System Architecture Specification

(Architekturspezifikation)

(TINF18C, SWE I Praxisprojekt 2019/2020)

Project:	*DD2AML Converter*
Customer:	*Rentschler & Ewertz* Rotebühlplatz 41 70178 Stuttgart
Supplier:	Team 3 - by Carl Beese (Nora Baitinger, Antonia Wermerskirch, Carl Beese, Lara Mack, Bastiane Storz) Rotebühlplatz 41 70178 Stuttgart

Version	Date	Author	Comment
0.1	07.09.2018		created
0.2	26.10.2019	Bastiane Storz	Headings and table of contents revised and added technical concepts
0.3	4.11.2019	Carl Beese	Added System Overview, System Architecture, System Design
0.4	5.11.2019	Bastiane Storz	Design revised

Contents

1. Introduction

The goal of this project is to develop a software which supports the conversion from a IODD, ESI or CSP+ file to an AML file or respectively an AMLX package. The main part of this software should be a library that can perform such a conversion. There also should be a command line tool and a tool with a graphical user interface, which use this library to convert GSD files. The whole software is dedicated to the .NET-Platform 4.7 or later.

1.1. Glossar

AML	Automation Markup Language is an open standard data format for storing and exchanging plant planning data.
.NET	The .NET Framework is a software development and runtime environment developed by Microsoft for Microsoft Windows.
CLI	The Command Line Interface from Microsoft Windows.
GUI	Graphical User Interface

2. System Overview

The system will work as follows. The user specifies a file, the system checks the format of that file an validates the syntax. If the syntax is valid the system can either save a new AML-X package in the same directory of return the AML file as a string.

2.1. System Environment

There will be two ways to use the converter. Both require that the supplied file is valid. The first is the library which can be implemented into other projects. This will save an AML-X package to a specified location. The second is the CLI or the GUI. They will give the user the option to either create an AML-X package or just an AML file.

2.2. 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 utilises the .NET framework 4.7 or later.

3. Architectural Concept

The system will be based on previous efforts by a team of students who created something very similar, namely a converter for GSDML files. It can be divided into three modules.

The first part is the library where the conversion will take place.
It will have a module to validate the input file with a parser. In case the file is corrupted or contains invalid syntax, this module will throw an exception an terminate the process.
The next module is responsible for the actual conversion. There will be specific conversion-rules for each of the three initial formats. There will be a function to create an AML-X package and one to create just the AML file.
Another module will contain the logic to collect all the dependencies of the AML file and save them all to a new directory. This directory will them have compressed into an AML-X package.
There will also be logging with a logger interface that will be implemented in the other modules to log the errors and warnings that may occur.

The second part is the CLI. It will be one way to connect the user to the system.

The third part is the GUI. It will ensure using the converter is accessible to the user who are not familiar or comfortable with a CLI.

3.1. Quality Goals

The following quality goals should be achieved by this architecture.

3.1.1. Usability

By offering different tools for different operating scenarios, a high degree of user-friendliness is to be achieved for every type of user.

3.1.2. Maintainability

To divide the software into these smaller modules should help to make the software easier to analyze, maintain and modify.

3.1.3. Portability

The software and the library should be portable. This means that the functionality to convert a DD file to AML/AMLX should be easy integrable by other software products.

3.2. **Architectural Model**

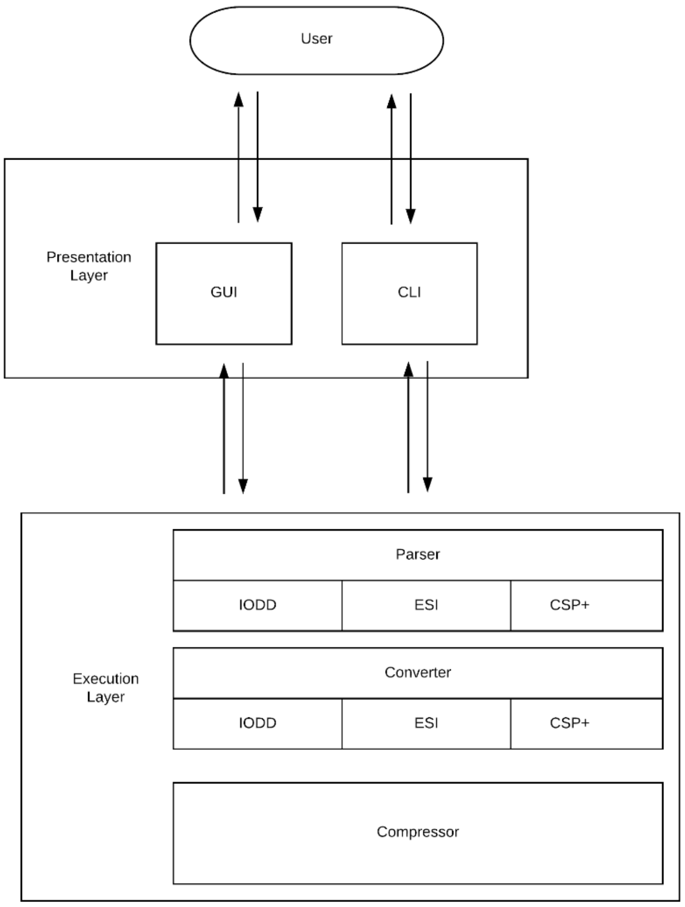

Figure 1: Architecture Model

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

4. Systemdesign

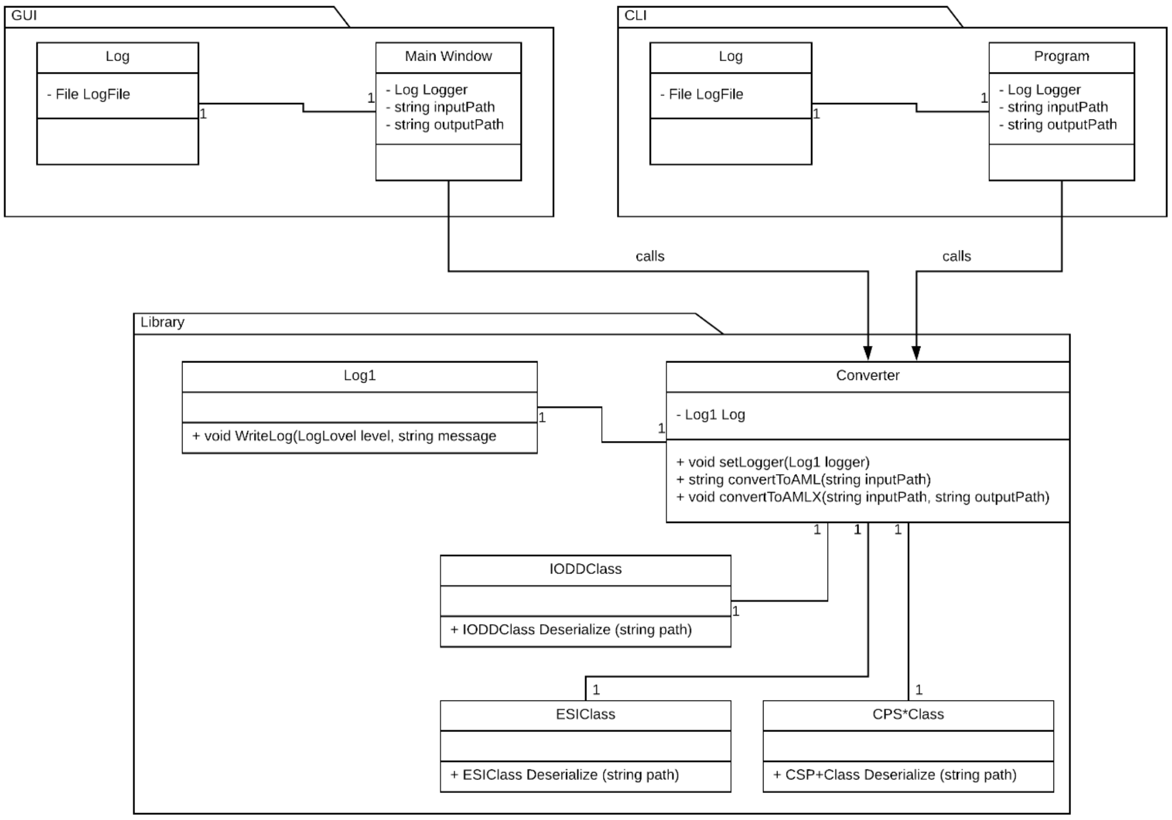
GUI
Log
- File LogFile
Main Window
- Log Logger
- string inputPath
- string outputPath
1
1
CLI
Log
- File LogFile
Program
- Log Logger
- string inputPath
- string outputPath
1
1
calls
calls
Library
Log1
+ void WriteLog(LogLovel level, string message
1
1
Converter
- Log1 Log
+ void setLogger(Log1 logger)
+ string convertToAML(string inputPath)
+ void convertToAMLX(string inputPath, string outputPath)
1
1
1
IODDClass
+ IODDClass Deserialize (string path)
1
1
ESIClass
+ ESIClass Deserialize (string path)
1
CPS*Class
+ CSP+Class Deserialize (string path)

Figure 2: System Design

5. Subsystemspecification

5.1. <MOD.001>: DD2AML.lib

This Module contains the logic for converting the input file to a AML file.

<MOD.001>	*DD2AML.lib*
System requirements covered:	*/LF10/, /LF20/, /LF30/, /LF40/*
Service:	• *Read and parse input file* • *Convert to AML file / AMLX package* • *Return the file/ package*
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 a AML file as a string*
External Data:	• *Input file* • *Output file / package*
Storage location:	*Not yet available*
Open Points:	-

5.2. <SUBMOD.001.002>: Converter logic

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:	*/LF30/*
Service:	• *Convert to AML file* • *Return AML file*
Interfaces:	• *IODDClass* • *ESIClass* • *CSP+Class* • *Output AML file structure*
External Data:	• *Input data from format-class* • *Output AML file structure*
Storage location:	*Not yet available*
Open Points:	-

5.3. <MOD.002>: Compressor

This Module collects al the dependencies and builds an AMLX package from the AML file and all additional referenced resources.

<MOD.002>	*Compressor*
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:	*Not yet available*
Open Points:	-

5.4. <MOD.003>: 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:	*/LF80/*
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:	• Converter library
Storage location:	*Not yet available*
Open Points:	-

5.5. <MOD.004>: GUI Tool

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

<MOD.004>	GUI Tool
System requirements covered:	*/LF90/*
Service:	• Handle user input • Handle all possible outputs, including any kind of occurring exceptions • Display a graphical user interface to the user
Interfaces:	• User input (DD file path, AMLX output path, trigger to start conversion) • Self-defined graphical interface
External Data:	• Converter library
Storage location:	*Not yet available*
Open Points:	-

5.6. <MOD.005>: Logging

<MOD.005>	Logging
System requirements covered:	-
Service:	• Specifying an interface for a logger that can be used within the Converter 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:	• *CLI* • *GUI*
External Data:	• *Log file*
Storage location:	*Not yet available*
Open Points:	-

6. Technical Concepts

6.1. Persistence

Not applicable, because data persistence is not relevant for this kind of project.

6.2. User Interface

The user interface will be specified in the module specification for MOD.004.

6.3. Ergonomy

Not applicable.

6.4. Transaction Control

Not applicable.

6.5. Session Control

Not applicable.

6.6. Communication with other IT-Systems

Not applicable.

6.7. Deployment

The CLI tool and GUI tool will be deployed with an installation package (*.msi). The library will be separately available as a NuGet package.

6.8. Data Validation

Before a conversion can take place the input file needs to be validated to ensure a conversion is possible.

6.9. Exception Handling

Not applicable.

6.10. Logging

The logging will be specified by the logging module MOD.005.

6.11. Configurability

Not applicable.

6.12. Parallelisation

Not applicable.

6.13. Internationalisation

Not applicable.

6.14. Migration

Not applicable.

6.15. Testability

The software will be tested with System Unit Tests.

6.16. Scalability

Not applicable.

6.17. Availability

Not applicable.

7. Figures