DHBW
Duale Hochschule
Baden-Württemberg
Stuttgart

System Architecture

Specification

(TINF21C, SWE)

Project: Modelling Wizard Improvements

Customer: Markus Rentschler

Christian Holder

Team:

Project Manager	– Robin Ziegler (inf21100@lehre.dhbw-stuttgart.de)
Developer	– Nils Hoffmann (inf21194@lehre.dhbw-stuttgart.de)
Test Manager	– Michael Grote (inf21111@lehre.dhbw-stuttgart.de)
System Architect	– Fabian Kreuzer (inf21106@lehre.dhbw-stuttgart.de)
Tech. Documentation	– Dana Frey (inf21099@lehre.dhbw-stuttgart.de)
Product Manager	– Maximilian Trumpp (inf21123@lehre.dhbw-stuttgart.de)
Graphical Designer	– Sophie Kirschner (inf21083@lehre.dhbw-stuttgart.de)

Change History

Version	Date	Author	Comment
0.1	13.10.2022	Fabian Kreuzer	created
0.2	16.10.2022	Nils Hoffmann	Added points 1-3
0.3	18.10.2022	Nils Hoffmann	Rework of some chapters
1.0	20.10.2022	Fabian Kreuzer	Visual updates
1.1	13.04.2023	Fabian Kreuzer	New Version Start
1.2	14.04.2023	Fabian Kreuzer	Removed previos points 1-3/Redundance
1.3	20.04.2023	Fabian Kreuzer	System Design
1.4	22.04.2023	Fabian Kreuzer	System Design
2.0	25.04.2023	Fabian Kreuzer	Finished

Introduction

This project targets the improvement and debugging of an existing stand-alone application which was based on a plugin for the AutomationML editor. This includes modifications on the existing "Easy/Advanced mode" functionality and the "Import" option. Also, improvements on the general usability and user-friendliness are intended in the events of this project. Furthermore, there are multiple cases where it is planned to fix bugs and errors

Glossar

_ AML _ *Automation mark-up Language is an open standard data format for storing and exchanging plant planning data.*

AMLX AML Package to store also not AML files in one package.

CAX File format of AML Device files.

C# High level language often used for programming

_ GUI _*Graphical User Interface*

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

AML DD AML Device Description

GSD General Station Description, data format for Profibus and Profinet devices.

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

GUID Globally Unique Identifier a 16 Byte nummer used as an unique Identificator

System Overview

The system is currently a stand-alone application for Windows 10 or higher. It might be usable on prior Windows versions but it is neither tested nor intended as the support for these versions has already run out.

System Environment

The stand-alone application can only be accessed via Windows, as this operating system is used as the platform. The application can be installed and the graphical user interface can be used via this platform.

Software Environment

For the stand-alone application to work, you need at least version 4.5 of the .Net framework, as C# was used developing it. This version of the framework is available from the "Windows Vista" operating system variant onwards. The application can be started either by first installing, via the current published installer and then starting the finished .exe application or running the source code in Microsoft Visual Studio.

Modifiability

Since the program is open source and publicly viewable, it can be extended by anyone. This is important if any use cases or issues arise in the future. Therefore we tried our best to keep our naming consistent and not to overcomplicate our code. Also we commented more complex code-blocks to further increase the modifiability.

Code Quality

Code quality is one of the most important aspects when it comes to software that is being developed further and may come from different developers. For this reason, we have made it our mission to address and improve the problem of code quality. To maintain a certain standard, we have agreed on certain conventions:

- Commenting on sections of code that are not clearly understandable in order to explain the implemented idea to others
- Programming paradigms and programming principles that make the code easy to understand, such as one main function per file and a sensible folder structure.
- In general the source code should be compliant with the common "Clean Code" guidelines

These lead to the point that the code is intented uniformly to improve readability and comprehension. Furthermore, the program is divided into modules that can then be imported and used. Otherwise, you run the risk of having to write duplicate code. In addition, it is our responsibility to write documentation that records which ideas have been implemented and how, so that the existing functions are easier to understand for future developments and can be built upon.

Architectural Concept

We used WinUI3 to implement a new and more Modern Style of the projekt. WinUI3 needs a different structure than the old Project so we decided to totally overhall the System Architecture. While doing this, we also decided to rewrite most of the code and make a more object oriented approatch than it was seen in the older project. Because of these reasons we needed to totally overhall the code but, we will try to implement all the previously implemented functions. Since in the previous project these functions are mostly working poorly or not working at all, we will give our best to improve them. Based on the time limitations we certainly will not be able provide a finished product with all the necessary functions to make it market-ready. We will focus on having everything work what is needed to show of our new usability concept and improved GUI. Thats why while designing the architecture we concentrated on an understandable and logic system, so that other teams after us have an easier time finishing these functionalities.

Architectural Model

The application is now designed based on the views and pages given by WinUI3. We tried our best to seperate the functions of the UI and processes so that its easier to find what is going on. Therefore we decided to seperate it in our folder structure. We now have views, with all the WinUI3 pages, processes with all our static functions, the user can trigger via elements in the GUI and finally we have objects, with all the different object models we needed. (Build on file structure to be implemented here) a particulary important object is the instances object, which is an internal object. It saves the currently loaded file. One could say it is the drive of our program. With this said you can tell we implemented a MVC patern with the objects as our model, the processes as our Controller and the UI as our View.

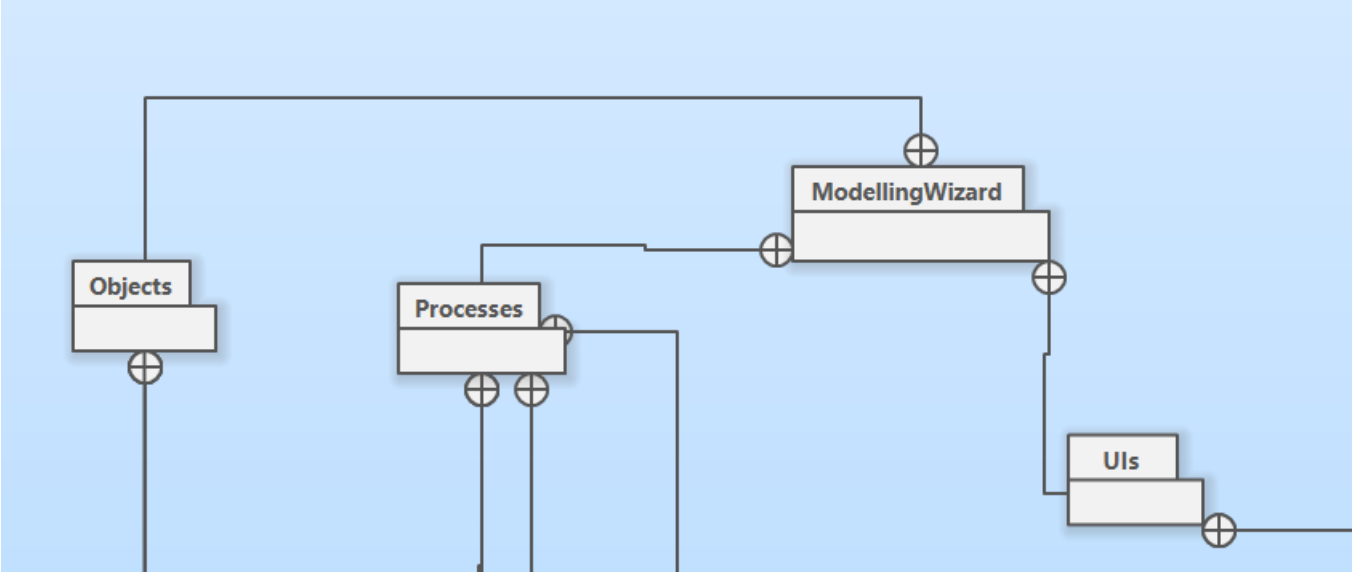

Figure 1 – Architecture

Almost all the logic is contained in the processes, which thus forms the centre of the entire system architecture and connects our objects with the GUI seen and accessed by the user.

System design

The program is designed as a stand-alone application with input from the Modelling Wizard. The "Program" class receives information from the Modelling Wizard's plugin and sends it to the GUI which displays the information for the user to interact with.

Views Mod.001

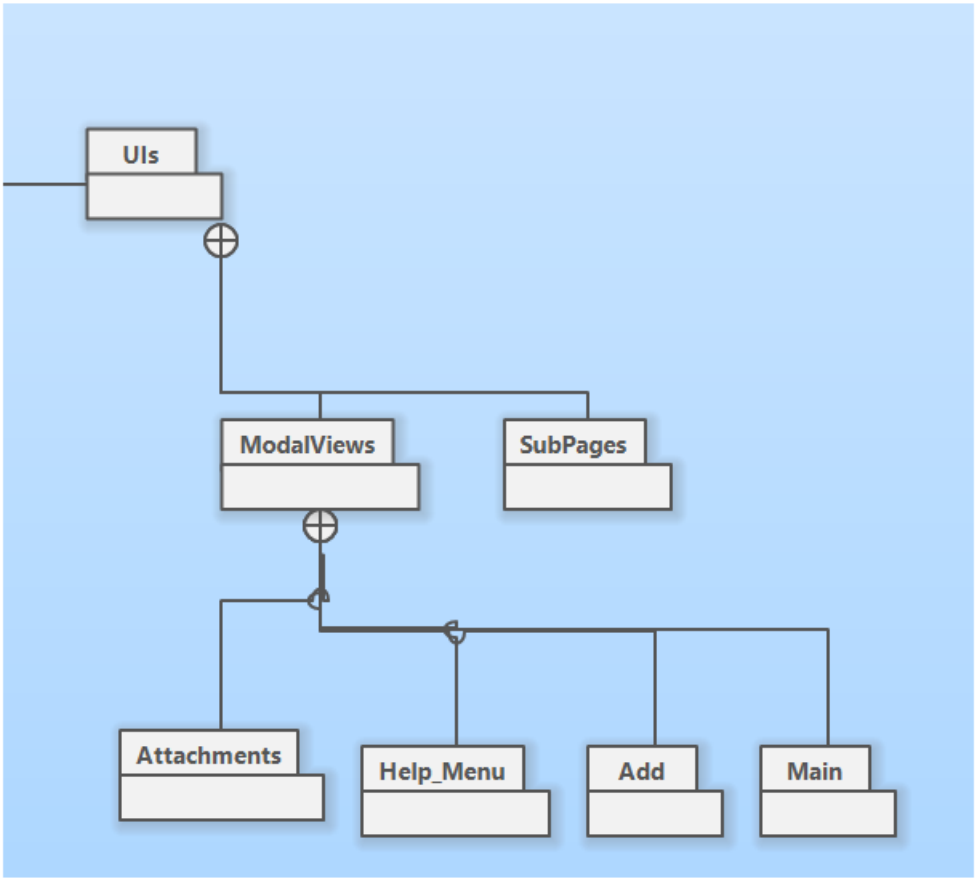

_Figure 2 – Views

The views are our whole GUI and is based on the newest WPF format WinUI3. Therefore the views are based on different XML pages. In this system we are loading the App.xml page first, this page is only a dummy page to swap to the MainWindow.xml. With this approatch we could easily load a different MainWindow without the need to also load a whole page with all its objects.

Main Window

In the MainWindow.xml we display almost no data. Its main use is to hold the control elements and some basic overview. With the control elements you can either for example open a file, create a new file or navigate to a file of our different sub pages.

Sub Pages

These pages are the place where the currently loaded data from instances is displayed and can be edited by the user.

Processes Mod.002

All the processes are different static C# classes. These classes hold different methods and would be called libraries in a more procedural approatch.

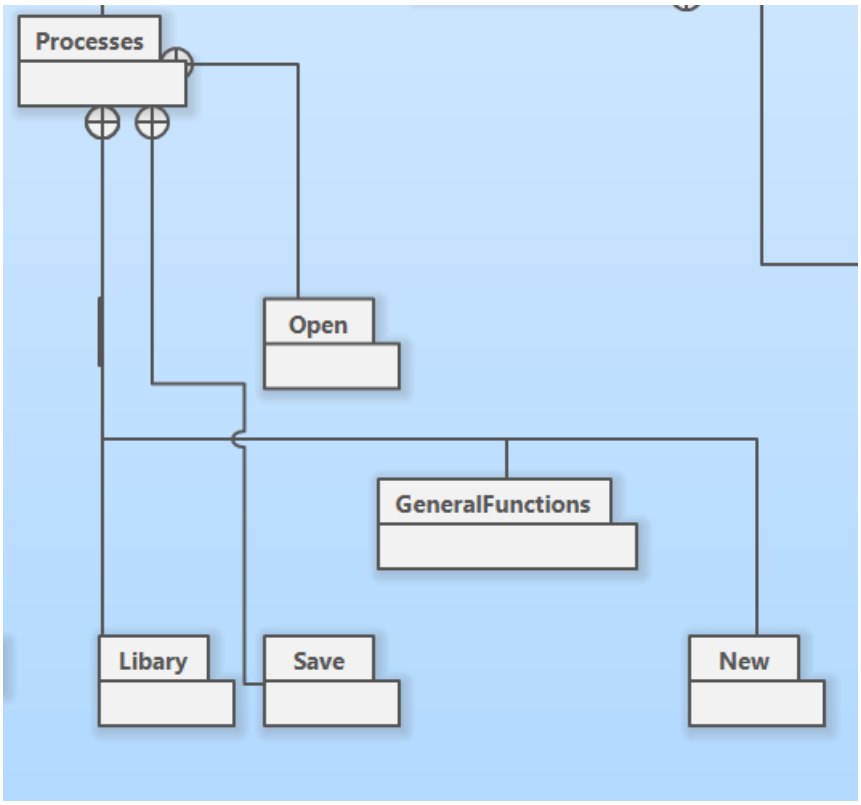

_Figure 3 – Processes

General Functions

General functions contain all our classes with functions which serve no higher purpose, like SeartchForGUID.cs. This file holds the search for GUI method, its parameter is a string the GUID and the return value is a library with this GUID or NULL if no library is found. It's a recursive search for a GUID of a library its main use is to find the library which is currently selected, to view its attributes.

Library

The library folder currently only holds everything to load the libraries. For a good overview we decided to make two different load files. The main Load.cs has different functions. These are mainly used to load all the necessary data from a byte array. It returns all the RoleClasses, Interfaces and SystemUnitClasses of the byte array. To achieve this we first of all use the AML.Engin to try and load a CAEXDocument object from the array and then recursifly load all sub libraries and all attributes with sub attributes. The LoadStandardLibrary.cs only holds a method to load all the preloaded files which hold the basic classes we use to add new libraries to our file.

New

In this folder all the classes we use to create a new document are stored. As the only required library is the SystemUnitClass we only have the CreateSysClass.cs class. The only used function is a void function called execute, which takes no parameters. It generates a new empty SystemUnitClass in our instances.

Open

The Open.cs class has the open method, which takes in and returns the same objects as the load file. It is only used to load .xaml files, it uses the load method of Load.cs to open .aml files. The loaded classes are stored in the different instances.

Save

This function should use our instances objects to save them either into the old CAEXDocument object or create a new one. Due to time limitations this function is currently not implemented.

Objects Mod.003

The Objects are our non static classes which hold different data models. We also save the current instances in this folder.

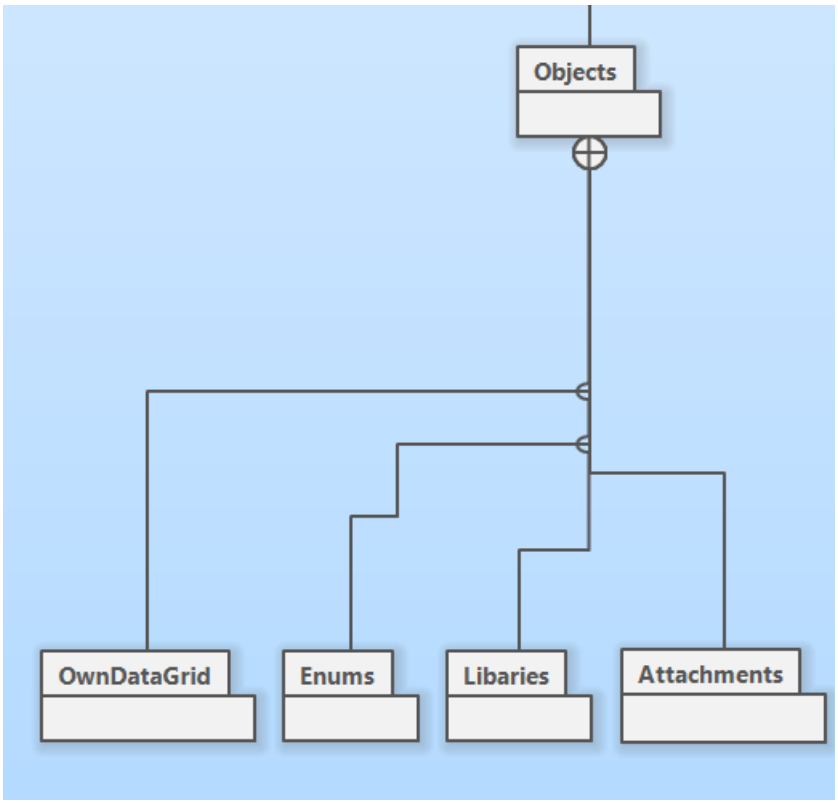

_Figure 4 – Objects

Attachments

An attachment object is our datastructor for an attachment. It only has the properties title and base64content. The title is just the attachment file name and the base64Content is the file content in base64.

Libraries

The Libraries are our base datatype for all the different classes we load out of the .aml/.amlx files. It has a name as string, GUID as string, SubLibrarys as a List of Librarys and a List of attributes. It also has methods to remove and find libraries using its name.

OwnDataGrid

OwnDataGrid is a child of the windows data grid. We use it to build different grids, where different values are bound to the grid depending on if the user is in expert mode or in easy mode.

DataforCreatePins

DataforCreatePins is used in the converter class used to convert .edz files to .xaml files.

Instances

Instances store everything what is important to save the current state of the program, like darkmode as bool, expertmode as bool, config data which holds all the preloaded libraries and loaded -Roleclass, -SystemUnitClass and -LoadedInterface each as a list of library objects.

MyTreeNode

MyTreeNode is used as the holder of the config data, which the user sees when clicking on the 'Add class'-buttons.

Subsystem specification

All modules have no external data.

<MOD.001>: Views (GUI)

<MOD.001>	Graphical User Interface
_ System requirements covered: _	LF40: GUI, LF50: Display device in a readable way, LF60: Edit device

_ Service: <MOD.001>	*The graphical user interface is taking input from the user and sending it to the controller by calling event functions.* Graphical User Interface
_ Interfaces: _	-__--
_ Modul documentation: _	https://github.com/robinziegler/TINF21C_Team4_Modelling_Wizard_Improvements/wiki/MOD.001--Views-(GUI)

<MOD.002>: Processes

_ <MOD.002> _	Processes
_ System requirements covered: _	LF10: Import, LF20: File validation, LF30: Error handling, LF70: Create device, LF80: Export device
_ Service: _	*Logic distribution is handled by the controller. It is reacting to the events triggered by the GUI and takes care of creating the respective objects. Also the input and output functions are implemented in the controller.*
_ Interfaces: _	-__--
_ Modul documentation: _	https://github.com/robinziegler/TINF21C_Team4_Modelling_Wizard_Improvements/wiki/MOD.002--Processes

<MOD.003>: Objects

<MOD.002>	Objects
_ System requirements covered: _	LF40: Display a GUI and accept user input, LF50: Display the attributes of selected device, LF60: Edit the attributes of a device
_ Service: _	*Logic distribution is handled by the controller. It is reacting to the events triggered by the GUI and takes care of creating the respective objects. Also the input and output functions are implemented in the controller.*
_ Interfaces: _	-__--
_ Modul documentation: _	https://github.com/robinziegler/TINF21C_Team4_Modelling_Wizard_Improvements/wiki/MOD.003--Objects

Technical Concepts

Persistence

The usage of the publicly available "AMLX" package standard allows the system to be persistent. It also makes the created or edited models available via the AutomationML Editor. Having a persistent application is best used for an industrial environment.

Deployment

To develop the application, one has to install a valid version of Visual Studio, the most recent version is "Visual Studio 2022". As for it now to use the application it can be installed via an installer as shown in our installation guide.

Data validation

All data checks are running in the background, invisible for the user. The controller is checking for missing information and incorrect entries, that must be specified as mandatory information for example before a conversion can take place, the input file needs to be validated to ensure a conversion is possible. The data validation currently checks for the wrong entries while loading a file. It should be able to at least load a file without every mandatory entry, but not save it.

Exception handling

Exception handling is necessary to prevent errors caused by the user while using the program. So called "try-catch" blocks are used to 'catch' these and prevent unwanted or incorrect behavior of the software. This way, the user is informed about what did not work and he/she may be able to fix the problem. For user compatibility if point 7.4 throws an exeption while saving the user should be shown the exact location where the error is in the GUI.

Figures