CZECH TECHNICAL UNIVERSITY IN PRAGUE

BACHELOR THESIS

Integration of IEC 61499 with OPC UA

Author:
Slavomír KOŽÁR

Supervisor:
Ing. Petr KADERA, Ph.D.

January 11, 2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2014-15**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	**Slavomír Kožár**
Studijní program:	**Otevřená informatika**
Obor:	**Počítačové systémy**
Název tématu česky:	**Integrace IEC 61499 s OPC UA**
Název tématu anglicky:	**Integration of IEC 61499 with OPC UA**

Pokyny pro vypracování:

Cílem práce je seznámení se standardem pro distribuované řízení IEC 61499, technologií pro průmyslovou komunikaci OPC UA a vytvoření komunikačního rozhraní mezi řídicí aplikací a OPC UA serverem. Toto zahrnuje příslušné rozšíření nástroje 4DIAC – IDE o konfigurační dialog komunikačních parametrů a rozšíření nástroje 4DIAC – FORTE o podporu samotné komunikace prostřednictvím existujícího a volně dostupného OPC UA klienta. Funkčnost navrženého řešení bude demonstrována na ukázkové aplikaci běžící na několika mini-počítačích Raspberry Pi.

Seznam odborné literatury:

[1] Zoitl, A.: Real-Time Execution for IEC 61499

[2] Lange, J., Iwanitz, F., Burke, T.: OPC – From Data Access to Unified Architecture

Vedoucí bakalářské práce:	Ing. Petr Kadera (ČIIRK)
Datum zadání bakalářské práce:	10. května 2015 (změna zadání); původní zadání 2. 12. 2014
Platnost zadání do[1]:	31. srpna 2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S. -4-

[illegible]
Doc. Ing. Jan Holub, Ph.D.
vedoucí katedry

[illegible]
Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 6. 2015

Declaration of Authorship

I, Slavomír KOŽÁR, declare that this thesis titled, "Integration of IEC 61499 with OPC UA" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a bachelor degree at the Czech Technical University in Prague.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date:

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Abstract

Faculty of Electrical Engineering

Bachelor

Integration of IEC 61499 with OPC UA

by Slavomír KOŽÁR

Today, in the time of fast changes on the global market and decreasing lifetime of the product, industry is forced into the philosophical shift. Manufacturing has to be quickly moved from the mass production to the mass customization. The physical changes in the production resources mean a need for dynamic reconfiguration at the control level. In order to achieve the real-time reconfiguration of the manufacturing systems, we need new software architectures and support from the execution environment.

The aim of this integration is to transform industrial control systems from the standard pyramide toplogy into the reconfigurable systems. 4DIAC, framework created according to the IEC 61499, allows a developer to create an industrial control application and brings them the feature of reconfigurating this application by creating and modifying FBs and connections among them. However, this reconfigurable application runs on only one industrial pyramide layer. The problem of this reconfiguration in control application is, that the systems on other layers do not react to this reconfiguration, because they are not capable of automatic detecting of this kind of reconfiguration. This problem was solved by using OPC UA as data sharing protocol. The information model of OPC UA which allows not only to store data, but also to store them in the structured web of interconnected nodes. Developped solution uses OPC UA not only to share values, but also structure of FBs with other systems in network. This brings new possibilities to the systems on other layers of the pyramide with detecting reconfiguration in 4DIAC application.

Acknowledgements

I would like to thank Dr. Techn. Alois ZOITL, author of 4DIAC framework for his consultations and assistance with this project. Last but not least, I would like to thank my supervisor Ing. Petr KADERA, Ph.D.for guidance with this paper and letting me to use this issue for my thesis.

List of Figures

List of Abbreviations

IPCMCS	Industrial Process Measurement and Control Systems
FB	Function Block
BFB	Basic FB
CFB	Composite
SIFB	Service Interface FB
ECC	Execution Control Chart
FBDK	Function Block Development Kit
FBRT	Function Block Run-Time
WSDL	Web Services Description Language
OPC UA	OPC Unified Architecture

Chapter 1

Introduction

Nowadays, we are standing in the time, when the fourth round of the industrial revolutions starts. Each of these revolutions was caused by the technological improvements. First one was caused by the change from labor work to the mechanization. The second one was started by the electrification. In this revolution, electric machines were used instead of the steam based motors. The third revolution was the last one, and it was caused by the digitization and the invention of the logical circuits. When we realize how much did the computers evolve, it's logical that also industry has to undergo another revolution. The upcoming revolution is caused by the introduction of the Internet of Things into the industry. [Brettel et al., 2014]

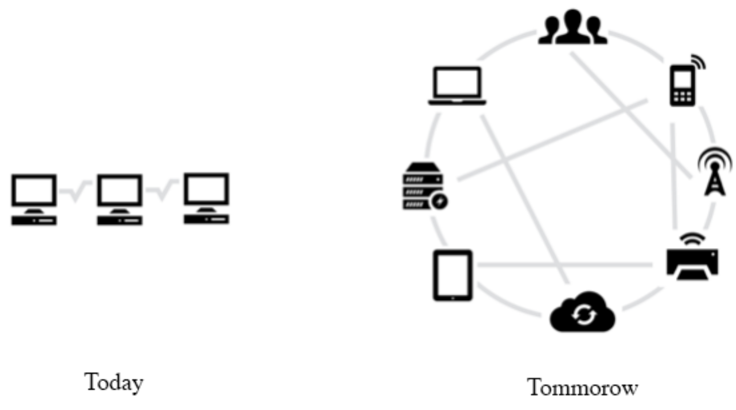


FIGURE 1.1: Change of industrial toplogy by introducing IoT

1.1 Reconfiguration

Today, in the time of fast changes on the global market and decreasing lifetime of the product, industry is forced into the philosophical shift. Manufacturing has to be quickly moved from the mass production to the mass customization. In order to rise to the challenge of these trends, the new operation methods are necessary. Production facilities need to be flexible, adaptable and allow fast changes at little cost. [Zoitl, 2008] Flexible production systems nowadays come with the higher cost, because these plants have higher initial costs, but even more important are the costs of downtimes needed to reconfigure such plant. Reconfiguration without the need to stop production is necessary.

This requires the reconfiguration of manufacturing plants at all levels, even the physical reconfiguration.

The physical changes in the production resources mean a need for dynamic reconfiguration at the control level. In order to achieve the real-time reconfiguration of the manufacturing systems, we need new software architectures and support from the execution environment.

On the figure below, the change needed to be done in the industry is shown. In the current approach, control system is divided into the layers which are horizontally configurable and the connection of the layers is declared in a static way. Reconfiguration of this kind of the control system requires the rebuilding of the whole pyramide from the bottom. While the new approach is also verticaly and horizontaly configurable.

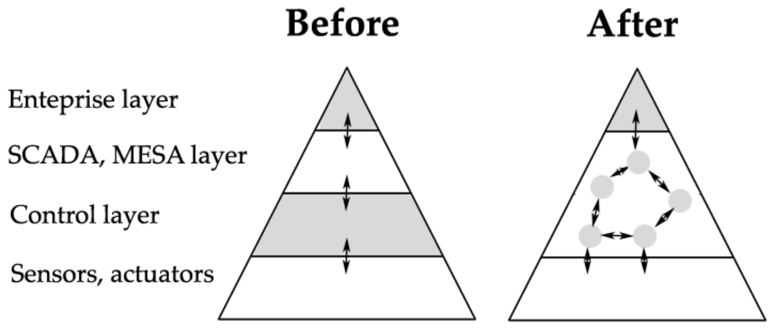


FIGURE 1.2: Change from strict layers into vertically and horizontaly configurable pyramide.

This thesis deals with the reconfiguration of the software of the control system. Whatever the solution for software able to reconfigure is, it must be simple, flexible, and have limited space requirements.

It is not so difficult to create reconfigurable software, when talking about huge applications running on the server clusters. However, different story is in the industrial control systems, which has to be runnable on small, often only 16 bit computers.

The simpliest theoretical solution is to create complex system capable of any function. To reconfigure this system, only the change of system parameters is needed. [Saad, 2003] This kind of solution can't be practically realized. System would be too large, consumpting too much of memory and storage, which makes this kind of system unable to run on the control system hardware. In case the technological advance would solve the lack of hardware resources, another serious problem occurs. In order to create such a system, developers would need to know all needed configurations of the system in advance. This is not real in the fast changing industrial world.

1.2 The aim of thesis

The aim of this thesis is to integrate OPC UA communication protocol into the system 4DIAC - controlling framework based on IEC 61499 standard.

Controll system created in 4DIAC framework is composed by the function blocks and connections among them. These function blocks and connections can be added, changed and removed from application on fly - during the run of application. By this ability to change the structure of the application while it is running, the reconfiguration on the control devices layers is achieved. To gain the reconfiguration on all of the layers, there is a need to destroy the hard boundaries between the layers.

In one or two layer systems (physical layer of actuators and control device layer), this would be enought to fulfill the aim of the reconfiguration. However, today in the large industrial systems, these two layers are not enough. As can be seen on 1.2, there is need to reconfigure also the connection between the layers, not only the layers themselves. To reconfigure the whole pyramide, not just single layers, need to change the comunication system arises.

OPC UA communication protocol allows user to create topologically ordered web of nodes called OPC UA Data Model. This nodes can contain object, value, of function which can modify this structure, or values.

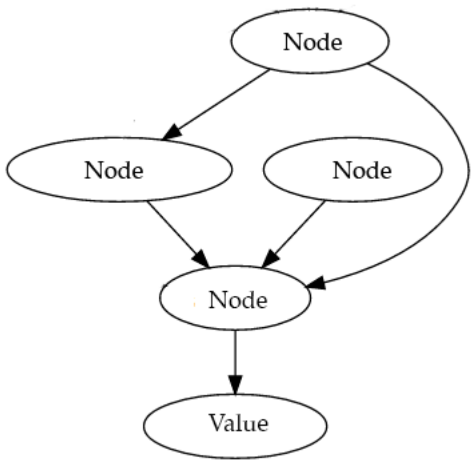


FIGURE 1.3: Data Model containing web of nodes.

This data model makes OPC UA technology suitable for transformation of idustrial pyramide from layer system into the system, where layers are stored in Data Model shared by all devices and technologies in control application.

Task of this thesis is to implement the communication stack inside of 4DIAC function blocks. Including client and also the server.

This thesis also aims to create data topology on the server based on a structure of the control system developped using a 4DIAC framework. By the integration of these two technologies, creates a system in which all elements of distributed control system could load structure and status of every other element using OPC UA protocol.

1.3 Chapters overview

Following second chapter contains a brief dive into the IEC 61499 standard and 4DIAC framework based on this standard. Basic principles of this framework as creating application, function blocks, deploying applications are described. Important part of using 4DIAC framework is the compiling of your own version of 4DIAC runtime environment dedicated for your device. The whole appendix is dedicated to this topic.

The third chapter is dedicated to communication protocol OPC UA, ways of using this protocol and its information model. Stacks based on OPC UA protocol are mentioned, focusing on OPEN 62541 stack, which was chosen to be used in this thesis.

In the fourth chapter, the solution of the integration problom described in the previous sections of this chapter is explained. Progress in the solution is divided into two stages. In the first stage, the basic principles of using 4DIAC and OPC UA stack are tested. The analysis of the best way of the integration of OPC UA with 4DIAC precedes second stage defining methods and function blocks in order to fill the aim of thesis and also to create interface suitable for use in the real control systems.

Chapter 2

IEC 61499

IEC 61499 is a new family of standards for Industrial Process Measurement and Control Systems (IPCMCS). This family consist of four parts:

1. IEC 61499-1 : Function Blocks - Part 1: Architecture
2. IEC 61499-2 : Function Blocks - Part 2: Software tools requirements
3. IEC 61499-3 : Function blocks for industrial-process measurement and control systems - Part 3: Tutorial information
4. IEC 61499-4 : Function Blocks - Part 4 : Rules for compliance profiles

Main topic of all these standards is defining Function Block (FB) and connections among them. This is also most important topic of this standards family, so there is no need to reference all of this individual standards and in this Thesis term IEC 61499 refers all four parts of this family.

IEC 61499 is based on an older IEC 61311 (1993) family of standards, which is the most common adopted standard in domain of IPMCS. [Zoitl, 2008] This makes IEC 61499 easy to adopt. There are also another key features which makes IEC 61499 easy to adopt standard like its modularity, distribution support, reconfiguration support and event-triggered execution model.

2.1 Introduction to IEC 61499

The IEC 61499 standard defines several models, which a developer uses to create a distributed control application in a graphical manner.

This brief introduction provides insight into the IEC 61499 standard for purposes of this thesis. A full description of architecture may be found in IEC61499-1 [Commission et al., 2005].

Models which are defined in IEC 61499 are (in hierarchical order, from the global model to atomic one): the application model, the system model, the device model, the resource model, the FB model.

The application model consists of the multiple system models, each of these system models consists of multiple device models etc.

The Base and most important model of IEC 61499 is FB. FB is independent, self-contained software component with the interface through which

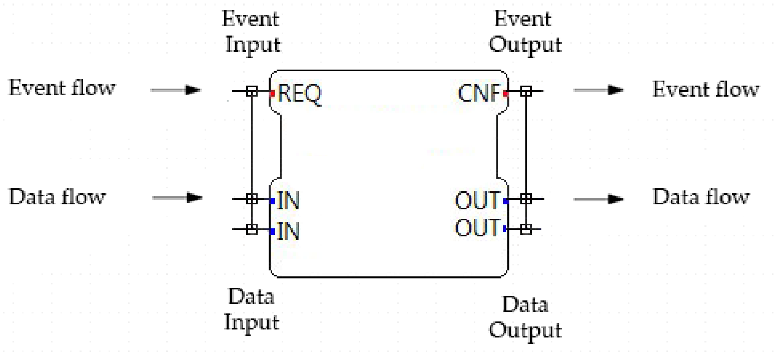


FIGURE 2.1: Standard FB IEC 61499 FB interface

it provides specific functions. This model was taken from IEC 61131-3 standard. In contrast to IEC 61131-3 FB definition in IEC 61499 event interface is added. The function block function is triggered by one of the input events. During the execution FB processes input data, set output data. When the processing is done FB generates triggers output event.

When comparing IEC 61131-3 and IEC 61499 the biggest difference is in the even-driven execution, while in IEC 61131-3 function was triggered by the cyclic execution.

Cyclic execution was problematic. It does not allow mass using of IEC 61131-3 in distributed systems. This type of execution is reliant to the system clock. This approach is not problematic in the scope of one device system. However, in a system with multiple devices there is a problem of sharing the system time. It is practically impossible to run this kind of system synchronously. In case of the cyclic execution every 1ms and not precise synchronous system it can take up to 1ms to handle any kind of change. In some kind of applications this time delay can lead to the destruction of product, machine or even whole manufacturing system. This problem can be solved by decreasing time between two executions. However this solution of delay problem is causing need of bandwidth for data transfer. It leads to the cost of data transport layer increase and also scale up data transfer error rate possibility.

In IEC 61499 standard this problem was solved by changing cyclic to event driven execution. Function Blocks are not executed cyclically, but are triggered by events. This solution prevents the problem with the central time and its sharing and caused also rapid decrease of needed bandwidth. In this approach the data are transferred only when an event is triggered. For example function block handling the end switch of machine does not have to propagate its state every 1ms like in the previous example. It propagates its state only when change state event occurs.

There is no support in IEC 61499 for cyclic execution anymore, but for purposes of back compatibility there is a solution of implementing IEC 61131 function into IEC 61499 system. The situation of a program is simply depicted by triggering of the cyclic execution by the use of an E_CYCLE FB. [Sunder et al., 2008] This function block triggers regular event to start execution of IEC 61131-3 compatible applications.

2.2 Types of FB in IEC 61499

2.2.1 Basic FB

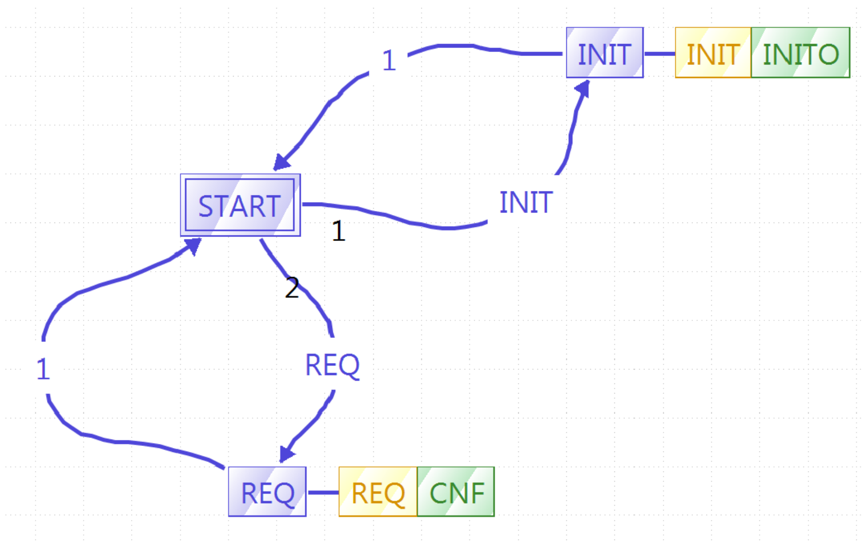


FIGURE 2.2: Example of basic FB's ECC

Basic FB (BFB) contains a state machine controlling internal execution called Execution Control Chart (ECC). ECC consists of three parts: ECC states with associated ECC actions and ECC transitions, which connects the states. ECC transitions are typically guarded by Boolean logical statements.

When an input event arrives, the first transition with true condition results in state change. With state entry also action associated with this state is executed. Algorithm can access only data input, data output and inner variables [Commission et al., 2005].

2.2.2 Composite FB

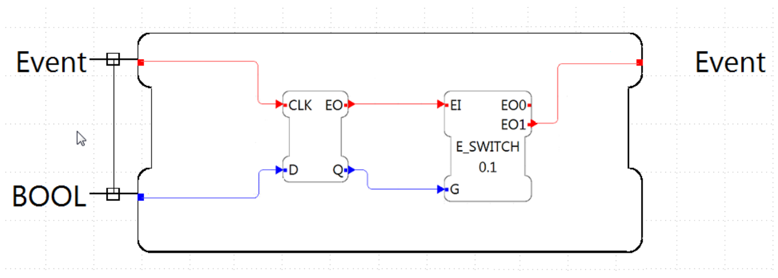


FIGURE 2.3: Example of composite FB structure

Composite FBs (CFBs) are containers for FB dedicated to generate cleaner design. Using Composite FBs, developer can create one FB for more complex, many times repeating function consisting of many basic or composite FBs. This allows designer to re-use his design. Incomming events and data connections are connected to the internal FBs and also outgoing connections are connected to internal FBs.

2.2.3 Service Interface FB

Service Interface FB (SIFB) is dedicated to function out of scope of IEC 61499. Typical function is the access to the device's hardware, I/O interface or communication interface. There are two general types of SIFBs in IEC 61499. Requester SIFB and responder SIFB. The requester SIFB remains passive, until it is application-triggered at one of its event inputs. The responder type is a resource or hardware triggered FB. It can trigger events by detecting actions of the hardware (e.g. interrupts) without need to trigger this FB from application.

2.3 IEC 61499 Base Model

Modeling of IEC 61449 system can be divided into two phases. In the first phase, designer creates Function Block Network by interconnecting of the FBs with data and event connection. In this phase, developer has in mind only the functionality and it does not depend on any device or control infrastructure. In the second phase, parts of the system model created in the first phase are mapped to control devices. For example, in Figure 2.4a, Application 1 is mapped to Devices 2, 3, 4, and 5, whereas Application 2 is mapped only to device 2.

IEC 61499 is executed on devices. Every device consists of device management component, communication iterface - provides communication between devices, process interface - provides services for accessing the sensors, actuators and other physical devices needed to control the process. Device can also contain resource.

Resources are functional units which contain applications or the parts of applications. Resources in device are independend. This means resources

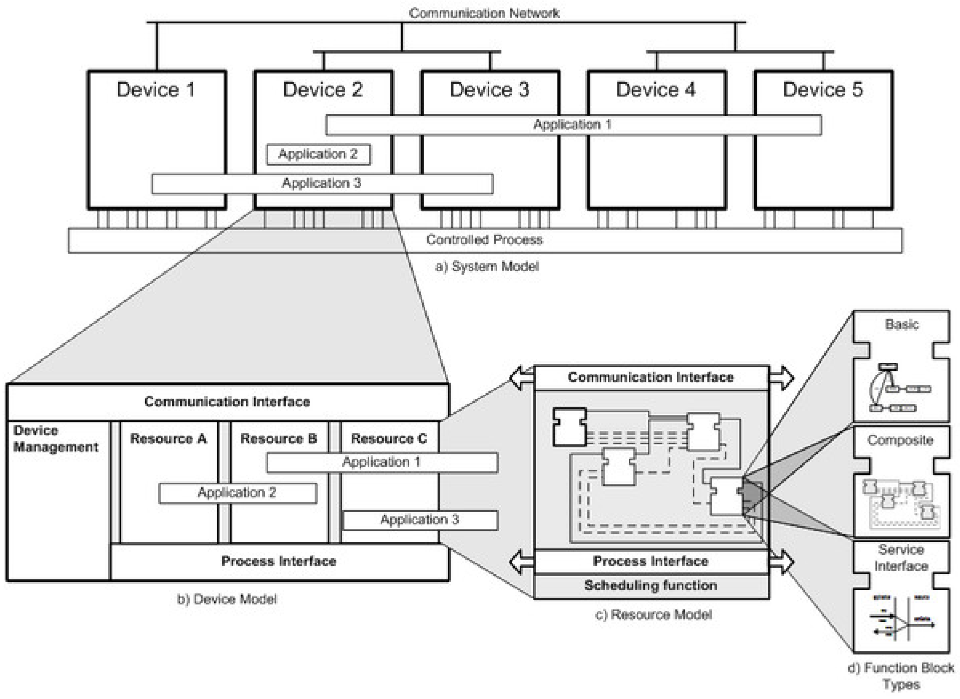


FIGURE 2.4: [**iec61499Model**]

can be added, modified, removed in any particular device without interferring any other resource. This approach is very important to reach the goal of reconfiguration. The task of the resource is to provide execution environment, delivering event notifications.

2.4 IEC 61499 applications

The current applications of IEC 61499 can be devided into the research and industrial sector. IEC 61499 standard exists since January 2005. Before standardization, in 2000 it was available in form of so-called Public Available Specification. Although IEC 61499 has been available in some forms for a long time, most published work on the standard up to now has been academic or only prototypical industrial test cases.[Strasser et al., 2008]

In industry sector the adoptions of IEC 61499 were mainly case studies and prototypes. A lot of case studies had a starting point via the Function Block Development Kit (FBDK) / Function Block Run-Time (FBRT) package from Rockwell Automation. FBRT is implemented in Java and IEC 61499 elements are implemented as Java Classes. This package is a reference implementation and was used to test models and standard. In FBRT the event notification is handeld by function call. The source FB calls notification function of the event connection object and this object triggers event on destination FB by calling his event function. This approach creates delays and is also one of the greatest reasons why FBRT has never been adopted by industry sector. Another reason is also that this Java implementation was not able to run on small industrial control platforms (8/16/32b computers).

2.5 The 4DIAC initiative

In July 2007 the 4DIAC open source initiative was founded by PROFACTOR GmbH and the Automation and Control Institute of Vienna University of Technology. Nowadays this initiative is conducted with and supported by international automation network O^{3}NEIDA.

Aim of 4DIAC initiative is to create an open-source framework based on IEC 61499 standard which will provide reference implementation of execution model for IEC 61499.

4DIAC initiative is currently developing two projects IEC 61499 compliant :

- 4DIAC IDE - engineering tool
- FORTE - runtime environment

To work with 4DIAC framework you have to use both of this parts.

You can find instructions how to install and run this project on your own computer in Appendix A. In the next sections brief informations about 4DIAC IDE and FORTE are introduced. These are just the minimal amount of necessary informatios in needs of this thesis.

2.6 4DIAC IDE

4DIAC IDE is IEC 61499 development evironmen based on the Eclipse open tool framework. Eclipse base makes 4DIAC IDE multiplatform open source IDE.

As all other IDEs based on Eclipse work in 4DIAC IDE is divided into perspectives. Every user can create his own perspective, but there are three perspectives which are created by default in 4DIAC IDE.

This perspective is dedicated to the basic creation of application. FBs can be added, created event and data connection. Below system configuration of one of the example supplied with 4DIAC IDE can be seen.

Application of figure consist of two devices, connected via Ethernet. Every of this device includes two resources. One of the resources is management resource allways named MGR and read-only.

By double clicking on resource you can edit Function Block Network running on this resource.

Type Management Perspective is dedicated to editing and creating of developers' own FBs. On the figure below, there are shown tools which can be applied on the function block. In the case of the basic FB you can edit also function of this FB by editing its EEC or Algorithm writen in pseudocode. The function of the Composite FB can by modified or created by editing Composite Network. Only Service Interface FBs function is not allowed to change in 4DIAC editor. Function of SIFBs can be modified only by editing forte source.

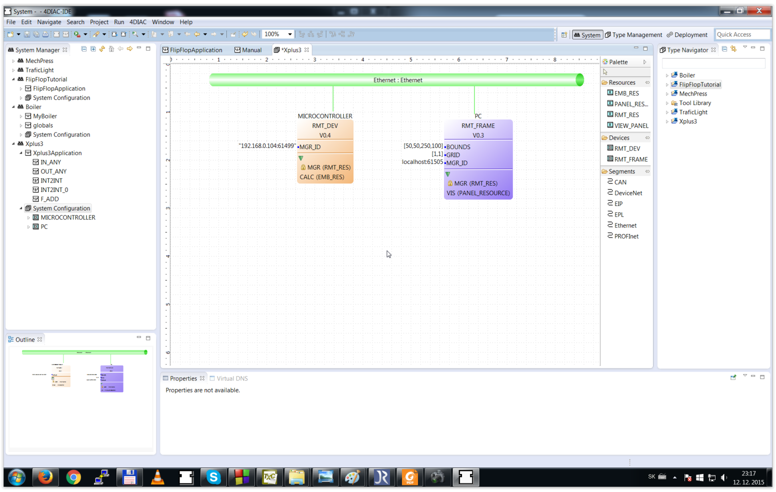


FIGURE 2.5: Perspective dedicated to basic creation of application

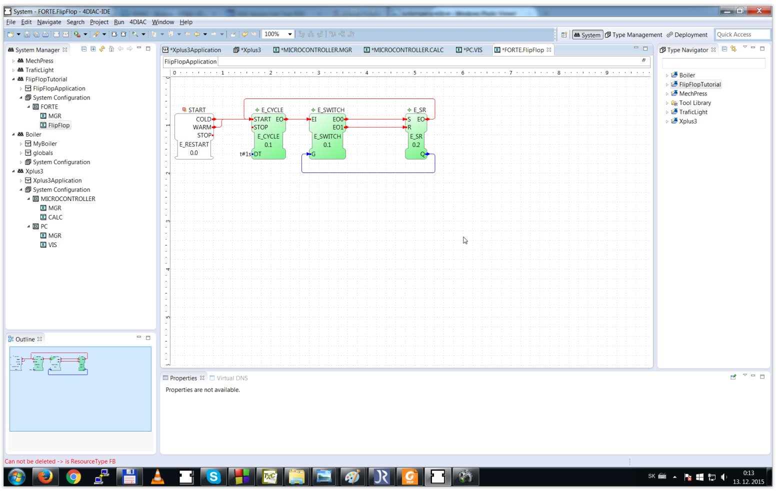


FIGURE 2.6: Editing Function Block Network in resource

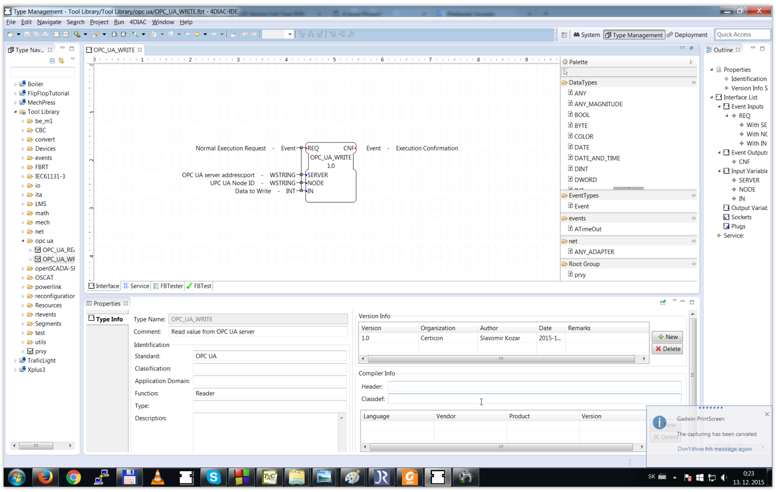
All changes made in Type Management Perspective have to be exported into the forte code. To use this modified FBs in control system it is necessary to recompile the FORTE with these updated function block.

Deployment Perspective is dedicated to the deployment and upload application into the control system devices by clicking on Download button.

There is also possibility to run local FORTE and FBRT directly from Deployment Perspective. In case of local FORTE runtime, all its output are shown in Console window.

2.7 4DIAC RUNTIME ENVIRONMENT - FORTE

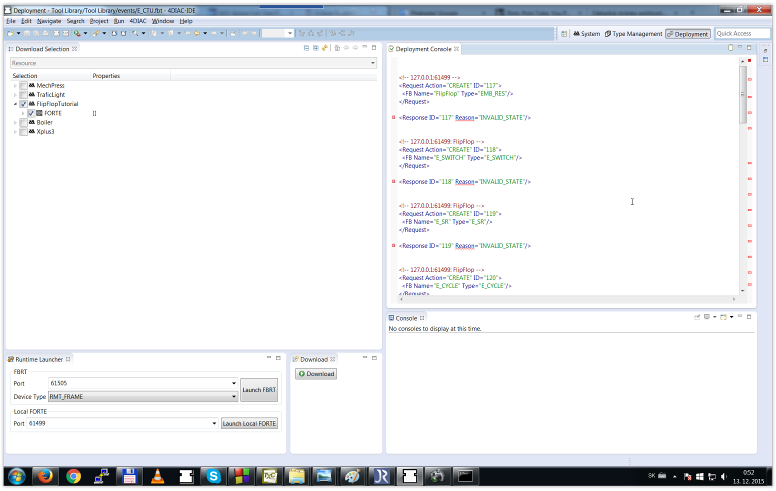
The FORTE is a portable C++ implementation of an IEC 61499 runtime environment. It is focused on the small embedded control devices (also 16/ 32



Type Name: OPC_UA_WRITE
Comment: Read value from OPC UA server
Identification
Standard: OPC UA
Function: Reader

Version	Organization	Author	Date
1.0	Certicon	Slavomir Kozar	2015-1...

FIGURE 2.7: Editing or Creating FBs



```
<!-- 127.0.0.1:61499 -->
<Request Action="CREATE" ID="117">
 <FB Name="FlipFlop" Type="EMB_RES"/>
</Request>

<Response ID="117" Reason="INVALID_STATE"/>

<!-- 127.0.0.1:61499: FlipFlop -->
<Request Action="CREATE" ID="118">
 <FB Name="E_SWITCH" Type="E_SWITCH"/>
</Request>

<Response ID="118" Reason="INVALID_STATE"/>

<!-- 127.0.0.1:61499: FlipFlop -->
<Request Action="CREATE" ID="119">
 <FB Name="E_SR" Type="E_SR"/>
</Request>

<Response ID="119" Reason="INVALID_STATE"/>

<!-- 127.0.0.1:61499: FlipFlop -->
<Request Action="CREATE" ID="120">
 <FB Name="E_CYCLE" Type="E_CYCLE"/>
</Request>
```

No consoles to display at this time.

FIGURE 2.8: Uploading application into devices

bit controllers) and provides execution of all IEC 61499 types of functions blocks. Currently is forte available for Windows, Posix (Cygwin, Linux), NET+OS 7, eCos. It can be also used on small embedded boards like RaspberryPi, BeagleBone or even Lego Mindstroms nxt.

2.7.1 Function Blocks in the FORTE

Basic and composite FBs are easy to create and edit in 4DIAC IDE. However Service Interfaces FBs, which are most important, because serves connection to the physical devices in control system, are defined twice. In 4DIAC IDE only outer interface, like event and data inputs outputs are defined function of these function blocks is defined in C++ function generated to every function block. Creating of function block will be discussed in the next chapters.

Chapter 3

OPC Unified Architecture

3.1 Service Oriented Architecture

Service Oriented Architecture is the family of principles which recommends the assembling of the composite application and any other systems from the independend parts servicing any kind of the service.

Nowadays, most of the information technologies are moving from monolitic closed systems into distributed system. With incomming Internet of Things into industry, this section is not an exception.

There is a tendency to connect industrial applications together to create larger systems, despite the fact, that they may differ a lot from each other, work on different platforms in different locations.

SOA principles forms connection among different systems

Main connection between different systems is the SOA, principles which can connect technologies on different platforms. This technology transfers alone control and regulation systems into the global solutions.

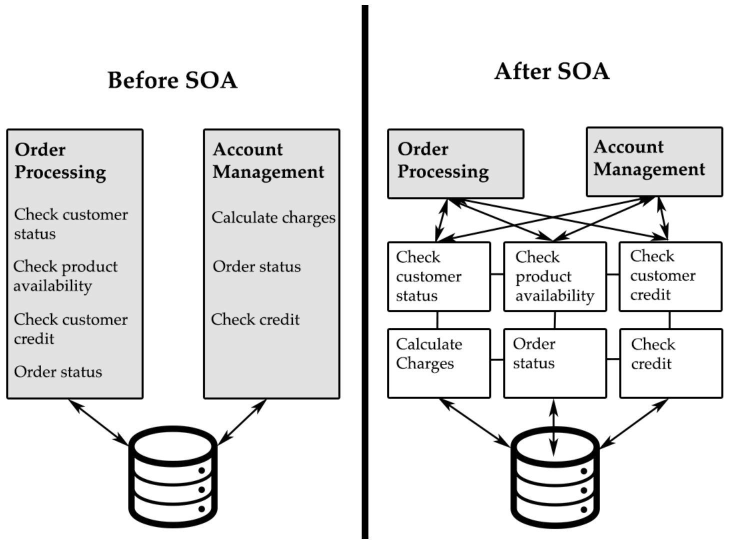


FIGURE 3.1: SOA effects on the e-shop application.

On the figure above it is obvious how SOA converts monotlitic and not-scalable system into the much more clear solution. Creating of the same application using single atomic services leads into the system which is much more scalable, reconfigurable and independent modules servicing data supports also much more debugging abilities of service without need to shut down the whole system.

This approach brings re-usability feature, which is common in the other IT sectors, into the industrial systems.

3.2 Web service

Web service is a software system for the communication of two computers in the network. It is described in a Web Services Description Language (WSDL). A Web service supports direct interactions with the other software agents using XML-based messages exchanged via Internet-based protocols. Alonso et al., 2004 Often the web port 80 is used to transfer data. This port is used mainly by web http protocol, so is opened on every firewall. That is the huge advantage against any kind of proprietal ports.

3.3 OPC Unified Architecture

Information system exceeds the borders of plant or event company, when companies are working together on common projects and products. Due to the huge demand of integration and interconnecting different control system, the standardization is needed. Standardization of information systems reduces cost and time of integration. With standards, also possibility to create generic adapters between any kind of systems comes.

OPC Foundation answers this need of standardization. In the begining, the OPC standed for OLE for Process Control. The OLE itself is proprietary technology called Object Linking and Embedding. This technology is used to create references between the data objects in Windows OS. Microsoft later published SDK for this technology, which leads to creation of the OPC.

OPC Foundation decided to redesign OPC components and technologies with modern, vendor independent solutions.Hannelius, Salmenpera, and Kuikka, 2008

The new specification is called OPC Unified Architecture (OPC UA). Nowadays the OPC means Openness, Productivity and Colaboration.

Currently, OPC is the communication standard in automation technology. Migration to OPC UA is needed to increase possible types of the integration solutions for which UPC can be used. This is achieved by using standard technologies to implement SOA and WS.

3.3.1 SOA in OPC UA

OPC UA is based on SOA. OPC UA server contains set of services which are used by clients. These services provides all OPC UA functionality. Set of services available in any particular OPC UA server is defined in profiles that are described in OPC UA specification.

Each service call in IPC UA consist of a request and response message. In OPC UA there is a huge difference between services and methods. Services are strictly defined in the standard and user cannot change it, methods are user defined. To invoke user-defined method on OPC UA server using of the service is needed.

3.3.2 OPC UA communication stacks

All the OPC UA standards are published by the OPC UA Foundation, but no official communication stack has been published yet. OPC Foundation published just the example code in Java and Ansi C, but no complete SDK or even documentation.

However there are few open source or proprietary stacks available. On OPCConnect website [*OPC UA stacks overview*] you can find Overview of Available SDKs and toolkits.

There are also some open stacks for OPC UA. However these stacks are often published under license which is not compatible with 4DIAC license. I can mention OpenOpcUa which is open source, but to use it there is need to pay the one time fee. I can also mention FreeOpcUa hosted by the GitHub, but this sdk is not fully working and lack of documentation makes it almous impossible to use it for purpose of this thesis. However FreeOpcUa is C/C++ and Python SDK and in Python version much bigger progress is made. This SDK serves great open-source Python GUI interface for discovering the OPC UA server.

Considering two important parameters license and documentation, open62541 stack seems to be optimal. This stack is used to integrate OPC UA into FORTE and will be described more in the below sections.

3.4 open62541 stack

Open 62541 is communication stack based on OPC UA standards published as IEC 62541 licensed under LGPL and free available on GitHub.

This stack is fully scalable, supports multi-threaded architecture, where every connection or session is operated by separate thread.

Open 62541 is written in C99 with POSIX support, so it is able to run on Windows, Linux, MacOS and Android. POSIX Linux support means open62541 stack can also run on small embedded machines like raspberryPi, PLCs, etc.

3.4.1 Building sdk

After downloading open62541 stack sources from GitHub, it is necessary to build them into header files. Also pre-generated sources and header files are available to downlaod. This pre-generated sources are just some demo with only basic functions like server, client and its basic functionality like read and write data. In this thesis also another functions like browsing across nodes on server and creating, editing and deleting nodes were needed, so in order to fullfil the aim of this thesis specific sources were build.

To build open62541 stack for use in this project it is necessary to set UA_ENABLE_NODEMANAGEMENT option during build configuration.

For detailed information about building library visit oficial documentation of open62541. [*OPEN61541 stack building*]

Chapter 4

The Realization of Integration

Integration of OPC UA and IEC 61499 can be divided into two separate stages.

During the first stage module for 4DIAC framework and simple OPC UA server and client using open62541 stack were created. This stage was very important, because there has not been any comprehensive guide for 4DIAC modules creating publshed yet.

While in the stage one the module for OPC UA integration into 4DIAC framework used only dummy client and server, the second phase is preceded by an of options of integration usable in industrial practice.

In the second stage the choosen way of integration was developped.

In the first stage of the integration of OPC UA features into the framework 4DIAC consisted of creating simple FBs called OPC_UA_READ and OPC_UA_WRITE in OPC_UA module, which is an organizational unit of FBs in 4DIAC FB library. In these function blocks, the basic functions of open62541 stack were used. OPC_UA_WRITE is the function block capable of writing value into the information model of OPC UA server, running on a remote host.

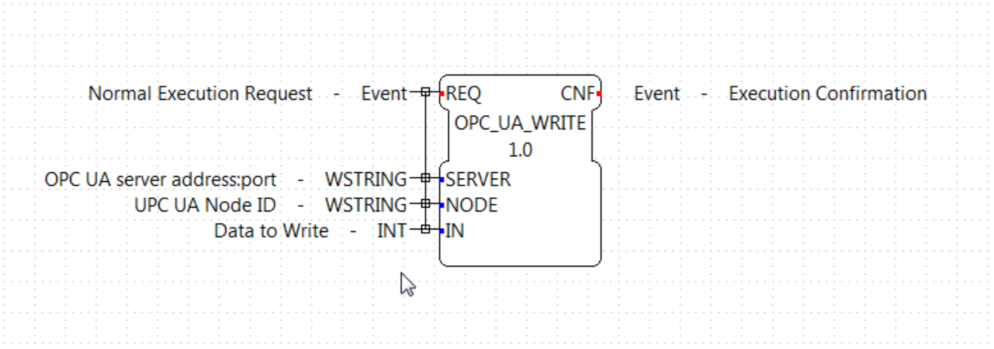

Normal Execution Request - Event
REQ
CNF
Event - Execution Confirmation
OPC_UA_WRITE
1.0
OPC UA server address:port - WSTRING
SERVER
UPC UA Node ID - WSTRING
NODE
Data to Write - INT
IN

FIGURE 4.1: OPC_UA_WRITE FB interface.

OPC_UA_READ is the function block in which the function to read a value from a OPC UA server on remote host is implemented. These both FBs have, except data input / output and event input and output, also inputs to select remote host IP address and port and node in which the value will be stored / from which the value will be read.

Any developer could use these function blocks to transfer any kind of data values among any resources, devices or applications in the same network.

4DIAC framework already features simplier method of data transfer in network using PUBLISH and SUBSCRIBE FBs, but this connection is just peer-to-peer. This means only transfer between two resources, devices, or applications is possible. So while build-in solution in 4DIAC allows only connection between two points, both running 4DIAC runtime, OPC_UA_WRITE and OPC_UA_READ can write and read values from any distant or local server, which can create connection among multiple devices, not necessarily using 4DIAC. In conclusion, these function blocks can find their purpose, e.g. in centralized data collecting.

Even that FBs can be used to trasfer data, this solution does not use the OPC UA information model in full range. The goal of this stage was discovering possibilities of the open62541 stack, its elementary functions like read and write values stored in pre-defines information model.

4.1 Module and FBs development

The first stage was very important stepping-stone to understand the work with 4DIAC framework, developing the own function block and source codes of C++ implementation 4DIAC runtime environment - FORTE.

There are basic tutorials for developing basic and also composite FBs on the official web sites of 4DIAC [*4DIAC Help*]. However, no complex documentation for FORTE sources exists, so also no documentation into the SIFBs, which were necessary for this implementation. Therefore the aim of this thesis is to provide the summary of the important information and approaches, which were used in the realisation of practical part of this thesis. The most common source of the information was the FORTE code analysis, consultation with authors of the FORTE project, or via discussion boards on the internet.

Default installation of 4DIAC framework already contains the library of FBs dedicated to the creating of simple, but also complex distributed controlling systems. Because of the simplier implementation in FORTE sources (differen groups of FBs requires different libraries, header files), and also higher clarity of FBs, the library is divided into the modules associating FBs with simmilar functionality. Module CONVERT, for example, contains FBs, which purpose is to convert variable from one type to another. The

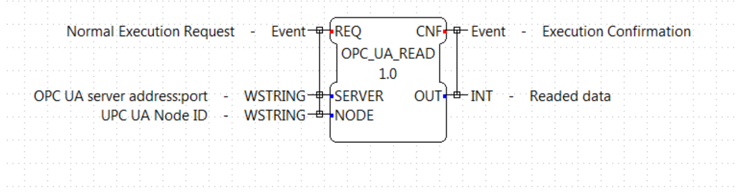

Normal Execution Request - Event
REQ
CNF
Event - Execution Confirmation
OPC_UA_READ
1.0
OPC UA server address:port - WSTRING
SERVER
OUT
INT - Readed data
UPC UA Node ID - WSTRING
NODE

FIGURE 4.2: OPC_UA_READ FB interface.

xample of FB from this module is FB called INT2BOOl, which converts integer variable into logic boolean variable using standard convention - value 0 for log. 0 and any other value for log. 1.

Designer, who decides to create a new FB, can add this FB into existing module of tool library - in case that the created FB fits logically and also with functionality in any existing module. Another recommended option, in the case developer creates whole family of function blocks, is to create own module.

The same way as the whole framework is divided into two main projects, which are developped by 4DIAC initiative, also the creation of modules and FBs is divided into the creation in 4DIAC IDE and also their interpretation in the runtime environment.

4.1.1 Creating module in 4DIAC IDE

Creating of the new module in 4DIAC IDE tool library means the creating of single folder in the file structure of this library. Folder in which the modules are stored in folder 'typelibrary' in root of 4DIAC project. Change in the library - the creation of a new module will have effect after the 4DIAC IDE restart. New created module will occur in every new created project.

Already existing projects content the own copy of this tool library form IDE in their workspace. It means that all new created modules need to be manually copied into this project library. However, this feature also gives a designer the oportunity to create module just for concrete project by creating a folder in this projects library in its own workspace. This feature is also necessary for the team development of 4DIAC projects. In case the project will not have its own library of FBs, it might happen, that every developer has different FBs in their libraries and it will lead to the disfunction of the project.

4.1.2 Creating module in FORTE

In the forte file structure, modules are interpreted as folders, too, and stored in /src/modules. Creating new module in forte sources means the creating single folder. However, just creating the folder does not mean that sources in this folder are included into the compilation process. Forte project uses CMake makefiles, so the next step in creating module is the setting propriet CMake files. First of all, all folders are mentioned in /src/modules/CMakeLists.txt, in which compiler looks for the sources. It is necessary to add this module folder into this list.

Another step to include this module into the forte compilation process is the creating of CMakeLists.txt inside the new module folder. The example of this file, with necessary and properly commented content is shown in the A.

4.1.3 Creating SIFB in 4DIAC IDE

As it was mentioned in the chapter dedicated to IEC 61499 and 4DIAC framework, 4DIAC IDE has a whole perspective dedicated to the creating and editing of FBs.

In the case of a SIFB creation, in this perspective only interface for connection with other FBs and properties needed for identification of FB (name, authorship) could be designed.

Every FB must have at least one input event and should have at least one output event (even when has no data output).

This part of the FB design is common for all kind of FBs. In 4DIAC help [*4DIAC Help*] is this part of design mentioned in Basic FB tutorial.

To define function of SIFB, there is need to export this interface into FORTE sources in 4DIAC IDE.

To show export dialog, click on the function block in the left window called Type Navigator with right button and select export option.

In the Export dialog, select 4DIAC Type Export and continue to select Export Destination.

In case you are creating new FB into existing module, select folder of this module in forte sources. In other case, follow instructions in the previous subsection.

4.1.4 Creating SIFB in FORTE

By exporting FB interface from 4DIAC IDE, IDE creates .c source and .h header file with prepared functions and definitions to handle triggered events, trigger output events, obtain the values of data inputs values and provide the values of data outputs.

For every input and output there is function handling its value defined in FB header file. For example, in case of input called property, to read this value in FB source code, only call of its function is needed.

value = PROPERTY();

Very important part of FB source is input event handler. In FB source exported from 4DIAC IDE function to handle this events is defined. This one function is dedicated to handle all input events. In parameter of this function is passed id of input event. There are symbolic names for this ids defined in FB header file.

Another very important part of FB source code is to trigger output events. To do this, just call function sendOutputEvent(event_id).

4.2 The second stage

In the second stage of integration, it was important to use the information model of OPC UA protocol in a development. The analysis of the required functions to be used preceeded the creating of OPC_UA_PUBSLISH and OPC_UA_SUBSCRIBE in order to use the potential of OPC UA in its full range.

As a part of this thesis, a new approach, in which every device runs its own OPC UA server, was developed. On this OPC UA server, a tree stucture is implemented where every published function block will be presented by one node and children of this node will represent published output values of this FB.

Developer publishes ouput of FB by connectiong simple OPC_UA_PUBLISH FB. This FB creates node with the unique name of FB as ID and saves value as variable child of this node.

Another resources, or devices, using OPC_UA_SUBSCRIBE FB can gain information from the server of the other devices, and also the information about structure and status of another devices.

4.3 The implementation of the FB OPC_UA_SUBSCRIBE

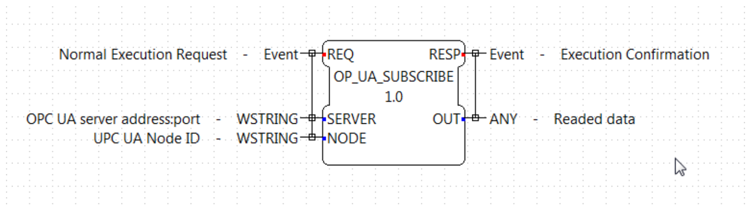

Normal Execution Request - Event
REQ
RESP
Event - Execution Confirmation
OP_UA_SUBSCRIBE
1.0
OPC UA server address:port - WSTRING
SERVER
OUT
ANY - Readed data
UPC UA Node ID - WSTRING
NODE

FIGURE 4.3: OPC_UA_SUBSCRIBE FB interface.

OPC_UA_SUBSCRIBE is a very simple FB. In its functionality, it tends to follow OPC_UA_READ FB. In OPC_UA_SUBSCRIBE inputs, developer defines remote server to which DB connects, name of FB and its output he wants to subscribe. When input event triggers, FB connects to OPC_UA server, it reads the value of selected node output and readed value is set into the output buffer of FB and it triggers output event.

In case of error during reading from the remote server, FB triggers output error event, which can developer detect and treat. This feature of OPC_UA_SUBSCRIBE FB is very important in the real industrial systems. Invalid data on output, without a possibility to treat these data the special way, may lead to the destruction of the product, or even the machine.

4.4 The implementation of the FB OPC_UA_PUBLISH

In comparison with the FB OPC_UA_PUBLISH, which is with the exception of the function for the asynchronous subscribe practically the same as the FB OPC_UA_READ, the new FB OPC_UA_PUBLISH is much different from its predecesor. The functions of this FB consist of various parts. OPC_UA_PUBLISH detects the connection and from the information gained about the FB connected to the OPC_UA_PUBLISH, it finds or creates the node in the informational moded of the server adn it writes the actual value of the connected item.

The activity of OPC_UA_PUBLISH can be divided into four important divisions:

- Information model exploration
- Node representation of FB creating
- Writing of the values
- Deleting node representaion of the FB

First three divisions create an input event with every trigger action. As one of the most importat functions of the 4DIAC framework is the support of the reconfiguration of the FB with the OPC_UA_PUBLISH connected, and it can change between the two event triggers, the parameters of the FB and of the information model have to be checked with every different input event and when needed, nodes in the information model have to be created.

The information model exploration is the most complicated and time consuming part of the function of this FB. This function is important because of the knowing if the node for data input exists or not. During the flow through the nodes of the information model, ID nodes are compared to the name of the FB connected to the OPC_UA_PUBLISH. It is possible to save the list of these nodes directly to the implementation of this FB. However, any duplicity of data causes the inconsistency of the data in case of deleting, renaming of the node on the server. These changes are not detectable, therefore it is difficult to find out how often the list needs to be renewed.

The last part - a deleting node representation is done before the deleting of the FB OPC_UA_PUBLISH.

Deleting of nodes is necessary because variable type may be changed during the proces of reconfiguration. This may cause, that variable type in

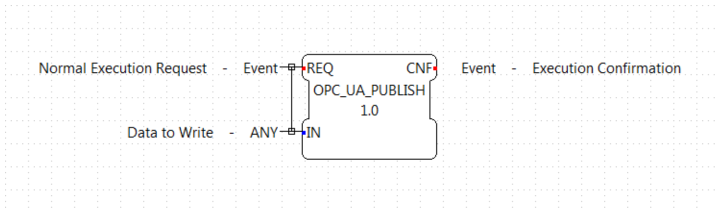

Normal Execution Request - Event
REQ
CNF
Event - Execution Confirmation
OPC_UA_PUBLISH
1.0
Data to Write - ANY
IN

FIGURE 4.4: OPC_UA_PUBLISH FB interface.

4DIAC application would differ from variable type on OPC UA server. Detection of this process is not possible in current version of FORTE. Most secure solution is to delete and create variable node on OPC UA server every time, instead of trying to write value into this node.

This approach brings out also the risk of the inconsistency in data. In case of the renaming of the FB or of its item connected to the OPC_UA_PUBLISH, this FB does not detect the change and it will not delete the node representing the connected FB in the information model, but it will create the node according to the actual FB.

As the actual version of the runtime environment does not allow any mean how to detect such a change in the functional implementation of the OPC_UA_PUBLISH, the solution for the developpers would be to delete and recreate the FB of the OPC_UA_PUBLISH within each reconfiguration of the FB network.

4.5 The implementation of OPC UA Server

The solution, coming at first sight from the IEC 61499 standard, positions the implemenation of OPC UA server into the SIFB source. This solution is intuitive, because all functions of application should be placed into the FBs and runtime environment should only transfer data and events among FBs. The main advantage of this solution is that the developer has the absolute control over the server. Disadvantage is the possibility to input more FB of the OPC UA Servers to one appliction. This solution is neither developer friendly nor ergonomic. In case that the developer decides to publish the values into th server, need to use the OPC UA server FB occurs. This solution is the most straightforward for the implementation. However, according to cons mentioned above, this approach was not chosen in this thesis.

Another approach is to position the OPC UA server to the implementation of the resource in runtime environment Forte. This solution comes from the idea of the absolutely independant resources in the system IEC 61499. This server would exists in every resource.

This approach seems ideal from the point of IEC 61499 standard and also from developer ergonomy. However, the biggest disadvanteg of this approach is the possibility of more resources running on 1 device with the same IP address. That means also more servers running on 1 device, which causes a problem with addressing of more servers. This problem is being solved in the net OSI ISO model in the transport layer y introducing ports. In the chapter dedicated to OPC UA, there is a note, why it is convenient, that OPC UA server runs on the standard HTTP port 80.

The problem with multiple servers on 1 device is solved by the implementation of the OPC UA server into the device. In every device, there is the OPC UA server implemented. This approach eliminates the problem with the need for the setting of the server ports in each resource. Also this approcah has its disadvantages. As the resources containing the FB of the OPC_UA_PUBLISH are independant from the device, the approach defined in the IEC 61499 allows the switching off and restart of one of the resources in the device without having the affect on other resources or the device itself. After the switching off the resource, the server still contains the nodes interpreting the FBs in the source, which is already switched off. This problem can be solved in a so-called destructor function of the OPC_UA_PUBLISH. This function is automatically triggered before ending of the FB and OPC_UA_PUBLISH deletes the nodes in the information model of the server.

The OPC UA server itself is just simple implementation created using tutorial on open 62541 website. [*First steps with open62541-server*] The server started in the device runtime is just blank server with default data model. There is no need to create any nodes in the moment the server is starting, because OPC_UP_PUBLISH creates all needed nodes.

All sources created while working on this thesis, and also needed changes in the FORTE sources are available at public GitHub repository. [*Integration of IEC 61499 with OPC UA source codes*]

Chapter 5

Conclusion

The objective of this thesis is the integration of IEC 61499 with OPC UA. This objective was met by creating FBs in 4DIAC framework dedicated to transmitting data using the OPC UA communication protocol. These FBs use OPC UA information model to exchange data among 4DIAC application and also among 4DIAC application and other industrial application.

The aim of this integration is to transform industrial control systems from the standard pyramide topology mentioned in the first chapter, into the reconfigurable systems. 4DIAC, framework created according to the IEC 61499, allows a developer to create an industrial control application and brings them the feature of reconfiguration this application by creating and modifying FBs and connections among them. However, this reconfigurable application runs on only one industrial pyramide layer. The problem of this reconfiguration in control application, is that the systems on other layers do not react to this reconfiguration, because they are not capable of automatic detecting of this kind of reconfiguraton. This problem was solved by using OPC UA as data sharing protocol. The information model of OPC UA which allows not only to store data, but also to store them in the structured web of interconnected nodes.

The integration of these two standards developed during working on this thesis, came with the solution to the problem. Using UPC_UA_PUBLISH developer shares not only value, but also the structure of FBs with other systems in network. This brings new possibilities to the systems on other layers of the pyramide with detecting reconfiguration in 4DIAC application. Also, it replaces the static way of transfering data between layers. Using OPC_UA module in 4DIAC framework allows other layers to read any data from 4DIAC application developer decides to publish. Using the one large information model for all the layers of the application results in merging layers into one complex system.

Solution created during working on this thesis is platform independent. 4DIAC framework can be executed on every platform (even 16b computers). In this solution, the usage of the OPC UA stack is limiting. Open62541 stack is available for Windows, OSX and Linux (debian, ubuntu). That means, that these are also the systems on which this solution works. The solution was developped on the Microsoft Windows platform , it was also compiled and executed on debian linux. The ability to run this solution on debian means, that it can be run also on most of small open source computers, like RaspberryPi, BeagleBone, CubieBoard, which use debian.

In the assignment of the work,creating a configuration dialog in 4DIAC IDE is included. This dialog was mentioned to configure OPC_UA_WRITE FB's target to write data. However, during the work on the thesis, the approach of publishing data used in the OPC_UA_PUBLISH FB was developed, which has no need for the configuration, because it gains the information from the structure of application.

The thesis fulfilled the assignment. However, there are also other posibilities how to improve a IEC 61499 and OPC UA integration. OPC UA has many features, which were not used in this thesis.

First of all, OPC UA includes techniques of the authentification and the autorization. The increase of the security is necessary to allow this integration usage in the real control systems.

OPC UA contains the ability of asynchronous subscription. Server can announce client, value on server has been changed. This feature could be used in the future version of OPC_UA_SUBSCRIBE FB. FB with enabled assynchronous subscription would automatically wait for this announcement from server, download actual value and trigger output event. In this use case, there is no need to trigger input event.

Another important feature of the OPC UA is the method calls. In some cases, this methods executed on the OPC UA server may process the data instead of complicated networks of the function blocks. The advantage of the creating method calls is also in the fact, that methods can be executed also from de remote device and can be re-used by another systems connected to the same OPC UA server.

Appendix A

Installation of 4DIAC-IDE

A.1 4DIAC IDE installation

The installation of 4DIAC-IDE is independent from the used operating system. In order to run 4DIAC-IDE you require Java 1.7 SDK or later, whereas it is currently NOT recommended to use Java 8.

To install 4DIAC-IDE you simply download the latest version for your operating system from https://eclipse.org/4diac/. Unzip it to any desired folder and start the 4DIAC-IDE. It already contains a function block library, some sample applications and also pre-build versions of FORTE. If you only want to use available Function Blocks you are ready to go.

Building your own 4DIAC-IDE from source: Running 4DIAC-IDE from source has the great advantage that you can easily keep up with the developments performed in the Mercurial repository. In case you want to run 4DIAC-IDE from source follow the Installation steps at [*4DIAC Help*].

A.2 FORTE compilation

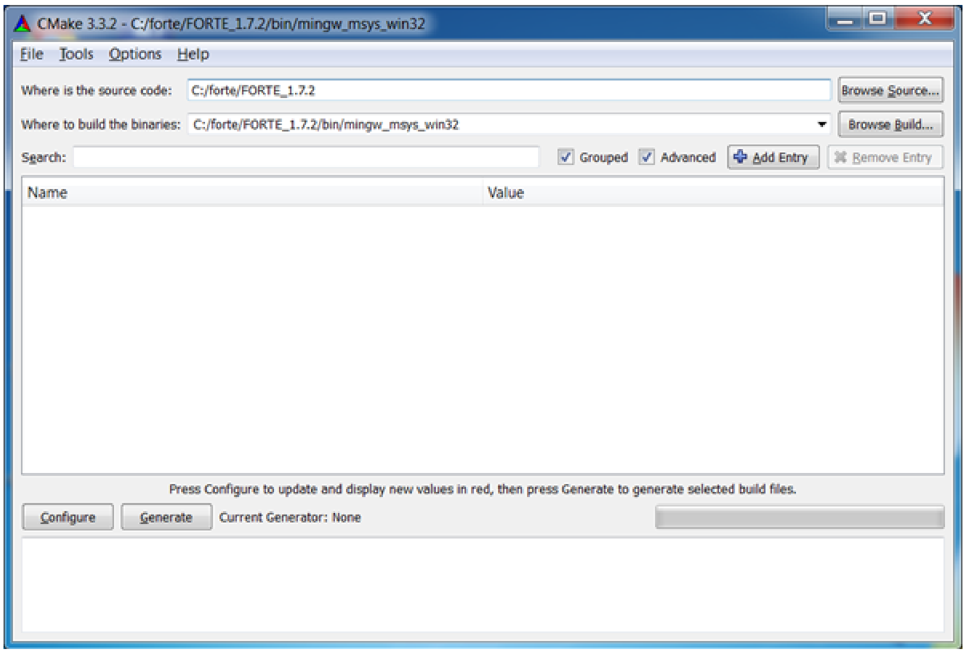
For conducting first experiments with 4DIAC, you can use the pre-build version of FORTE which comes along in the runtimes directory of the 4DIAC-IDE package. However if you want to develop your own Function Blocks or you want to run FORTE on different control devices you have to download and build FORTE from source.

The compiling and debugging of FORTE consists of few steps:

Download source code You can download the latest Version of FORTE on http://www.eclipse.org/4diac/. Extract the file into your desired working directory. You can also use Mercurial Hg like TortoiseHG to get FORTE from http://hg.code.sf.net/p/fordiac/forte.

Prepare compilation and linking tools In case you want to create own function blocks, or edit existing one you are going to compile your own version of FORTE. According to your operating system, you have several options to choose. In Linux – like systems required packages to compile are:

- binutils



CMake 3.3.2 - C:/forte/FORTE_1.7.2/bin/mingw_msys_win32

Where is the source code: C:/forte/FORTE_1.7.2

Where to build the binaries: C:/forte/FORTE_1.7.2/bin/mingw_msys_win32

Search:

✓ Grouped ✓ Advanced

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: None

FIGURE A.1: Selection of source data and output folder

- gcc
- gdb
- make

In case you are Mac user, you can compile FORTE in X-Code. Compiling on Windows is more complicated. There are few possibilities:

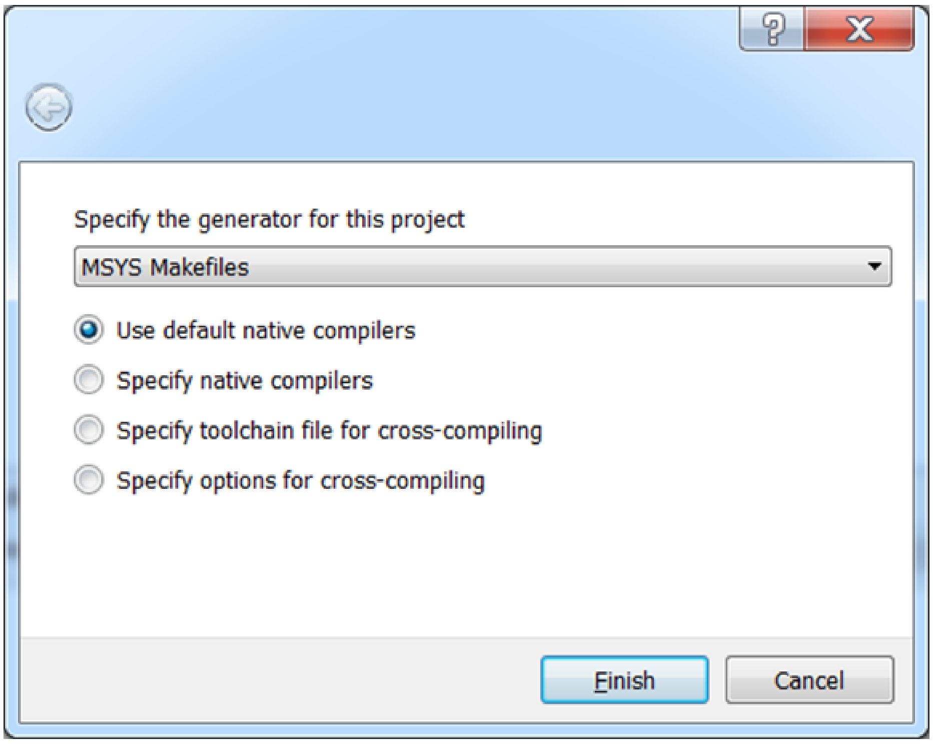
- Compiling and Debbugging FORTE with MS Visual Studio Express
- Compiling using Cygwin
- Compiling using MinGW – I have used this option. Whole subsection is dedicated to this option

CMake for generating the make file

CMake helps you to configure FORTE for compilation with your desired development environment or hardware device. For starters, it is recommended to use the GUI tool that comes with CMake.

When starting the CMake-GUI you have to select the source directory, which is the main FORTE directory and the bin directory (e.g.FORTE/bin/posix) which is the output directory. There CMake will put the build project files (e.g., the makefiles) as well as any configuration data.

After that you will need to press the configuration button. A window will pop up that lets you to select the kind of project you would like to build. In this step, you have to have installed compillers. Select MSYS Makefiles as the generator for this project.


Specify the generator for this project

MSYS Makefiles

- [✓ Use default native compilers]
- Specify native compilers
- Specify toolchain file for cross-compiling
- Specify options for cross-compiling

FIGURE A.2: Specifing the generator

Windows	**POSIX**
FORTE_ARCHITECTURE_WIN32	FORTE_ARCHITECTURE_POSIX
FORTE_MODULE_CONVERT	FORTE_MODULE_CONVERT
FORTE_MODULE_IEC61131	FORTE_MODULE_IEC61131
FORTE_MODULE_OPC_UA	FORTE_MODULE_OPC_UA
FORTE_MODULE_Test	FORTE_MODULE_Test
FORTE_MODULE_UTILS	FORTE_MODULE_UTILS
FORTE_SUPPORT_MONITORING	FORTE_SUPPORT_MONITORING

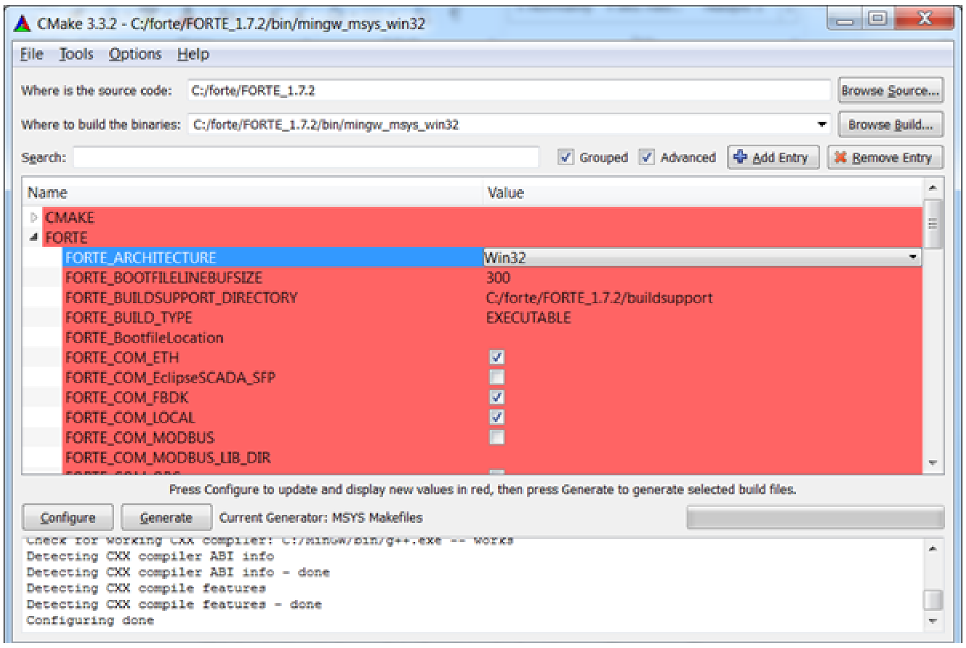
For the correct Project Setting please have a look at the next step. In the CMake main window a list of red marked options will appear. These options allow you to configure your FORTE build. The minimal configuration you have to perform is to select the architecture you like to build for (e.g., FORTE_ARCHITECTURE to POSIX / WIN32) and the modules with the function block libraries you like to use. You should also keep FORTE_SUPPORT_MONITORING enabled for Debugging and FB-Testing.

Then you need to press again the configure button and depending on your selection in the previous step new options (marked in red) may appear. Press configure until no new options are appearing and then the generate button for generating the project files.

After that you can start the build process.

Configuration of CMake for different OS:

IDE to work with the FORTE code You can use different development Environments, whereas the C++ Compiler you can use to build FORTE not



Where is the source code: C:/forte/FORTE_1.7.2

Where to build the binaries: C:/forte/FORTE_1.7.2/bin/mingw_msys_win32

✓ Grouped ✓ Advanced

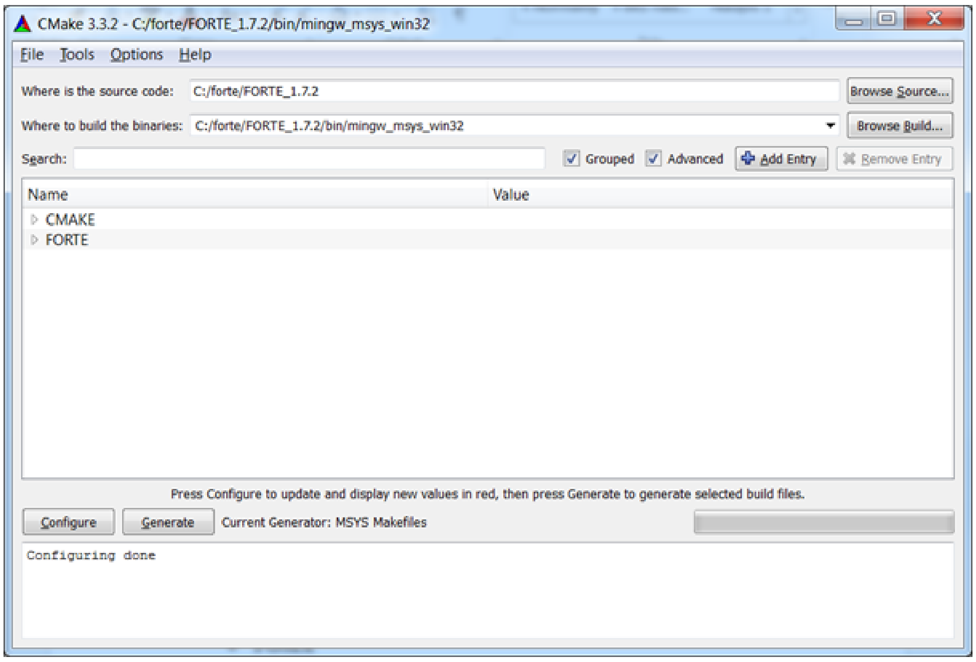
Name	Value
CMAKE	
FORTE	
FORTE_ARCHITECTURE	Win32
FORTE_BOOTFILELINEBUFSIZE	300
FORTE_BUILDSUPPORT_DIRECTORY	C:/forte/FORTE_1.7.2/buildsupport
FORTE_BUILD_TYPE	EXECUTABLE
FORTE_BootfileLocation	
FORTE_COM_ETH	✓
FORTE_COM_EclipseSCADA_SFP	☐
FORTE_COM_FBDK	✓
FORTE_COM_LOCAL	✓
FORTE_COM_MODBUS	☐
FORTE_COM_MODBUS_LIB_DIR	

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: MSYS Makefiles

```
Check for working CXX compiler: C:/MinGW/bin/g++.exe -- works
Detecting CXX compiler ABI info
Detecting CXX compiler ABI info - done
Detecting CXX compile features
Detecting CXX compile features - done
Configuring done
```

FIGURE A.3: Configuring architecture of compilled FORTE



Where is the source code: C:/forte/FORTE_1.7.2

Where to build the binaries: C:/forte/FORTE_1.7.2/bin/mingw_msys_win32

✓ Grouped ✓ Advanced

Name	Value
CMAKE	
FORTE	

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: MSYS Makefiles

```
Configuring done
```

FIGURE A.4: Configuration done.

only depends on this environment but also on your operating system. For compiling FORTE under Windows you can use either Visual Studio (Express or full edition) or Eclipse. When using Eclipse for development and debugging under Windows you will need to use a Posix emulation environment like cygwin or minGW.

- Compiling and Debugging FORTE with MS Visual Studio Express
- Compiling and Debugging FORTE with Eclipse
- Compiling and Debugging FORTE with CodeBlocks

For the development with FORTE, the understanding of the general file structure is helpful. Therefore the essential parts as well as the Makefiles which are important for the configuration and compilation of FORTE are listed in the following:

- *src/modules* this folder contains the source code (cpp, h) of all Function Blocks available for FORTE
- *bin/<yourSystem>/src* contains the forte executable after compilation with Makefile all
- *bin/<yourSystem>/src_gen* contains the object files generated during compilation with Makefile all
- all this Makefile generates the object files for all FORTE core files and Function Block source code files
- clean this makefile removes all generated object files.

A.3 Installing and Settipg up MinGW for FORTE Development

Download and install MinGW from http://www.mingw.org/ Launch MinGW installer and install default setting and add following packages:

- Mingw32-gcc
- Mingw32-gcc-g++
- mingw32-make
- msys-make
- mingw32-libz (newer version of windows doesn't include libraries)
- mingw32-gmp (newer version of windows doesn't include libraries)

After installing, go to the Control Panel/System/Advanced/Environment Variables. Change PATH variable (click on it) add path where your MinGW binaries have been installed in e.g., C:\MinGW\bin\;. Add C:\MinGW\bin; C:\MinGW\msys\1.0\bin; in the Windows file PATH. **Test MinGW** Open command prompt window by pressing Windows button and entering cmd. Enter bash, if bash prompt appears it was successful.

Appendix B

Appendix Title Here

Example of CMakeLists.txt in module folder.

```
//name and description of module
forte_add_module(OPC_UA "OCP Unified Architecture Function Blocks")

############################################
# OPC UA FB
############################################
forte_add_include_directories(${CMAKE_CURRENT_SOURCE_DIR})

forte_add_sourcefile_hcpp(OPC_UA_READ)
forte_add_sourcefile_hcpp(OPC_UA_WRITE)
forte_add_sourcefile_hcpp(OPC_UA_PUBLISH)
forte_add_sourcefile_hcpp(OPC_UA_SUBSCRIBE)


//include posix libraries
if("${FORTE_ARCHITECTURE}" STREQUAL "Posix")
        forte_add_link_library( libmodbus.so )
        forte_add_include_directories( ${CMAKE_CURRENT_SOURCE_DIR} )
        forte_add_link_directories( ${CMAKE_CURRENT_SOURCE_DIR} )

//include Win32 libraries
elseif("${FORTE_ARCHITECTURE}" STREQUAL "Win32")
                forte_add_link_library( libopen62541.dll )
        forte_add_include_directories( ${CMAKE_CURRENT_SOURCE_DIR} )
        forte_add_link_directories( ${CMAKE_CURRENT_SOURCE_DIR} )
endif()
```

Bibliography

4DIAC Help. URL: http://www.eclipse.org/4diac/documentation/help.html.

Alonso, Gustavo et al. (2004). "Web Services". English. In: *Web Services*. Data-Centric Systems and Applications. Springer Berlin Heidelberg, pp. 123–149. ISBN: 978-3-642-07888-0. DOI: 10.1007/978-3-662-10876-5_5. URL: http://dx.doi.org/10.1007/978-3-662-10876-5_5.

Brettel, Malte et al. (2014). "How virtualization, decentralization and network building change the manufacturing landscape: An Industry 4.0 Perspective". In: *International Journal of Science, Engineering and Technology 8 (1), 37* 44.

Commission, International Electrotechnical et al. (2005). "IEC 61499-1: Function Blocks-Part 1 Architecture". In: *International Standard, First Edition, Geneva* 1.

First steps with open62541-server. URL: http://open62541.org/doc/current/tutorial_server_firstSteps.html.

Hannelius, T., M. Salmenpera, and S. Kuikka (2008). "Roadmap to adopting OPC UA". In: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pp. 756–761. DOI: 10.1109/INDIN.2008.4618203.

Integration of IEC 61499 with OPC UA source codes. URL: https://github.com/slavokozar/Integration-of-IEC61499-with-OPC-UA.

OPC UA stacks overview. URL: http://www.opcconnect.com/uakit.php#overview.

OPEN61541 stack building. URL: http://open62541.org/doc/current/building.html.

Saad, Sameh M. (2003). "The reconfiguration issues in manufacturing systems". In: *Journal of Materials Processing Technology* 138.1–3. {IMCC2000}, pp. 277 –283. ISSN: 0924-0136. DOI: http://dx.doi.org/10.1016/S0924-0136(03)00085-2. URL: http://www.sciencedirect.com/science/article/pii/S0924013603000852.

Strasser, T. et al. (2008). "Framework for Distributed Industrial Automation and Control (4DIAC)". In: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pp. 283–288. DOI: 10.1109/INDIN.2008.4618110.

Sunder, C. et al. (2008). "Considering IEC 61131-3 and IEC 61499 in the context of component frameworks". In: *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pp. 277–282. DOI: 10.1109/INDIN.2008.4618109.

Zoitl, Alois (2008). *Real-Time Execution for IEC 61499*. ISA. ISBN: 1934394270, 9781934394274.