**System Architecture   
Specification**

**(Architekturspezifikation)**

(TINF19C, SWE I Praxisprojekt 2020/2021)

Project: OPC UA Server Farm

Customer: Rentschler & Holder

Rotebühlplatz 41

70178 Stuttgart

Supplier: by Philipp Förster - Team 3  
 (Nora Baitinger, Antonia Wermerskirch, ~~Carl Beese~~, Lara Mack, Bastiane Storz)

Rotebühlplatz 41

70178 Stuttgart

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comment** |
| 0.1 | 03.11.2020 | Philipp Förster | created |

**[Contents](http://www.bredemeyer.com/papers.htm Inhalt)**

[1. Introduction 3](#_Toc39596149)

[1.1. Glossar 3](#_Toc39596150)

[2. System Overview 4](#_Toc39596151)

[2.1. System Environment 4](#_Toc39596152)

[2.2. Software Environment 4](#_Toc39596153)

[2.3. Quality Goals 4](#_Toc39596154)

[2.3.1. Usability 4](#_Toc39596155)

[2.3.2. Maintainability 4](#_Toc39596156)

[2.3.3. Portability 4](#_Toc39596157)

[3. Architectural Concept 5](#_Toc39596158)

[3.1. Architectural Model 5](#_Toc39596159)

[3.2. Component Diagram 6](#_Toc39596160)

[4. Systemdesign 7](#_Toc39596161)

[5. Subsystemspecification 8](#_Toc39596162)

[5.1. <MOD.001>: Library 8](#_Toc39596163)

[5.2. <SUBMOD.001.001>: Converter 8](#_Toc39596164)

[5.3. <SUBMOD.001.002>: AML Packager 9](#_Toc39596165)

[5.4. <MOD.002>: Command Line Tool 9](#_Toc39596166)

[5.5. <MOD.003>: Graphical User Interface 10](#_Toc39596167)

[5.6. <MOD.004>: Logging 10](#_Toc39596168)

[6. Technical Concepts 11](#_Toc39596169)

[6.1. Persistence 11](#_Toc39596170)

[6.2. User Interface 11](#_Toc39596171)

[6.3. Ergonomics 11](#_Toc39596172)

[6.4. Communication with other IT-Systems 11](#_Toc39596173)

[6.5. Deployment 11](#_Toc39596174)

[6.6. Data Validation 11](#_Toc39596175)

[6.7. Exception Handling 11](#_Toc39596176)

[6.8. Logging 11](#_Toc39596177)

[6.9. Internationalisation 12](#_Toc39596178)

[6.10. Testability 12](#_Toc39596179)

[6.11. Availability 12](#_Toc39596180)

[7. Figures 13](#_Toc39596181)

# Introduction

Our project is about creating a server farm in order to test OPC UA clients locally. It must be able to run many instances of servers which will be accessible via network. They will be

configured using AutomationML / CAEX 3.0

We will preconfigure 10 servers that cover different use cases for the clients.

## Glossar

**.NET** The .NET Framework is a software development and runtime environment developed by Microsoft for Microsoft Windows.

**AML** Automation Markup Language is an open standard data format for storing and exchanging plant planning data.

**AML DD** AML Device Description

**AMLX** AML Package

**CLI** The Command Line Interface from Microsoft Windows

**GSD** General-Station-Description

**GUI** Graphical User Interface

**CSP+** The control and communication profile CSP+ is a specification and description file that contains and provides the necessary data for CC-Link family compatible devices for commissioning, operation and maintenance.

**IODD** IO Device Description describes the sensors and other participants in an IO-Link network.

# System Overview

In order to create a new server instance, the user must provide a configuration file for it. This file will be interpreted and according to that, the server settings will be changed. After that, you should be able to connect to the server using the OPC UA Client UA-Expert. TODO IP???

## System Environment

We will create one library containing the key functionality and a second one for the UI. By now, it could either be a console application or a graphical one.

The core library will contain logic to interpret the configuration files and be able to create new instances.

It will be used in the main program with the UI, where the user can start servers.

Splitting up UI and logic allows us and other programmers to build different applications based on the server farms code.

## Software Environment

The system requires the .NET framework version 4.7 and up in order to run. That framework only works on Windows 7 or later. The library can be implemented into any kind of software that utilities the .NET framework 4.7 or later. TODO??? C99

## Quality Goals

We have the following quality goals for the OPC UA server farm:

### Usability

We provide 10 configurations for the servers which should cover most of the users use cases. That way, they don’t have to deal with the configs and can focus on testing the client. It would be best to have a graphical user interface with for example a selection box to choose the config or a button which opens the file dialog to select a custom one.

### Maintainability

Software principles like SOLID help to keep software maintainable and flexible. Future changes won’t force a lot of refactoring, so the implementation of new features doesn’t cause trouble.

We will try to focus on clean code and self-explanatory code.

### Portability

The server farm must be able to run on both Linux and Windows. There are some features that require OS methods and settings such as creating new network adapters. They might only be implemented for one OS, however there is the possibility to implement it for others.

# Architectural Concept

The system is based on the open62541 stack, which is a open source implementation of OPC UA.

## Architectural Model

The system can be divided into two main parts. The execution layer, where the actual conversion takes place and the presentation layer, which makes the execution layer accessible for the user.

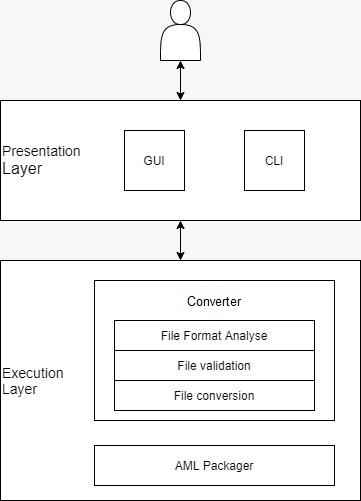


Figure 1 - Architecture Model

The presentation layer is divided into the two different ways to access the library. The library contains the conversion rules to generate the requested AML file.

The first option is the command line interface and the second the graphical user interface. The GUI will ensure the access to the library to users who are not familiar or comfortable with a CLI.

The second main section is the execution layer that can be divided into the two parts Converter and AML Packager. These two parts combined form the Library that can work as its own and could be used as a stand-alone library by developers.

## Component Diagram

Ein Bild, das Screenshot enthält.

Automatisch generierte Beschreibung

Figure 2 - Component Diagram

The main module of the whole DD2AML Converter is the Library. It is divided into two components. Together they can work as a stand-alone library to fulfil the task of converting a DD-file into the corresponding AML file and create the AMLX package.

The Converter module has some tasks to fulfil: First of all it must analyse the format of the given file and check if it is one of the listed ones (IODD, CSP+ and GSD).

It also has to validate the input file with a parser. In case the file is corrupted or contains invalid syntax, this will throw an exception and terminate the process.

The next task is the actual conversion. There will be specific conversion rules for each of the three initial formats to generate the correspondence to the newly generated AML file.

In the second part of the Library, the AML packer will create an AMLX package. It contains the logic to collect all the dependencies of the AML file and save them all to a new directory. It uses the AML Engine to get the task done.

The components CLI and GUI allow the user to access the library and to interact with the other components.

Additional, there will be a Logger module that provides a logging interface. It will be implemented in the other modules to log the errors and warnings that may occur.

# Systemdesign

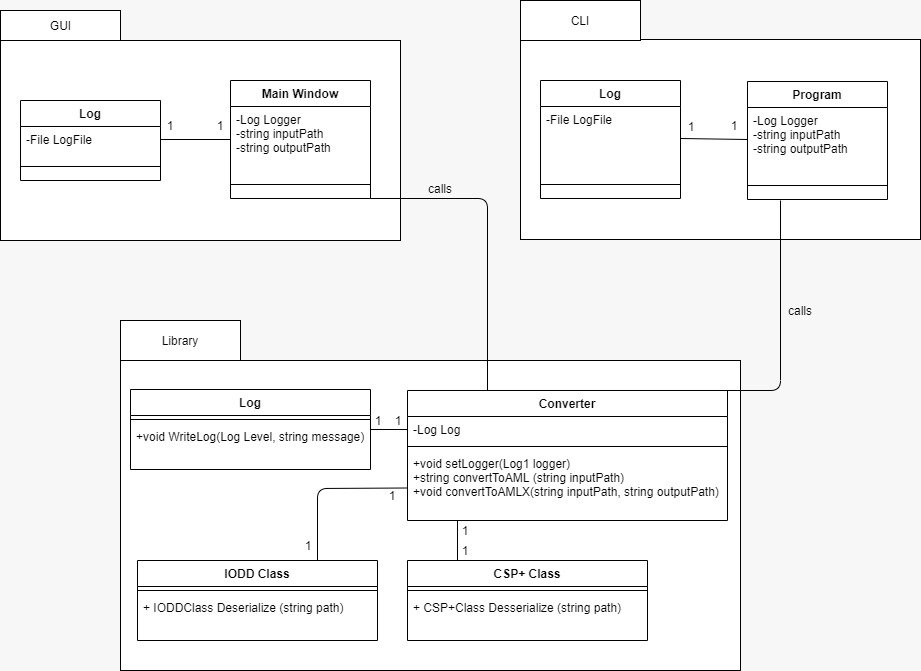


Figure 2: System Design

# Subsystemspecification

## <MOD.001>: Library

This Library is the most important module because it contains the logic for converting the input file to an AML file. It also includes the AML Packer that generates an AMLX with the AML file and all its dependencies.

|  |  |
| --- | --- |
| ***<MOD.001>*** | DD2AML.lib |
| ***System requirements covered:*** | /LF50/, /LF80/ |
| ***Service:*** | * Providing a stand-alone library |
| ***Interfaces:*** | * Function that has the input format, the path to the input file and the path for the output AMLX package * Function that has the input format, the path to the input file and returns an AML file as a string |
| ***External Data:*** | * Input file * Output file / package |
| ***Storage location:*** | [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.Lib*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.Lib) |

## <SUBMOD.001.001>: Converter

This module performs the conversion. It takes the information from one of the format-classes and builds the AML accordingly.

|  |  |
| --- | --- |
| ***<SUBMOD.001.002>*** | Converter logic |
| ***System requirements covered:*** | /LF10/, /LF20/, /LF30/, /LF100/ |
| ***Service:*** | * Handle incoming conversion requests * Analyse the format type of the file * Read and parse input file (file validation) * Convert to AML file * Return AML file |
| ***Interfaces:*** | * IODD class * CSP+ class * Output AML file structure |
| ***External Data:*** | * Input data from format-class * Output AML file structure |
| ***Storage location:*** | [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/Converter.cs*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/Converter.cs) |

## <SUBMOD.001.002>: AML Packager

The AML Packager collects all the dependencies and builds an AMLX package from the AML file and all additional referenced resources.

|  |  |
| --- | --- |
| ***<SUBMOD.001.003>:*** | AML Packager |
| ***System requirements covered:*** | /LF40/ |
| ***Service:*** | * Collect all referenced resources * Compress the collected data into AMLX package |
| ***Interfaces:*** | * File references in AML of input file |
| ***External Data:*** | * AML file * Input file * AMLX package |
| ***Storage location:*** | [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/AMLPackager.cs*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Lib/AMLPackager.cs) |

## <MOD.002>: Command Line Tool

This module is about the command line tool. It specifies and implements the exact input and output for the command line interface.

|  |  |
| --- | --- |
| ***<MOD.003>*** | Command Line Tool |
| ***System requirements covered:*** | /LF60/, /LF100/ |
| ***Service:*** | * Handle user input * Display information to the user * Handle all kinds of upcoming exceptions (Corrupted DD file, failed conversion, failed compression, …) |
| ***Interfaces:*** | * User input (Parameters, DD file path, AMLX output path…) * Command line interface (Tool, provided by Windows) |
| ***External Data:*** | * Library * Logging |
| ***Storage location:*** | [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.CLI*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.CLI) |

## <MOD.003>: Graphical User Interface

This module specifies and implements the graphical user interface and manages all possible in- and outputs.

|  |  |
| --- | --- |
| ***<MOD.004>*** | Graphical User Interface |
| ***System requirements covered:*** | /LF70/, /LF100/ |
| ***Service:*** | * Display a graphical user interface to the user * Handle user input * Handle all possible outputs, including any kind of occurring exceptions |
| ***Interfaces:*** | * User input (DD file path, AMLX output path, trigger to start conversion) * Self-defined graphical interface |
| ***External Data:*** | * Library * Logging |
| ***Storage location:*** | [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.Gui*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.Gui) |

## <MOD.004>: Logging

The Logging module provides the logic for logging the systems status and the current state of conversion. This information will help the users to understand the converting process and achieve error information if necessary.

|  |  |
| --- | --- |
| ***<MOD.005>*** | Logging |
| ***System requirements covered:*** | /LF80/ |
| ***Service:*** | * Specifying an interface for a logger that can be used within the library * Implementing a logger based on the interface for the command line tool * Implementing a logger based on the interface for the GUI tool |
| ***Interfaces:*** | * self-defined interface for logger |
| ***External Data:*** | * Log file |
| ***Storage location:*** | * [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.CLI/Logger.cs*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.CLI/Logger.cs) * [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Gui/Logger.cs*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/blob/master/SOURCE/src/Dd2Aml.Gui/Logger.cs) * [*https://github.com/WAntonia/TINF18C\_Team\_3\_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.Lib/Logging*](https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/tree/master/SOURCE/src/Dd2Aml.Lib/Logging) |

# Technical Concepts

## Persistence

Servers can be created using the CAEX config files. They must be stored on the users local drive in order to be imported by the program.

## User Interface

To create new servers, a UI is needed. There will either be a console application or a graphical user interface.

## Ergonomics

The GUI will have a simple and clean look. A standard, easy to read font will be used. It should work intuitive without the need of a usage guide. Since there are not many elements needed, this should be relatively easy to achieve.

## Communication with other IT-Systems

The server farm is split into 2 parts, where the logic is completely separated from the

presenter. Therefore, anyone can use the core library and integrate it into their project or build a custom UI for it.

## Deployment

The CLI tool and GUI tool will be deployed with an installation package (\*.msi).TODO???

## Data Validation

The user provides the configuration files for the servers. They need to be checked, and only if they follow correct syntax and contain all required fields, the server can be configured accordingly.

## Exception Handling

If the configuration file is not valid, the user will be informed. In the same way any other error will be presented to the user. Especially the OS operations like creating new network adapters might fail due to lacking permissions.

## Logging

Important information as well as errors will be logged into a separate file.

## Internationalisation

Everything (code and UI) will be available in English. We will try to avoid complex wording; therefore, most people won’t have to deal with a language barrier.

## Testability

The code will be covered by unit tests for each module.

To test the whole system working together, we will design a test plan for testmanger. It will be a manual test.

## Availability

The program and source code are available on GitHub and therefore accessible for everyone.

# Figures

[Figure 1 - Architecture Model 5](#_Toc39598041)

[Figure 2 - Component Diagram 6](#_Toc39598042)