CZECH TECHNICAL UNIVERSITY

BACHELOR THESIS

Integration of IEC 61499 with OPC UA

Author:
Slavomír K

Supervisor:
Ing. Petr KADERA PhD.

in the

December 13, 2015

Declaration of Authorship

I, Slavomír K, 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 research degree at this University.
- 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.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

CZECH TECHNICAL UNIVERSITY

Abstract

Faculty of Electrical Engineering

Bachelor

Integration of IEC 61499 with OPC UA

by Slavomír K

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too…

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or

Physical Constants

Speed of Light $c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

For/Dedicated to/To my…

Chapter 1

Introduction

Nowadays we are standing in the time, when the fourth industrial revolutions starts.Every one of these revolutions were caused by technological improvements. First one was caused by change from labor work to mechanization. Second one was started by electrification, in this revolution electric machines were used instead of steam based motors. Third revolution was the last one, and was caused by digitization and invention of logical circuits. When we realize how much did the computers evolved it's logical that also industry has to pass another revolution. Upcoming revolution is caused by introducting Internet of Things into industry. Brettel et al., 2014

1.1 Reconfiguration

Today in the time of fast changes on global market and decreasing lifetime of product, industry is forced into philosophical shift. Manufacturing thave to be quickly moved from mass production into mass customization. In order to rise to the challenge of there trends, new operation methods are necessary. Production facilities need to be flexible, adaptable and allow fast changes at little cost. Flexible production systems nowadays comes with higher cost, because these plants has higher initial cost, but even mor important are costs of down-times needed to reconfigure such plant. Reconfiguration without need to stop production is necessary. This needs reconfiguration of manufacturing plants on all levels, even the physical reconfiuration. Whatever the solution, it must be simple, flexible, and have limited space requirements. For example one of the simpliest approaches, changing the parameters of software components leads to large program that must count with any possible change combination in advice.

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

On the figure below change needed to be done in industry is shown. In current approach control system is divided into layers which are horizontally configurable and connection of layers are declared static way. Reconfiguration of this kind of control system needs to rebuild the whole pyramide from the bottom. While the new approach with also vertical and horizontal configurable.

1.2 Aim of thesis

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

Controll system created in 4DIAC framework is composed of function blocks, my task is implement communication stack inside of function blocks. Including client and also server.

OPC UA protocol allows user to create topologically ordered web of data. My task is also to create data topology on server based on a structure of control system based on a 4DIAC framework. By integration of these two technologies I create 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

In following second chapter my aim is to introduce you a IEC 61499 standard and 4DIAC framework based on this standard. I am going to show basic principles of this framework as creating application, function blocks, deploying applications. Important part of using 4DIAC framework is compiling of your own version of 4DIAC runtime environment dedicated for your device. To this topic is dedicated whole section. Another section of this chapeter will be dedicated to compiling and runnig 4DIAC runtime on raspberry pi.

Third chapter is dedicated to communication protocol OPC UA, ways of using this protovol and its information model. I am going to mention stacks based on OPC UA protocol. I am focusing on OPEN 62541 stack, which i have choosen to use in this thesis.

In fourth chapter I am explaining my solution of problem explained in the previous sections of this chapter. Also I am describing example application to work with OPC UA in 4DIAC.

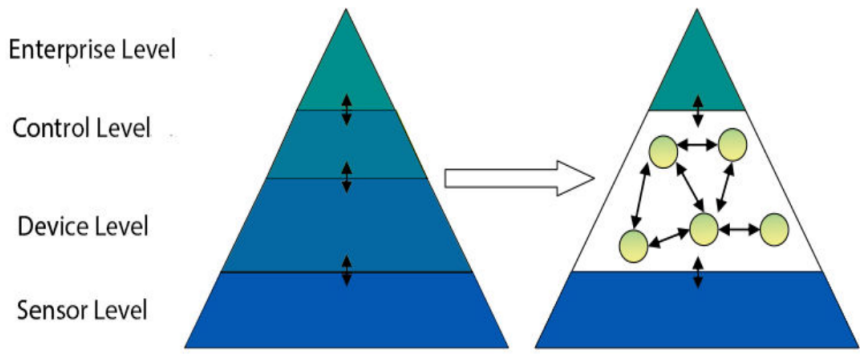

FIGURE 1.1: sfasdfasdf

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 purpose of all parts of this family is to define Function Block (FB), so in this Thesis term IEC 61499 references whole family of these standards. IEC 61499 is based on an older IEC 61311 (1993) family of standards, which is most widely adopted standard in domain of IPMCS. //citovat Aloisa 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 allows developer to create a distributed control application in graphical manner. This brief introduction will give you insight into the IEC 61499 standard for purposes of this thesis. A full description of architecture may be found in IEC61499-1.

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 Fucntion Block (FB) model.

The application model consist of multiple system models, the one of these system models consist of multiple device models etc.

The Base and most important model of IEC 61499 is FB. FB is independent, self-contained software component with interface through which it provides specific functions. This model was taken from IEC 61131-3 standard. Against IEC 61131-3 FB definition in IEC 61499 event interface is added. The function block function is triggered by one of 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 cyclic execution.

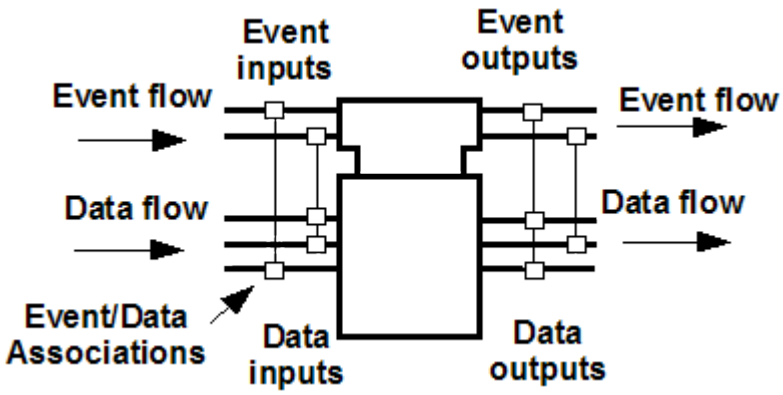

FIGURE 2.1: http://www.automation.com/images/isa$_automation_week/IEC61499FunctionBlock.p

Cyclic execution was problem that 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 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 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 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 event. This solution prevents problem with central time and its sharing and caused also rapid decrease of needed bandwidth. In this approach the data are transferred only when 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 cyclic execution by 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

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.

Then 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. //citovat IEC 61499-1

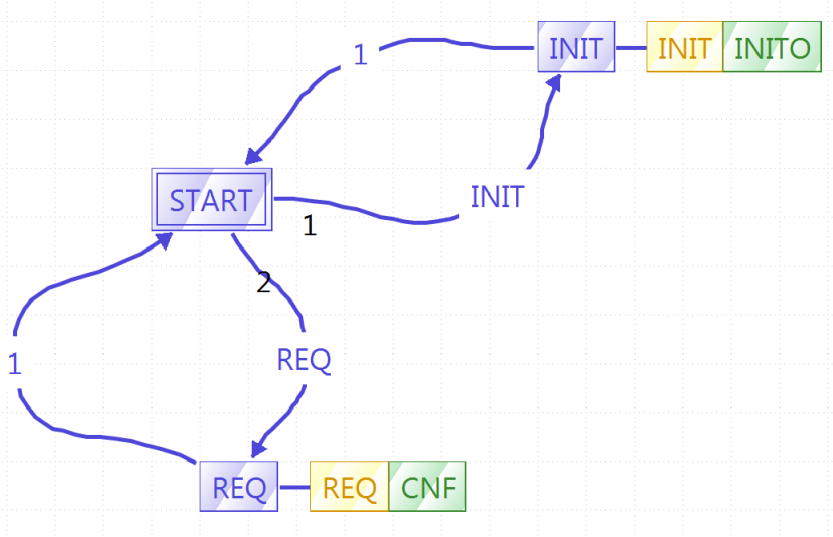

FIGURE 2.2: Example of basic FB's ECC

2.2.2 Composite FB

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 event and data connections are connected to the internal FBs and also outgoing connections are connected to internal FBs.

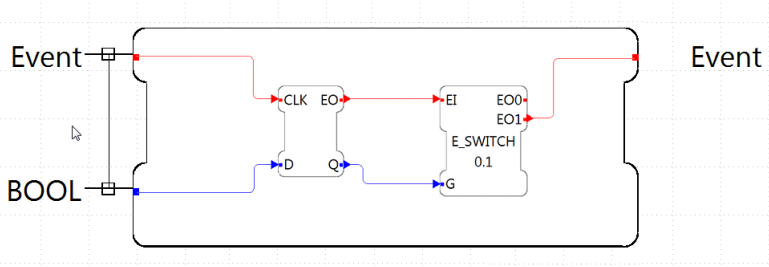

FIGURE 2.3: Example of composite FB structure

2.2.3 Service Interface FB

Service Interface FB is dedicated to function out of scope of IEC 61499. Typical function is 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 is an 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 dividet into two phases. In the first phase designer creates Functin Block Network by interconnecting of FBs with data and event connection. In this phase developer has in mind only 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.

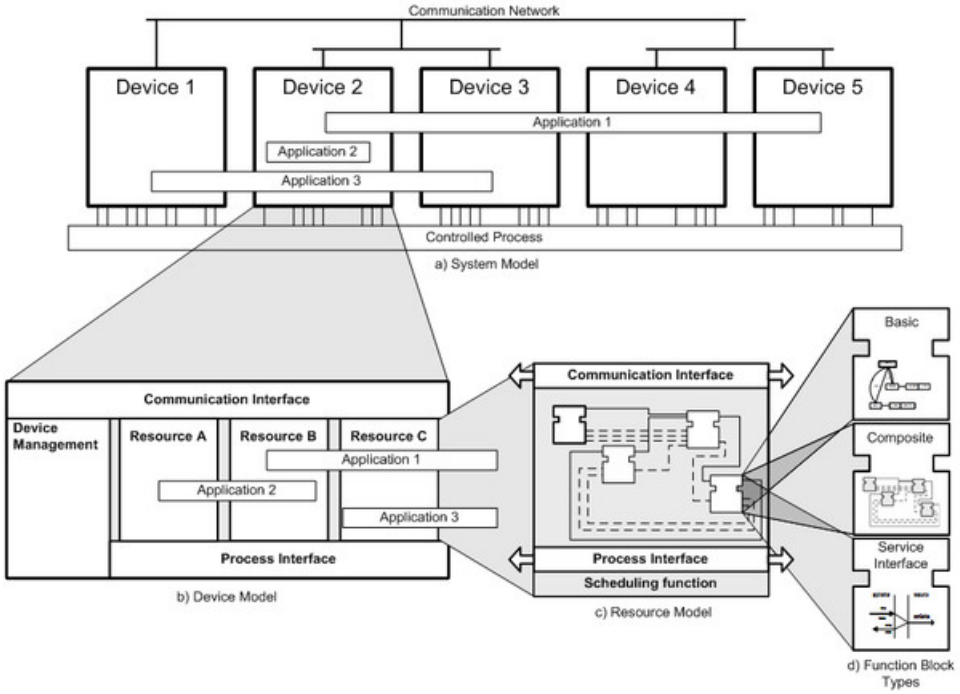

FIGURE 2.4: http://2.bp.blogspot.com/-qzJhgeMHSnQ/UNZC7b0op6I/AAAAAAAALwk/aW3FpwY3e5o/s1600/IEC+61499+Models.jpg

IEC 61499 is executed on devices. Every device consist 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 contains applications or parts of applications. Resources in device are independend. This means resources 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. Task of resource is to provide execution environment, delivering event notifications.

2.4 IEC 61499 applications

The current application of IEC 61499 can be devided into research and industrial sector. IEC 61499 standard exist sinc January 2005. Before standardization since 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 academin or, if industrially-based has resulted only in prototypical 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 starting point wa FBDK/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 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 was never 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 ans supported by international automation network O[1]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.

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.

[1] 3

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

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.

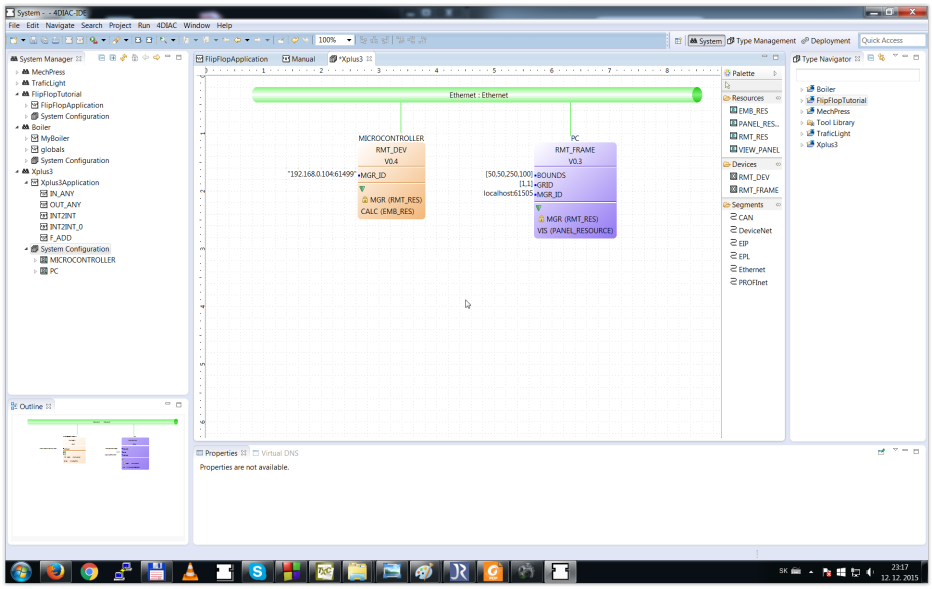

FIGURE 2.5: Perspective dedicated to basic creation of application

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 figure below are shown tools which can be applied on function block. In case of basic FB you can edit also function of this FB by editing its EEC or Algorithm writen in pseudocode. Function of 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 editin forte source.

All changes made in Type Management Perspective have to be exported into 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 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.

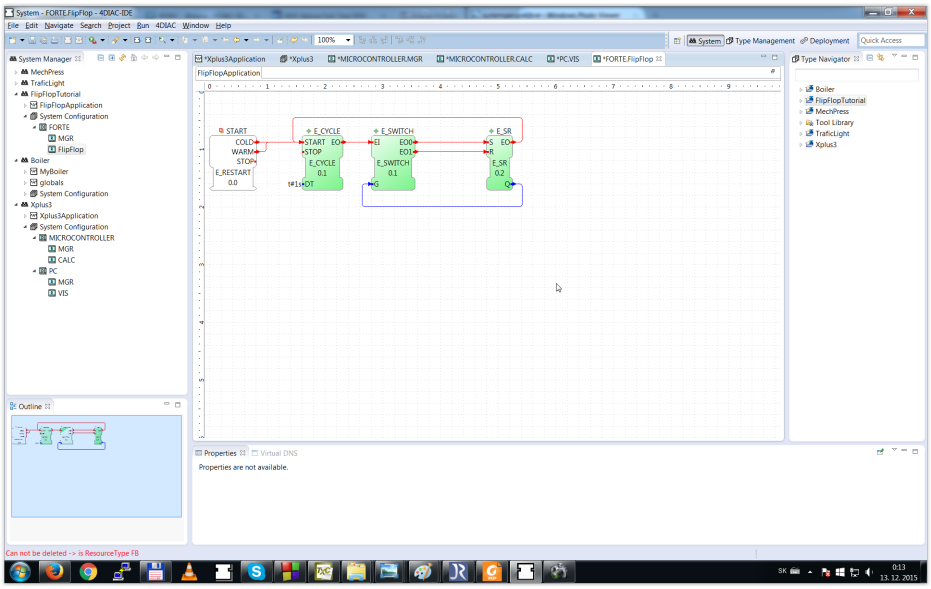

FIGURE 2.6: Editing Function Block Network in resource

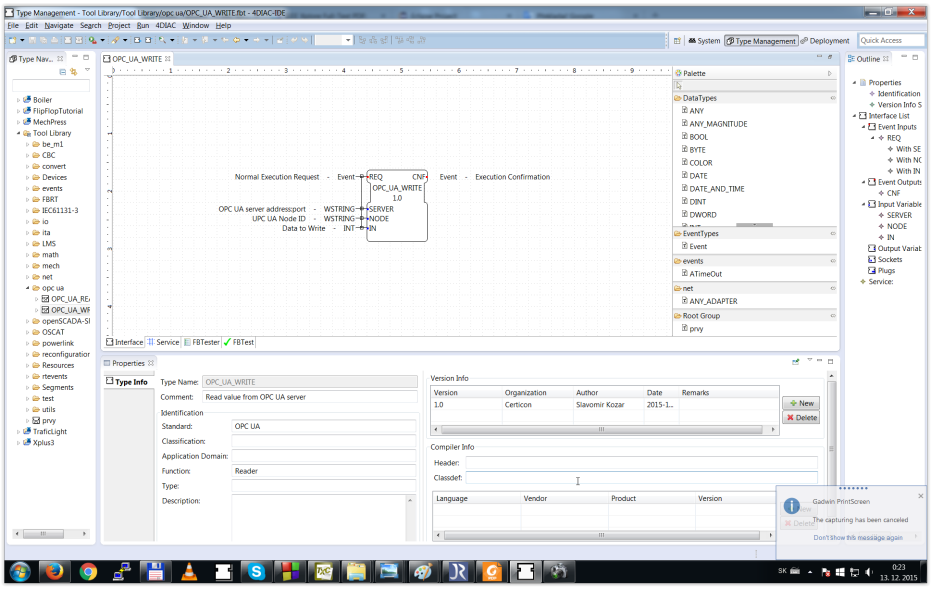

FIGURE 2.7: Editing or Creating FBs

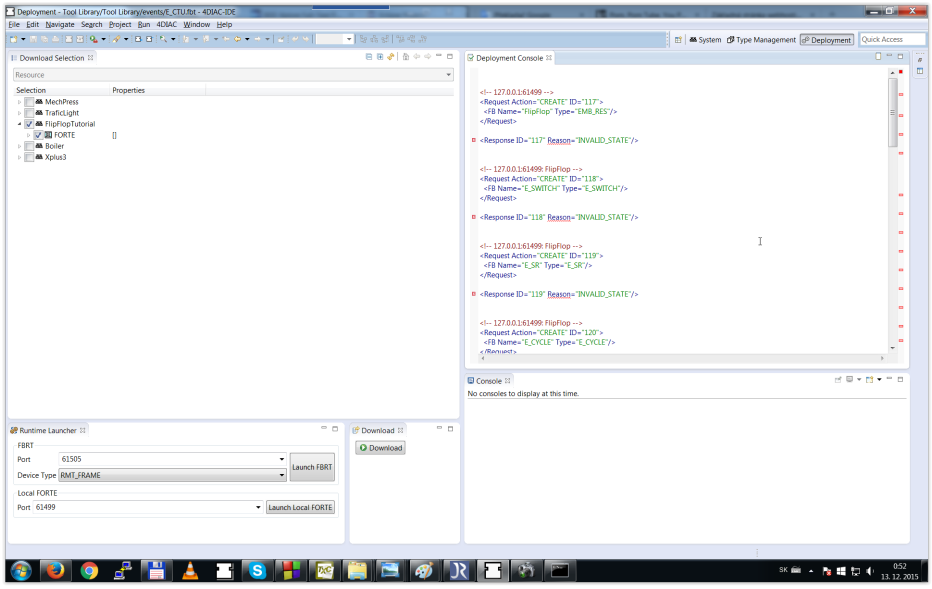

FIGURE 2.8: Uploading application into devices

2.7 4DIAC

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 :

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

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

2.8 Function blocks

Chapter 3

OPC UA

3.1 What is OPC UA

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultricies lacinia euismod. Nam tempus risus in dolor rhoncus in interdum enim tincidunt. Donec vel nunc neque. In condimentum ullamcorper quam non consequat. Fusce sagittis tempor feugiat. Fusce magna erat, molestie eu convallis ut, tempus sed arcu. Quisque molestie, ante a tincidunt ullamcorper, sapien enim dignissim lacus, in semper nibh erat lobortis purus. Integer dapibus ligula ac risus convallis pellentesque.

3.2 Data structure

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

3.3 OPEN 62541 stack

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu neque.

3.3.1 compiling

3.3.2 first use

3.4 Other stacks

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus

ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

Chapter 4

Results

4.1 Created function blocks

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam ultricies lacinia euismod. Nam tempus risus in dolor rhoncus in interdum enim tincidunt. Donec vel nunc neque. In condimentum ullamcorper quam non consequat. Fusce sagittis tempor feugiat. Fusce magna erat, molestie eu convallis ut, tempus sed arcu. Quisque molestie, ante a tincidunt ullamcorper, sapien enim dignissim lacus, in semper nibh erat lobortis purus. Integer dapibus ligula ac risus convallis pellentesque.

4.2 Implementation of function blocks

4.2.1 server

Nunc posuere quam at lectus tristique eu ultrices augue venenatis. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam erat volutpat. Vivamus sodales tortor eget quam adipiscing in vulputate ante ullamcorper. Sed eros ante, lacinia et sollicitudin et, aliquam sit amet augue. In hac habitasse platea dictumst.

4.2.2 subscriber

Morbi rutrum odio eget arcu adipiscing sodales. Aenean et purus a est pulvinar pellentesque. Cras in elit neque, quis varius elit. Phasellus fringilla, nibh eu tempus venenatis, dolor elit posuere quam, quis adipiscing urna leo nec orci. Sed nec nulla auctor odio aliquet consequat. Ut nec nulla in ante ullamcorper aliquam at sed dolor. Phasellus fermentum magna in augue gravida cursus. Cras sed pretium lorem. Pellentesque eget ornare odio. Proin accumsan, massa viverra cursus pharetra, ipsum nisi lobortis velit, a malesuada dolor lorem eu neque.

4.3 Example application

4.4 Main Section 2

Sed ullamcorper quam eu nisl interdum at interdum enim egestas. Aliquam placerat justo sed lectus lobortis ut porta nisl porttitor. Vestibulum mi dolor, lacinia molestie gravida at, tempus vitae ligula. Donec eget quam sapien, in viverra eros. Donec pellentesque justo a massa fringilla non vestibulum metus vestibulum. Vestibulum in orci quis felis tempor lacinia. Vivamus

ornare ultrices facilisis. Ut hendrerit volutpat vulputate. Morbi condimentum venenatis augue, id porta ipsum vulputate in. Curabitur luctus tempus justo. Vestibulum risus lectus, adipiscing nec condimentum quis, condimentum nec nisl. Aliquam dictum sagittis velit sed iaculis. Morbi tristique augue sit amet nulla pulvinar id facilisis ligula mollis. Nam elit libero, tincidunt ut aliquam at, molestie in quam. Aenean rhoncus vehicula hendrerit.

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 https://eclipse.org/4diac/documentation

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
- gcc
- gdb
- make

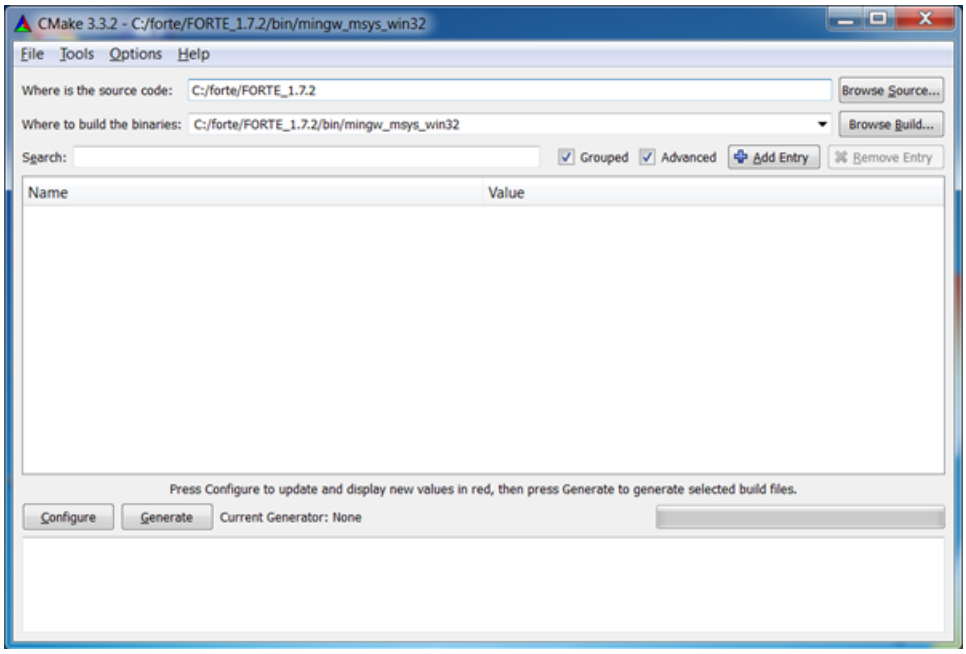

FIGURE A.1: Selection of source data and output folder

In case you are Mac user you can compile FORTE in X-Code. Compiling on Windows is most 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 we recommend 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 select the kind of project you like to build. In this step you have to have installed compillers. Select MSYS Makefiles as the generator for this project.

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.

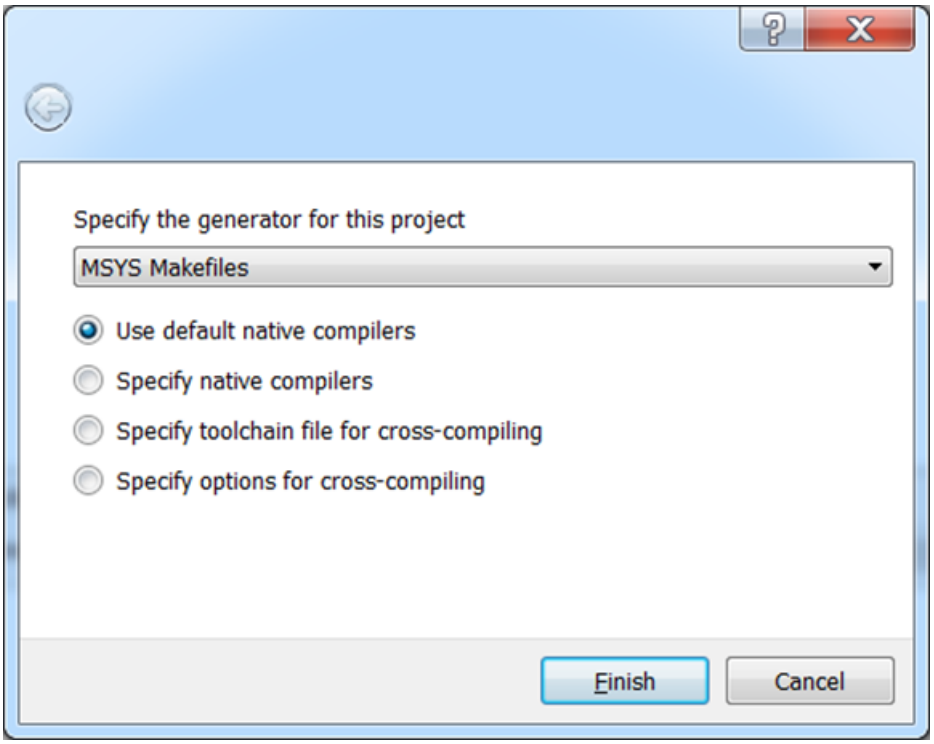

FIGURE A.2: Specifing the generator

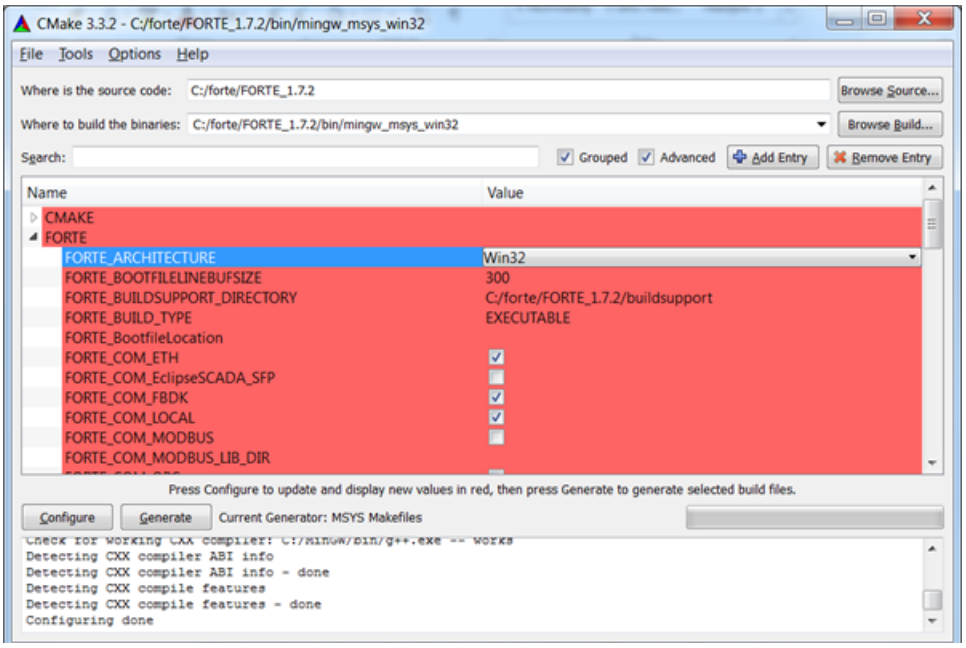

FIGURE A.3: Configuring architecture of compilled FORTE

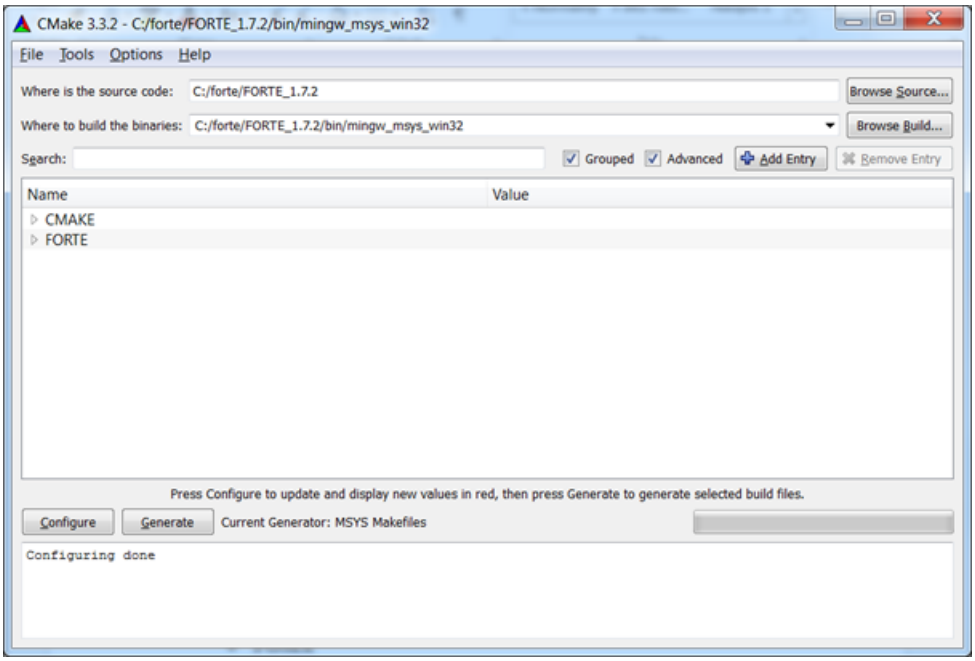

FIGURE A.4: Configuration done.

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

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 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 install 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.

Bibliography

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.

Drath, R. and A. Horch (2014). "Industrie 4.0: Hit or Hype?" In: *IEEE Industrial Electronics Magazine* 8.2, pp. 56–58. ISSN: 1932-4529. DOI: 10.1109/MIE.2014.2312079.

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.