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
(Niklas Huber, Niclas Hörber, Daniel Zichler, Kay Knöpfle, Nico Fischer)
Rotebühlplatz 41
70178 Stuttgart

Version	Date	Author	Comment
0.1	03.11.2020	Philipp Förster	created
0.2	08.11.2020	Philipp Förster	updated systemdesign / specification
0.3	09.11.2020	Team 3	minor changes to finalize SAS
1.0	23.04.2021	Niklas Huber	Removed Unit Tests

Based on: https://github.com/WAntonia/TINF18C_Team_3_DD2AML-Converter/tree/master/PROJECT/SAS

Contents

1. Introduction

The 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
There will be 10 preconfigured servers that cover different use cases for the clients.

1.1. Glossar

AutomationML	Automation Markup Language is an open standard data format for storing and exchanging plant planning data. [1]
CAEX	XML – like format to store data [2]
CLI	Command Line Interface (Console)
GUI	Graphical User Interface
Open65241	Open source implementation of OPC UA [3]

2. 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, the server is going to start and you should be able to connect to the server using the OPC UA Client UA-Expert.

2.1. System Environment

There will be 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 contains logic to interpret the configuration files and is able to create new server instances. For this, the open62541 stack library is used.
It is 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.

2.2. Software Environment

The open62541 stack is coded in the programming language C (C99). There should be no problem running it from Windows or Linux, because the C runtime is already included in the compiled program.

2.3. Quality Goals

The following quality goals for the OPC UA server farm shall be considered:

2.3.1. Usability

The 10 provided configurations for the servers should cover most of the users use cases. That way, users 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 systems file dialog to select a custom one.

2.3.2. Maintainability

Software principles like SOLID help to keep software maintainable and flexible. Future changes will not force a lot of refactoring, so the implementation of new features does not cause trouble.
We will try to focus on clean code and self-explanatory code.

2.3.3. 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.

3. Architectural Concept

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

3.1. Architectural Model

As mentioned earlier, the software is split into 2 parts: The logic component, which is responsible for the config import and server hosting and the view component, where all user interaction will take place.

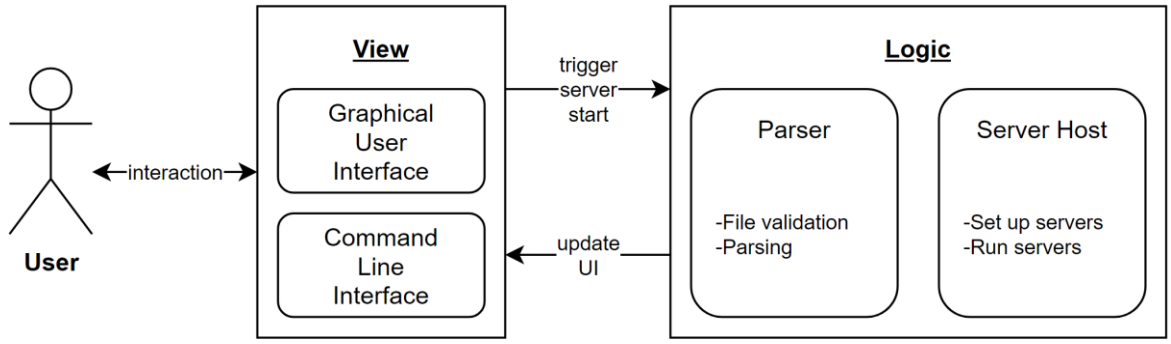

Figure 1 - Architecture Model

The view is needed to show the user all the important information. That can for instance be errors or important events from the system. It also deals with the users input and uses it to pass along data to the underlying logic component.
This is where the configuration files get parsed and the servers are set up corresponding to the analysed config. These functions can be called by the view when special events trigger, such as pressing a start button.

4. Systemdesign

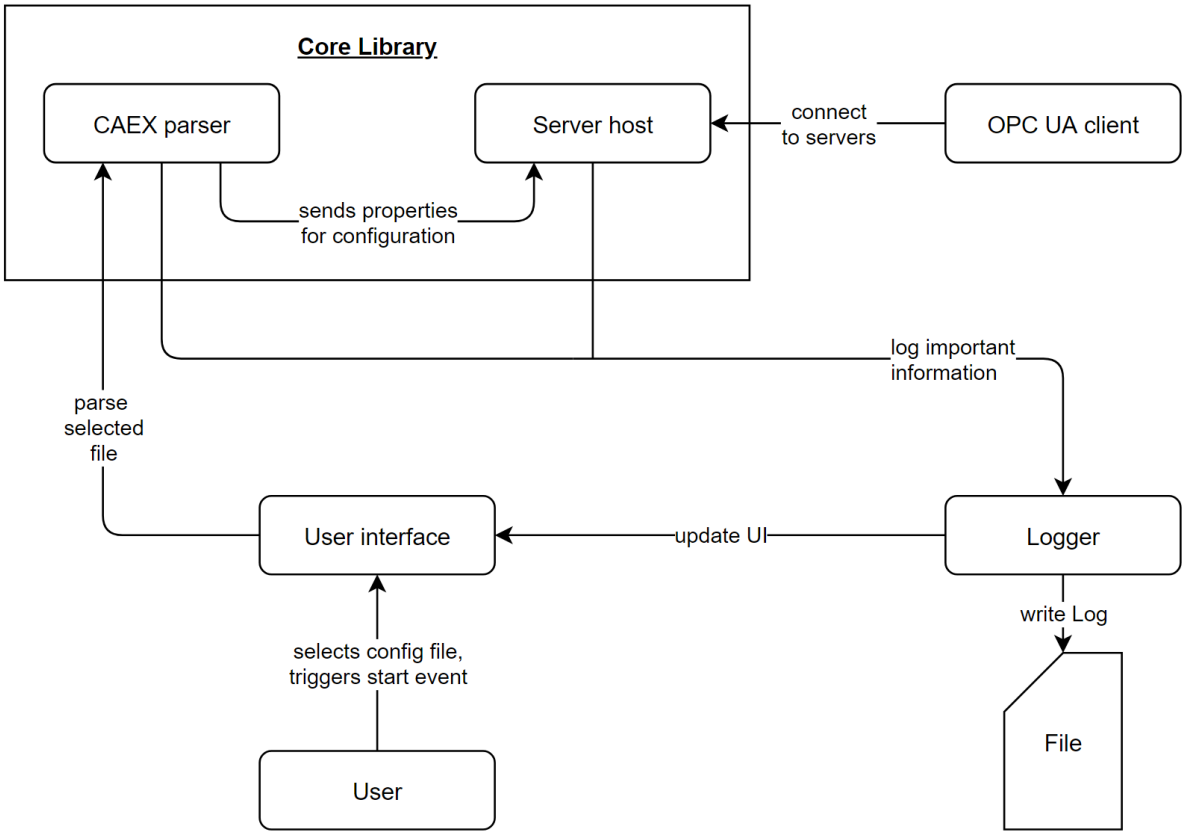

Figure 2 - System Design

The main part of the OPC UA server farm is the core library which can configure and host server instances according to a specified CAEX 3.0 configuration file.
At first, the user has to select a configuration and confirm the startup of a new server in the user interface. The UI will pass on the files path to the CAEX parser. Here, the file will be checked for syntax errors and is tried to be interpreted. In case of success, the extracted settings are passed to the server hosting component.
With the help of the open62541 stack, servers are set up using the settings from the parser. The server will run locally and is accessible under one IP address. To host multiple servers, different ports need to be used.
After starting one or more servers, the user can connect to them using the OPC UA Client UA - Expert.
All important information such as unexpected errors, but also successful operations like “file parsing complete” will be passed to the logger. It will update the UI to notify the user about the current state of the program and write the same information to a logfile.

5. Subsystemspecification

5.1. <MOD.001>: Core Library

This is the main module of the OPC UA server farm. The whole logic for parsing files and setting up servers is found here.

<MOD.001>	*Core Library*
System requirements covered:	*/NF40/*
Service:	• *Host an OPC UA server farm*
Interfaces:	• *Function to run a server with settings according to given configuration file*
External Data:	• *Configuration file*
Storage location:	

5.2. <SUBMOD.001.001>: Parser

This module is responsible for the parsing of the configuration file. It extracts all necessary information needed to set up a server.

<SUBMOD.001.002>	*CAEX 3.0 Parser*
System requirements covered:	*/LD10/, /LD20/, /NF20/, /LF20/, /LF30/*
Service:	• *Validate file* • *Parse file* • *Return settings*
Interfaces:	• *Function extract settings from config file*
External Data:	• *Configuration file*
Storage location:	

5.3. <SUBMOD.001.002>: Server Host

The server hosting module will configure and run servers using the settings it gets from the parser.

<SUBMOD.001.003>:	*Server Host*
System requirements covered:	*/NF10/, /NF30/, /LF30/, /LF40/, /LF50/*
Service:	• *Authenticate user* • *Set up servers* • *Host servers*
Interfaces:	• *Settings from parser*
External Data:	• *none*
Storage location:	

5.4. <MOD.002>: User Interface

The user interface is the module which users of the server farm get to see. It is needed to use the functions of the core library.

<MOD.004>	*User Interface*
System requirements covered:	*/LD20/, /LF10/*
Service:	• *User communication* • *Select configuration file*
Interfaces:	• *Configuration file* • *Command line or graphical user interface to start servers*
External Data:	• *none*
Storage location:	

5.5. <MOD.003>: Logger

The logger is used to save information about the systems current state in an external file which helps reproducing errors and bugs. All events will also be written to the console.

<MOD.005>	Logging
System requirements covered:	*/LF60/*
Service:	• *Logs messages or events to file and to the UI*
Interfaces:	• *Logging message*
External Data:	• *Log file*
Storage location:	

6. Technical Concepts

6.1. Persistence

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

6.2. User Interface

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

6.3. 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.

6.4. 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.

6.5. 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.

6.6. 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.

6.7. Logging

Important information as well as errors will be logged into a separate file and shown on screen.

6.8. 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.

6.9. Testability

To test the whole system working together, we will design a test plan for testmanger. It will be a manual test.
No Unit Tests will be implemented because a team member left the team.

6.10. Availability

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

7. Figures

8. References

[1] N. Baitinger, „DD2AML Converter - SAS,“ DHBW, 2020.

[2] Wikipedia, „CAEX – Wikipedia,“ [Online]. Available: https://de.wikipedia.org/wiki/CAEX. [Zugriff am 08 11 2020].

[3] Open62541, „open62541: an open source implementation of OPC UA,“ [Online]. Available: https://open62541.org. [Zugriff am 08 11 2020].