Content

1. Introduction

The goal of this project is to develop a web-based application which allows the user to easily configure cables, add device interfaces, ports, and data attachments using a user-friendly GUI. AutomationML packages corresponding to the rules for AML component models are to be created as file output.

1.1. Glossary

.NET The .NET Framework is a software development and runtime environment developed by Microsoft for the C# programming language.

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

Angular An open-source web application framework based on TypeScript.

ASP.NET ASP.NET is an open-source, server-side web-application framework built upon the .NET framework.

API Application Programming Interface.

HTML HyperText Markup Language is the standard language for documents meant to be displayed in browsers.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation.

Node.js Node.js is an open-source JavaScript runtime environment.

REST Representational State Transfer.

2. System Overview

The system will work by providing users a GUI within which they can configure the properties of cable. For that, the user will be presented with different connector types and various other attributes to select. Upon request, the application will parse these desired properties to an AutomationML file.

2.1. System Environment

The modelling wizard for cable-models is accessible through the Angular front-end within the website of the cable online shop. Users are able to export their configurations to AutomationML-compliant files. This export alongside other logic, is handled by an ASP.NET backend.

2.2. Software Environment

The frontend of this application relies on the framework Angular, which uses the back-end JavaScript runtime environment Node.js. When the application is running, the user can access the GUI with any browser supporting the HTML 5 standard.

The backend of the application used the framework ASP.NET, which is designed for the Windows operating systems. Other operations may not support this framework.

The frontend and backend communicate via a [[REST API]]. Therefore, both the frontend and backend must be configured to use available ports.

3. Quality Concept

3.1. Functionality

The export of the configuration to an AutomationML file should allow for better storage and exchange of these configurations between systems and stakeholders. Because AML is an open standard, this export would enable a better distribution of cable-configurations within the industry.

3.2. Maintainability

Because every system needs maintenance, a major focus of this project lies within dividing the system into maintainable modules. Dividing the software into smaller modules should help to make the software easier to analyze, maintain and modify.

3.3. Usability Concept

The usability of the application is the main factor that determines whether users will consider it worth using. In order to assure usability, the following criteria are to be considered and maintained:

- **Intuitiveness**: The user should not require training or much experience with the application in order to understand and use it. Thus, the layout and features should be self-explanatory.
- **Efficiency**: The application should enable users to achieve high productivity when used. Therefore, the application should require low amount of steps to be taken in order for the user to reach a desired result. Additionally, it should provide powerful features that make skipping trivial work possible.
- **Reliability**: The application should also have a low error rate, so users can trust it to produce the correct results consistently. Additionally, it should aid users in avoiding mistakes enable them to fix occurring mistakes quickly.
- **Satisfaction**: The application should feel pleasant to use and get users to subjectively determine that they like it.

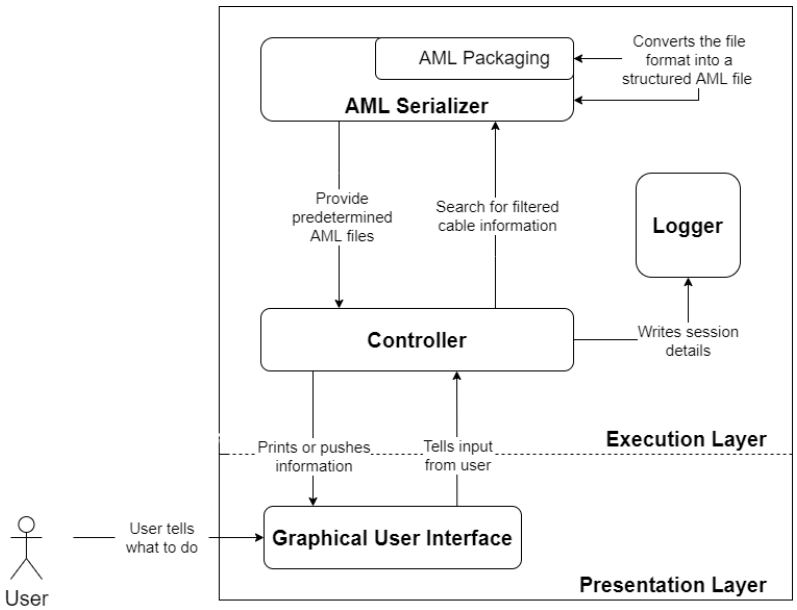
It is critical that these characteristics are incorporated within the application, however, they must be balanced and compromised between them have to be made, while the functionality of the application has to be maintained.

In order for the GUI to maintain these characteristics, the following guidelines are taken into consideration.

- **Appearance**: The GUI should feature a pleasant color palette, featuring a soft primary and matching secondary color, as well as a vibrant accent color for highlighting. Additional colors are generally to be avoided, but may be used to signal special meaning, such as green for success and red for warnings.
- **Consistency**: The GUI should use the same color palette across all of its elements and display elements objects in similar ways. Likewise, elements that look similar should behave similarly.
- **Simplicity**: The GUI should always contain a limited about of elements at one time to allow the user to quickly recognize the displayed elements and their functionality. Different parts of the GUI should be accessible with few and non-convoluted steps.
- **Expressiveness**: Texts should generally be kept short and concise in order to convey meaning quickly. When possible, expressive icons should be used to convey meaning visually. Elements representing physical objects should contain images displaying that object. Special elements, such as input fields, should be marked in a special manner to easily distinguish them.

4. Architectural Concept

The application is split into two layers, the presentation layer and the execution layer.



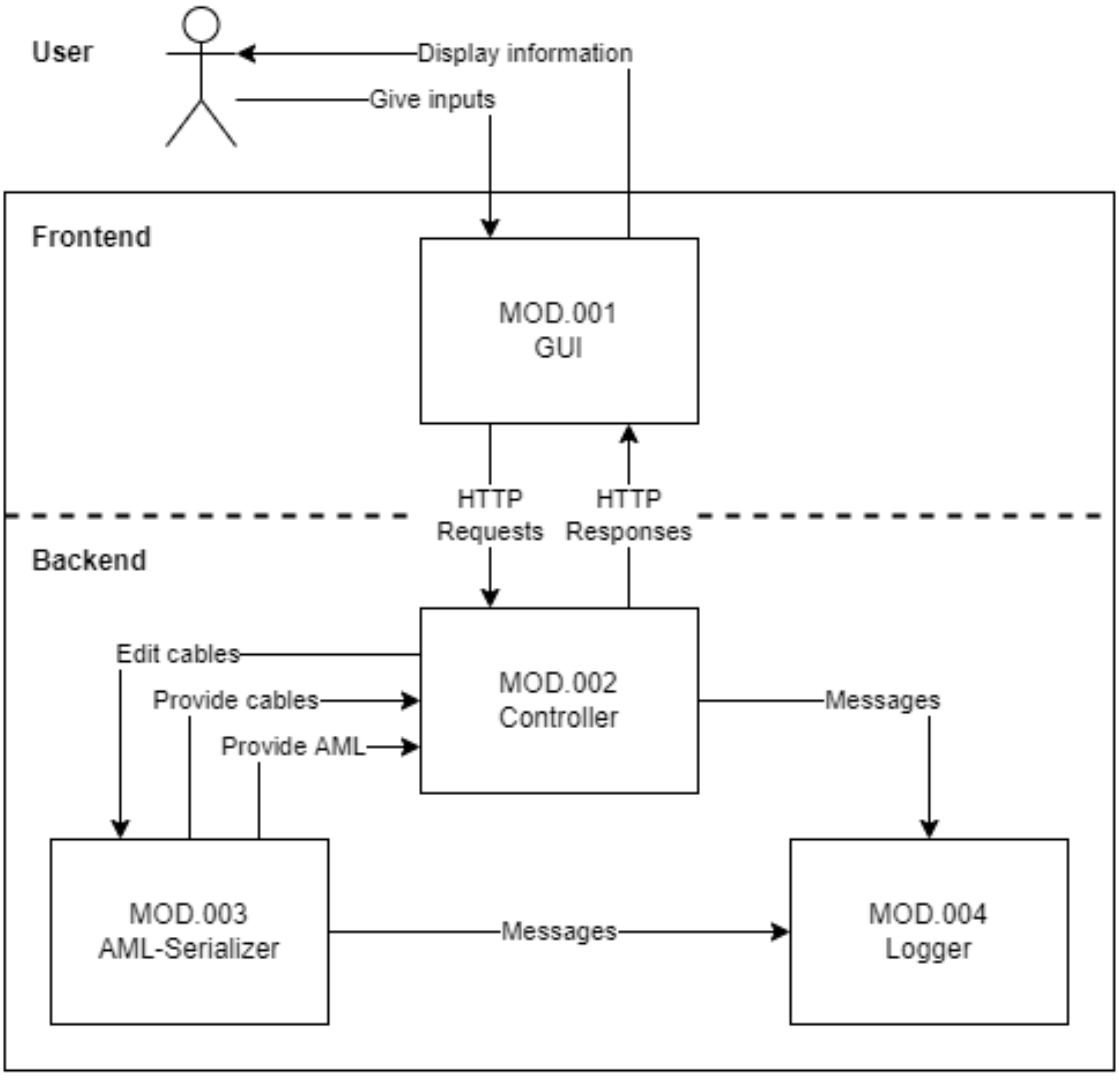
Architecture diagram

The presentation layer consists of the graphical user interface, which is tasked with handling user interaction. It displays information to the user using data received from the execution layer, and sends requested changes to it.

The execution layer contains most of the complex business logic. It receives requests from the presentation layer and responds to these with the requested data, or makes changes to the existing data based on these requests. It manages the files used the store the data and creates log files to write down information about important events and potential errors that may occur.

5. System Design

The application is designed in a frontend-backend architecture. The frontend represents the presentation layer, while the backend represents the execution layer. They communicate via the [[REST API]], with the frontend sending HTTP requests and the backend returning matching HTTP responses.



System Design

The frontend consists of one module, the GUI (MOD.001). The GUI is an Angular website, which can be accessed from the user's web browser. It is tasked with displaying information to the user and taking their inputs. This works by sending HTTP requests to the backend, which responds with the data to display. The GUI also performs minor logic like filtering and ordering search results.

The backend takes care of the complex business logic. It consists of three modules, the controller (MOD.002), the AML serializer (MOD.003) and the logger (MOD.004). The controller handles the incoming HTTP requests sent from the GUI. It validates and interprets them, and then controls the AML serializer to perform the desired tasks. The AML serializer reads and writes AML data to AML files and AMLX containers. The logger is used by both the AML serializer and controller to log important events and potential errors.

6. Subsystem Specification

6.1. MOD.001 Graphical User Interface

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

<MOD.001>	Graphical User Interface
System requirements covered	/LF10/, /LF20/, /LF30/, /LF50/, /LF60/
Services	- Handle the user input - View existing cables - Create, modify and delete cables- Provide images and documents for specific cables - Communicate changes to backend
Interfaces	Frontend of the [[REST API]]
External Data	—
Storage Location	Frontend/src/app/
Module documentation	[[MOD.001 Graphical User Interface]]

6.2. MOD.002 Controller

The controller manages incoming request and the conversation between the user and the database.

<MOD.002>	Controller
System requirements covered	/LF10/, /LF30/, /LF40/, /LF60/, /LF70/, /LF80/
Services	- Respond to the GUI requests - Load requested files and other data - Get AML data from AML serializer
Interfaces	- Backend of the [[REST API]]- Send information to logger- Control operations in AML serializer
External Data	—
Storage Location	Backend/Controllers/
Module documentation	[[MOD.002 Controller]]

6.3. MOD.003 AML Serializer

The serializer converts the information about a particular cable in the database into a AutomationML-compliant format and transfer that data to the controller.

<MOD.003>	AML Serializer
System requirements covered	/LF10/, /LF40/, /LF60/, /LF70/, /LF80/
Services	- Convert cable information to file - Provide structured AML packages
Interfaces	- Getting and setting AML data for Controller - Logging events and errors
External Data	- AML files and AMLX containers
Storage Location	Backend/AmlSerializer.cs
Module documentation	[[MOD.003 AML Serializer]]

6.4. MOD.004 Logger

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.004>	Logger
System requirements covered	/NF40/
Services	- Write timestamped information about events to the console and logfiles.
Interfaces	- Info, Warn, Error and Fatal methods to communicate messaged to be logged.
External Data	- Log files
Storage Location	Backend/Logger.cs
Module documentation	[[MOD.004 Logger]]

7. Technical Concepts

7.1. Data model

An approximate overview of the data model can be found in the SRS. Considering the aspects that are necessary a JSON model has been created that is usable to communicate between frontend and backend. The model looks as follows:


```
{
  "id": "a9c79bdb-81d4-4098-9eb8-bebb796d2842",
  "name": "BCC M313-M313-30-300-EX43T2-050",
  "library": "ProductLibrary_BALLUFF",
  "attributes": {
    "manufacturer": "Balluff GmbH",
    "manufacturerUri": "www.balluff.com",
    "deviceClass": "Double-Ended Cordsets",
    "model": "BCC M313-M313-30-300-EX43T2-050",
    "productCode": "BCC051J",
    "ipCode": "IP67",
    "material": "Zinc",
    "length": 224,
    "width": 68,
    "height": 38,
    "weight": 695,
    "temperatureMin": -5,
    "temperatureMax": 70
  },
  "wires": [
    "C1",
    "C2",
```

```
    "C3"
  ],
  "connectors": [
    {
      "type": "M12A3PinFemale",
      "path": "ConnectorLib_IEC61076/IEC61076-2/IEC61076-2-104/M8A/M8A3Pin",
      "pins": [
        {
          "name": "1",
          "connectedWire": "C1"
        },
        {
          "name": "2",
          "connectedWire": "C2"
        },
        {
          "name": "3",
          "connectedWire": "C3"
        }
      ]
    },
    {
      "type": "M12A3PinMale",
      "path": "ConnectorLib_IEC61076/IEC61076-2/IEC61076-2-104/M8A/M8A3Pin",
      "pins": [
        {
          "name": "1",
          "connectedWire": "C1"
        },
        {
          "name": "2",
          "connectedWire": "C2"
        },
        {
          "name": "3",
          "connectedWire": "C3"
        }
      ]
    }
  ]
}
```

7.2. Persistence

The application does not require storing data between sessions. The data required for possible connectors and cable attributes is read but never modified by the application. The files containing this data can be manually edited to change the product range.

7.3. User Interface

The graphical user interface (GUI) is the interface between user and program logic. The user can use this GUI to select from connector types and cable properties and then download the product data in the form of an AML file.

7.4. Ergonomics

It is important for an ergonomic GUI to be intuitive, and the user experience as appealing as possible. This includes making sure that the GUI is visually appealing and that it is designed in such a way that the user intuitively understands how to use it.

7.5. Communication with other IT-Systems

The software currently does not require communication with external IT-Systems. Developers can integrate the application into other websites.

7.6. Deployment

To deploy the frontend, the content of production build directory must be moved to a web server. Because these files are static, they can be hosted on any web server capable of serving files, such as Node.js or .NET.

To deploy the backend, a production build must be generated and the executable simply has to be launched.

7.7. Data Validation

Data validation is guaranteed by only providing the user valid cable properties to choose from.

7.8. Exception Handling

If unexpected exceptions arise, its stack trace will be written out in order to allow finding and resolving the issue in future builds.

7.9. Logging

The application generates log files while running. These logs contain relevant information about occurring events within the application combined with timestamps.

7.10. Internationalisation

The language used within the GUI as well as the manual and readme files is english. Other languages are not supported.

7.11. Testability

The software is composed of different modules. These modules are tested separately. The system tests the system test plan will provide more information, and the system test report will contain the results.

7.12. Availability

The software is only distributed on GitHub.