="insert" type="BOOL" comment="" />
  <variable name="err1" type="BOOL" comment="" />
  <variable name="err2" type="BOOL" comment="" />
  <variable name="R_TRIG_0" type="R_TRIG" comment="" />
  <variable name="R_TRIG_1" type="R_TRIG" comment="" />
```

```
<code>
  <network comment="" title="">
    <instruction id="1" type="NOcontact">
      <inputs>
        <input description="in">
          <connection>0</connection>
        </input>
      </inputs>
      <output>
        <connection>2</connection>
      </output>
      <parameters>
        <parameter name="variable">^.isRunning</parameter>
      </parameters>
    </instruction>
```

Figure 6.1: Screenshot of MDDE's graphical user interface

Evaluation process in MDDE

- We created Templates representing the parts of given example model situation.
- We described Templates' codes and variables by provisional XML structure described in XX.
- We created Project and given Component configuration and CPUs in it.

- We assigned CPUs to all Component is Project's Application.
- We generated the PLCopen XML files for both CPUs.

All used and generated XML files are too extensively to be shown here, so they are stored in attachment and attached CD.

Further, in e!COCKPIT [27] we created the testing project to more advanced evaluation.

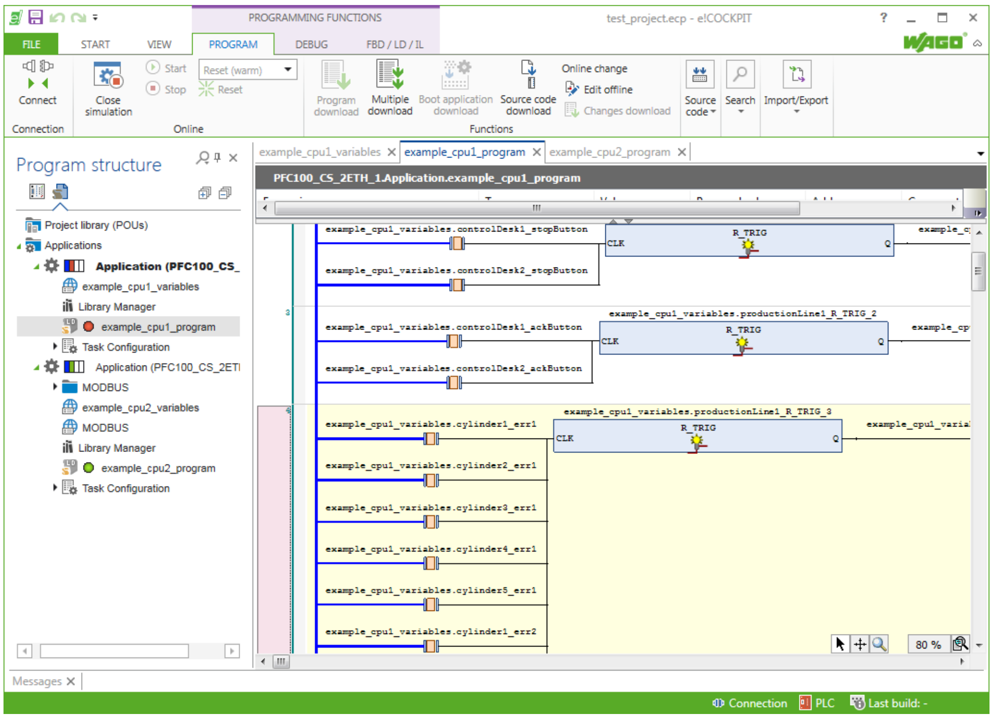

PROGRAMMING FUNCTIONS
test_project.ecp - e!COCKPIT
FILE
START
VIEW
PROGRAM
DEBUG
FBD / LD / IL
Connect
Close simulation
Start
Stop
Reset (warm)
Reset
Program download
Multiple download
Boot application download
Source code download
Online change
Edit offline
Changes download
Source code
Search
Import/Export
Connection
Online
Functions
Program structure
Project library (POUs)
Applications
Application (PFC100_CS_
example_cpu1_variables
Library Manager
example_cpu1_program
Task Configuration
Application (PFC100_CS_2ETI
MODBUS
example_cpu2_variables
MODBUS
Library Manager
example_cpu2_program
Task Configuration
example_cpu1_variables
example_cpu1_program
example_cpu2_program
PFC100_CS_2ETH_1.Application.example_cpu1_program
example_cpu1_variables.controlDesk1_stopButton
example_cpu1_variables.controlDesk2_stopButton
R_TRIG
CLK
Q
example_cpu1_variables.productionLine1_R_TRIG_2
example_cpu1_variables.controlDesk1_ackButton
example_cpu1_variables.controlDesk2_ackButton
example_cpu1_variables.productionLine1_R_TRIG_3
example_cpu1_variables.cylinder1_err1
example_cpu1_variables.cylinder2_err1
example_cpu1_variables.cylinder3_err1
example_cpu1_variables.cylinder4_err1
example_cpu1_variables.cylinder5_err1
example_cpu1_variables.cylinder1_err2
80 %
Messages
Connection
PLC
Last build: -

Figure 6.2: Screenshot of the PLC code simluation in e!COCKPIT

Evaluation process in e!COCKPIT

- We created the same device configuration as in XX; the two CPUs connected by bus.
- We imported generated PLCopen XML files to testing projects to check the correctness of generated PLC code.
- Because program performed in CPU2 uses the variables declared in CPU1, we set the tag sharing between these CPUS.
- We started PLC simulation to check that, PLC code is executable without errors.

In e!COCKPIT's Ladder Diagram editor the final generated code can be seen. The code is too long to be shown here. Complete e!COCKPIT testing project is stored in attachment and attached CD.

Chapter 7
Conclusion

7.1 Model situation

In 4.8 we presented simple model situation. We decided to describe the model automated plant by five Templates. We decided to describe the model automated plant by five Templates. Each Template can consist of variables and PLC code, but it's not necessary. In Belt conveyor template, no variable is needed. It is only "checkpoint" for in advance an unknown number of motors and without any complex function. On the other hand, the Motor template includes only one variable "run" representing on/off function of a motor. This variable is controlled by a macro instruction in 4.4 from Belt-conveyor template. We expressed that Belt-conveyor can have n motors, using that. In Production line template we had used macro instructions to collect information from other components and used them for setting control states like stop, start error, etc. These control states are as follows used help by containment referencing in 4.4 for controlling cylinders and control desks. Indirect referencing 4.4 is in Templates' code used for internal, control and help variables like timers, edge detections, and flip-flops.

7.2 Implementation

We implemented the development instrument described in 4. The application provides the basic graphical user interface and allows a user to perform development process in 4.5. The originally designed decomposition of tools of development instrument in 4.7 is not preserved but their function yes. The application's graphical user interface is dived into these main parts: Template Explorer, Project Explorer and Template Editors. They provide all functionalities of the individual tools from 4.7.

Template Explorer allows a user to create and delete Templates. Templates' PLC code and variables are edited through Template Editor. In Project Explorer, a user creates Projects representing the automation task. Project Explorer realizes rest of basic functionality. In Project's folder Control system a user can create and delete CPUs. In Application folder, a user creates from Templates and deletes the tree hierarchy of Components representing the parts of an automated plant. Further, it provides functionalities to CPU assignments and code generation.

7.2.1 Future work and extensions

The primary task for the future is the development of a graphical Ladder diagram editor and a table Variable editor. The actual way of PLC code editing is not suitable; the XML is technology for data storing and does not meet requirements on user comfort and quick PLC code editing. Now it serves as the intermediate step before the full graphical editor allowing developing and testing the PLC code generation process.

In application's Model (MVC architecture) we have implemented only functionalities needed to generation instructions included in the model example. These instructions are NO, NC contact, coil, TON, RS, R_TRIG and Maro OR. Thus, the extension of the set of instructions will be needed in the future.

7.3 Evaluation

We have evaluated the designed development process and implementation of the development instrument in two phases described in 6. In the first phase, we successfully generated PLC code by our implemented development tool. Secondly, we imported generated codes into fully qualified development environment e!COCKPIT and run the PLC code in e!COCKPIT's PLC simulator.

Bibliography

[1] K. Thramboulidis and G. Frey, “Towards a Model-Driven IEC 61131-Based Development Process in Industrial Automation”, Journal of Software Engineering and Applications, vol. 4, no. 4, pp. 217-226, 2011.

[2] B. Vogel-Heuser, S. Braun, M. Obermeier, K. S., and K. Schweizer, “Usability evaluation on teaching and applying model-driven object oriented approaches for PLC software”, 2012 American Control Conference (ACC), pp. 4463-4469, 2012.

[3] K. Thramboulidis, “IEC 61131 as enabler of OO and MDD in industrial automation”, IEEE 10th International Conference on Industrial Informatics, pp. 425-430, 2012.

[4] K. Thramboulidis, “Towards an Object-Oriented extension for IEC 61131”, Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies Factory Automation (ETFA 2012), pp. 1-8, 2012.

[5] A. Otto and K. Hellmann, “IEC 61131: A general overview and emerging trends”, IEEE Industrial Electronics Magazine, vol. 3, no. 4, pp. 27-31, 2009.

[6] K. Thramboulidis and G. Frey, “An MDD process for IEC 61131-based industrial automation systems”, ETFA2011, pp. 1-8, 2011.

[7] B. Werner, “Object-oriented extensions for iec 61131-3”, IEEE Industrial Electronics Magazine, vol. 3, no. 4, pp. 36-39, 2009.

[8] B. G. Silva and M. de Sousa, “Internal inconsistencies in the third edition of the IEC 61131-3 international standard”, 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1-4, 2016.

[9] A. Zoitl, R. Smodic, and C. Sunder, “Enhanced real-time execution of modular control software based on IEC 61499,”, Proceedings 2006 IEEE International Conference on Robotics and Automation, pp. 327-332, 2006.

[10] K. Thramboulidis, “IEC 61499 Function Block Model: Facts and Fallacies”, EEE Industrial Electronics Magazine, vol. 3, no. 4, pp. 7 - 26, 2010.

[11] K. Thramboulidis, "IEC 61499 as an Enabler of Distributed and Intelligent Automation: A State-of-the-Art Review—A Different View", Journal of Engineering, vol. 2013, 2013.

[12] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review", IEEE Transactions on Industrial Informatics, vol. 7, no. 4, pp. 768-781, 2011.

[13] J. H. Christensen, T. Strasser, A. Valentini, V. Vyatkin, and A. Zoitl, "The IEC 61499 Function Block Standard: Software Tools and Runtime Platforms", Conference: ISA Automation Week 2012, 2012.

[14] P. Tichý, P. Kadera, R. J. Staron, P. Vrba, and V. Mařík, "Multi-agent system design and integration via agent development environment", Engineering Applications of Artificial Intelligence, vol. 25, no. 4, pp. 846-852, 2012.

[15] L. Vogel, Eclipse 4 Application Development: The complete guide to Eclipse 4 RCP development, Wizard's wand series (Vogella). 2012.

[16] "PLC Scan Cycle". [Online]. Available: `http://plcsoftwar.blogspot.cz/2016/02/plc-scan-cycle.html`. [Accessed: 05-Apr.-2017].

[17] "PLC Memory". [Online]. Available: `http://automationprimer.com/2016/08/28/plc-memory/`. [Accessed: 05-Apr.-2017].

[18] "PROGRAMMABLE LOGIC CONTROLLER (PLC)". [Online]. Available: `https://www.myodesie.com/wiki/index/returnEntry/id/2962`. [Accessed: 05-Apr.-2017].

[19] "What is a PLC?". [Online]. Available: `https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/what-plc/`. [Accessed: 05-Apr.-2017].

[20] "Engineering Essentials: What Is a Programmable Logic Controller?". [Online]. Available: `http://machinedesign.com/engineering-essentials/engineering-essentials-what-programmable-logic-controller`. [Accessed: 05-Apr.-2017].

[21] "Programmable Logic Controllers (PLC): Chapter 6 - Ladder Logic". [Online]. Available: `https://www.allaboutcircuits.com/textbook/digital/chpt-6/programmable-logic-controllers-plc/`. [Accessed: 05-Apr.-2017].

[22] "A Developer's Perspective of PLC Configuration and Programming using FBD and ST". [Online]. Available: `https://www.design-reuse.com/articles/25025/plc-configuration-programming-fbd-st.html`. [Accessed: 17-Apr.-2017].

[23] "PLC Programming - How Do The Different Languages of IEC 61131-3 Compare?". [Online]. Available: `http://www.plcedge.com/plc-programming.html`. [Accessed: 17-Apr.-2017].

[24] "Programovací jazyky pro PLC". [Online]. Available: `http://www.plcedge.com/plc-programming.html`. [Accessed: 17-Apr.-2017].

[25] Technical Paper PLCopen Technical Committee 6: XML Formats for IEC 61131-3. 2009. Avaible: `http://www.plcopen.org/pages/tc6_xml/` [Accessed: 05-Apr.-2017].

[26] "CODESYS". [Online]. Available: `https://www.codesys.com`. [Accessed: 17-Apr.-2017].

[27] "WAGO". [Online]. Available: `http://global.wago.com/en/index-en.jsp`. [Accessed: 17-Apr.-2017].

[28] "BECKHOFF". [Online]. Available: `https://www.beckhoff.com/english.asp?twincat/twincat-3-extended-automation-engineering.htm`. [Accessed: 17-Apr.-2017].

[29] "XML (Extensible Markup Language)". [Online]. Available: `http://searchmicroservices.techtarget.com/definition/XML-Extensible-Markup-Language`. [Accessed: 05-Apr.-2017].

[30] "XML Tutorial". [Online]. Available: `https://www.w3schools.com/xml/default.asp`. [Accessed: 05-Apr.-2017].

[31] "IEC 61131-3:2013: Programmable controllers - Part 3: Programming languages". [Online]. Available: `https://webstore.iec.ch/publication/4552#additionalinfo`. [Accessed: 28-Apr.-2017].

[32] "What is IEC 61499?". [Online]. Available: `http://www.holobloc.com/papers/iec61499/overview.htm`. [Accessed: 30-Apr.-2017].

[33] "Eclipse documentation - Current Release: What is Eclipse?". [Online]. Available: `https://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fint_eclipse.htm`. [Accessed: 19-May-2017].

[34] "MVC Architecture:". [Online]. Available: `http://www.tutorialsteacher.com/mvc/mvc-architecture`. [Accessed: 19-May-2017].

Appendix A
Content of attached CD

Folder	Content
cz.cvut.fel.bp.mh.mdde	source codes
e!cockpit_testing	e!COCKPIT testing project
latex	latex source codes
mdde	executable MDDE application
mdde_testing_files	MDDE testing files