كلية الهندسة جامعه حلوان

جامعة حلوان
HELWAN UNIVERSITY

Faculty Of Engineering

Computer Engineering Department

Helwan University

AI Configuration Tool For Autosar Based Applications

Supervised by Dr.Amr ElSayed

Sponsored by SIEMENS

Presented by

Ahmed Taha Ahmed Mohammed

Sarah Saeed Fawzy Abdeltawab

Ahmed Samir Abdelsalam Mousa

Basma Abdelhaleem Abdelahaleem Ali

Acknowledgements

First and foremost , we would like to express our deepest gratitude to our project supervisor, **Dr/Amr ElSayed**, for their unwavering guidance, support and encouragement throughout the course of this project. Their expertise and insights have been invaluable in helping us navigate the complexities of this project and stay focused on our goals.

We would also like to extend our sincere thanks to our sponsoring organisation, **Siemens**, for their generous support and sponsorship of this project. We would like to express our gratitude to **Eng/ Mohamed Al Ansary** and **Eng/Khaled Mansour**, for their invaluable advice and technical support through this project. Their expertise and guidance have been instrumental in helping us overcome technical challenges.

Finally, each member in the team would like to express appreciation to each other, despite facing numerous challenges, the entire team persevered through difficult days and remained committed to doing their best. Thank you.

Abstract

The aim of this work is to simplify the development process and increase efficiency by eliminating the need for manual coding of Autosar based hardware modules' drivers. To achieve this, a GUI configuration tool has been designed to make it easier for tier-1 companies to use the generated code while the process of writing the applications.

The tool provides a user-friendly graphical interface that allows developers to specify the required inputs, such as the type of hardware module which can be Adc ,Can,Canf,Com,ComM, Dio, Spi,..etc, where each module has its corresponding containers, subcontainers and parameters. Once inputs are specified, the tool generates the corresponding Arxml file that contains the parameters' values of the configured module, with the ability to parse and edit the generated file, validate it to meet the required standards and specifications, apply queries , merge and split the files and finally generate the .c and .h files required for the driver. The tool also has the ability to use AI to generate Arxml files based on the user input , despite the limited training data, complexity of Arxml files which may contain multiple levels of abstraction and the limited hardware resources. The ultimate aim of this project is to improve the quality and reliability of autosar based systems while reducing development costs and time to market. With a GUI configuration tool, developers can quickly and easily configure the necessary parameters and generate the corresponding code, saving time and reducing the risk of errors.

abbreviations

AUTOSAR	Automotive Open System Architecture
ECUs	Electronic Control Units
SWCs	Software Components
MCAL	MicroController Abstraction Layer
ADC	Analog Digital Converter
OEMs	Original Equipment Manufacturers
MD	Module Definition (standard file)
ARXML	Autosar XML file.
NLP	Natural Language Processing.
LM	Language Model.
NER	Named Entity Recognition.
OOV	Out Of Vocabulary.
RNNs	Recurrent Neural Networks.
CNNs	Convolutional Neural Networks.
LSTM	Long Short Term Memory.
EOS	A special token representing the end of a sentence.
BOS	A special token representing the beginning of a sentence.
PPO	Proximal Policy Optimization
RLHF	Reinforcement learning using human feedback

Table Of Content

List Of Figures

Chapter 1: Introduction

1.1 What is Autosar?

AUTOSAR (Automotive Open System Architecture) is a standardised software architecture for automotive electronic control units (ECUs). It was developed by a group of automotive manufacturers, suppliers, and tool developers with the aim of simplifying the development and integration of automotive software.

AUTOSAR has implemented a layered architecture similar to OSI model.It has different layers to handle and abstract different operations of code. AUTOSAR is used for micro controllers which targets applications mostly in automotive space which utilises CAN, Flex Ray, Ethernet etc. Being used in applications based on micro controllers, it is developed with a view to use the least memory possible as micro controllers have resource constraints.

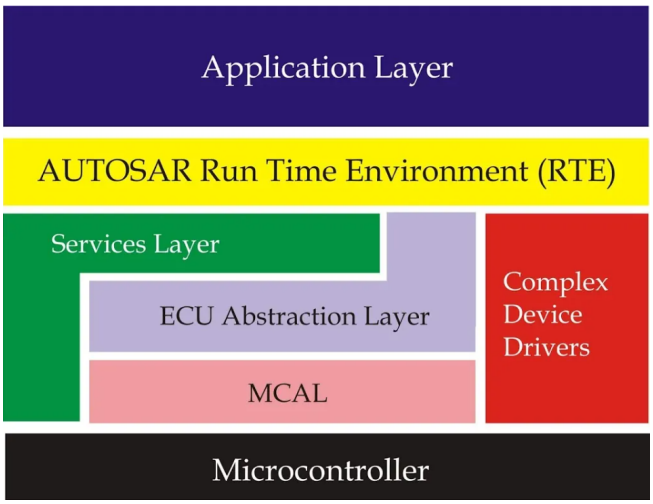


Figure 1: Simplified AUTOSAR Layered Architecture.

The above image is a simplified AUTOSAR layered architecture . It's called simplified because there are also deep layers in each block which are hidden.

- **application layer:** this layer has the application code which resides on top. It can have different application blocks called as software components for each feature which the ECU needs to support according to application. for example, the functions like power window and temperature measurement will have separate SWC. This is not a norm, but it depends on the Designer.
- **AUTOSAR RTE:** this is one of the most important layers in AUTOSAR, it provides communication between different SWCs and also between ECUs. application layer uses this layer while communicating with other layers below using ports.
- **service layer:** this layer provides different services for application to use. services like: system services, memory services, communication services,..etc.
- **ECU abstraction layer**:this layer provides ECU related abstractions. It contains different abstracted layers like I/O hardware abstraction layer, memory hardware abstraction, on board device abstraction ,etc. to make applications hardware independent.
- **MCAL:** this is MicroController Abstraction Layer, it has drivers which the above layers use to communicate with the microcontroller peripherals.

1.2 AUTOSAR specification for an ECU

we can take ADC module for example:

if we want to create a driver for ADC based on the autosar standards, we should follow the document “specifications of ADC driver” which contains the following:

- The required number of variables and their data types
- The required functions, including:
  - number of functions.
  - the standard name of each function.
  - the input parameters
  - return types.
- The standard name of enumerations, structures, symbolic constants, etc.

here’s an example from the SWS:

Service name:	Adc_Init	

Syntax:	`void Adc_Init(`
	`  const Adc_ConfigType* ConfigPtr`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to configuration set in Variant PB (Variant PC requires a NULL_PTR).
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Initializes the ADC hardware units and driver.	

Figure 2: a standard description of a function based on autosar SWS.

1.3 AUTOSAR benefits

listed below the problems that faced the automotive industry when writing a software for an ECU and how AUTOSAR provided a solution for it:

1. changes in hardware requires applying changes in the application

   after autosar:

   the application is totally separated from hardware, so changing the microcontroller will impact the application code.

2. Each company writes the code of software components from its perspective.

   after autosar:

   All companies can understand other companies' code because autosar provides standardisation, where maintenance and adding more features becomes more applicable than before.

3. In the early days of the automotive industry, OEMs were dealing with only one company to write the SWCs it needed. The motivation behind this was that there wasn't a standard to follow, which made it hard for OEMs to switch to another company.

   after autosar:

   There's a standard to follow , which makes it easier for OEMs to switch to other companies.

4. application was a dependent block of code sold by one company.

   <u>after autosar:</u>

   application is divided into a set of tasks or software components, each written based on autosar standards and sold separately by companies.

1.4 Types of companies in the Automotive industry

There are several types of automotive companies , each with a different focus and business model. here are some of the most common types:

1. **Original Equipment Manufacturers (OEMs):** these are the companies that **design and manufacture cars**, trucks, and other vehicles. They are often the biggest players in the automotive industry and include brands like Toyota,Ford, and General Motors.
2. **Tier 1 suppliers**: these are the companies that **supply parts and software components** directly to OEMs. They often specialise in specific areas, such as engine or transmission manufacturing. examples include Valeo, Bosch,..etc.
3. **Tier 2 suppliers:** these are the companies that manufacture the microcontrollers and its external hardware. These companies also **write the microcontroller drivers, supply tools** to tier 1 companies.

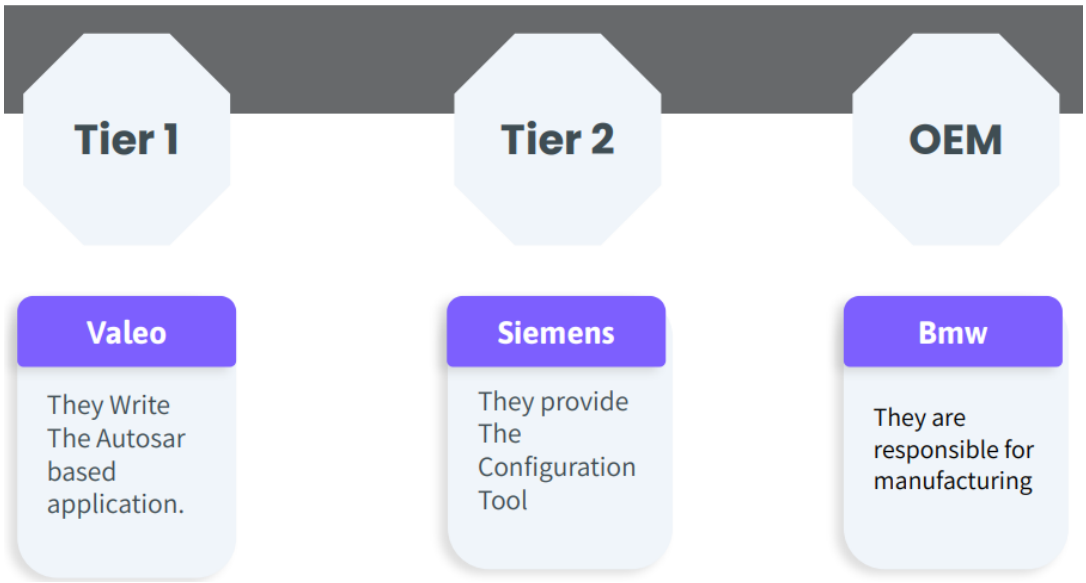


Figure 3: types of companies in the automotive industry.

1.5 Problem Statement

The automotive industry has become increasingly complex, with vehicles featuring more and more electronic systems and components. As a result developing and configuring these systems become a significant challenge, with engineers needing to navigate a complex web of hardware and software components. the risk of errors has been increased and the size of drivers too.

To address this problem, AUTOSAR configuration tools were developed by the OEMs and other automotive manufacturers as a way to simplify the process of configuring electronic components in a vehicle. These tools provide standardised configuration for different components in a microcontroller, making it easier for engineers to ensure that they work together seamlessly and on a standard. These tools also validate the configuration, reducing the risk of errors. and generate codes, reducing time and efforts.

Our task is to design an autosar configuration tool that prevents engineers from manually configuring each component of a vehicle's electronic system, which is a time-consuming and error prone process, provide validation for these configuration based on autosar specifications, generate source and headers, compile these files and finally have the ability to generate ARXML configuration files from a user's natural language input.

1.6 Objective and Motivations

This tool's role is important in solving some of the key problems faced by the automotive industry, particularly in the area of electronic systems development and integration. here some of the ways our tool has helped to address some challenges :

1. **Standardisation** : one of the main challenges in the automotive industry is the lack of standardisation across different components and interfaces. Our tool provides a **standardised interface for configuring different modules** , making it easier for engineers to ensure that they work together seamlessly.
2. **simplification**: before the existence of the AUTOSAR configuration tools, engineers had to manually configure each module in a vehicle electronic system, which is a time consuming and error prone process. Our tool simplifies this process by **providing a user-friendly interface** for configuring components, makes it easy to be edited, and automates the process of validating this configuration.

3. **validation**: our tool can **validate** the user's configuration based on the standard AUTOSAR "MD.arxml" , apply dependency checks , reducing the risk of errors and conflicts.
4. **code generation**: before the configuration tool, the hardware 's module driver was including all the features and functions of this module, which increased the size of the driver. Our tool **generates .c and .h files** which contain **only** the user's configured parameter.
5. Our tool is different from other configuration tools, as it's an **AI-Powered configuration tool**, where we applied a **new research** of how an AI model can generate ARXML configuration structure only from a human unstructured description. It's a new and hard task due to the **complexity of ARXML files and their dynamic length.**

Chapter 2 : Analysis and requirements

2.1 Problem Analysis

in order to arrive at the specific requirements, we wrote below, we did some steps to ideas and data about the proposed project as following:

- We did some analysis on existing tools used by automotive engineering teams, this helped us to understand the main components of the tool, and how the configuration process works.
- We studied the AUTOSAR standard as we tried to understand its comprehensive guidelines and specifications for developing a configuration tool.
- We had a lot of discussion with our supervisors in Siemens about the main components of our tool.
- We've conducted a lot of research in the AI feature.

2.2 Functional requirements

REQ-X	Description
1. provide AI to convert handwritten configuration into ARXML.	Providing the capability of using AI-based techniques to convert hand-written configuration information from different users written in different formats into arxml-based configuration files.
2. provide a user friendly GUI.	provide a user interface that allows users to configure modules based on autosar specifications.
3. generate ARXML files.	the tool should generate an ARXML file after the user has finished all the configuration.

4. parse ARXML files.	The tool should have the ability to parse ARXML files in order to use the parsed data in the generation of the configuration ARXML.
5. apply queries and edit ARXML files.	the user should have the ability to view the configured ARXML file, search for a specific parameter's value, get the number of the configured containers/subcontainers.
6. Validate ARXML files.	The configured ARXML files should be validated in order to check its compliance with the MD (Module Definition), consistency checks should applied in order to check the correctness of parameters dependency.
7. merging and splitting ARXML files.	The tool should have the ability to merge two files/containers to more than one file. For splitting, one file can be splitted into more than one file.
8. generate .c and .h code.	the tool should have the ability to generate .c and .h files based on the user's configuration and the required target.
9. integrate 3rd party tools.	The tool should contain 3rd party tools, such as compiler, in order to flash and burn the generated source and header files.

2.3 Non Functional requirements

1. **usability**

   description

   the tool should be easy to use and navigate.

2. **performance**

   description

   the tool should be able to handle software systems without slowing down or crashing.

3. **Scalability**

   description

   the tool should be able to handle configuration for a wide range of modules and generate their corresponding .c and .h codes.

4. **Reliability**

   description

   the tool should be reliable and consistent, producing the same results every time it's used. also make sure that the configuration files are generated as user's specification whether through the GUI interface or AI interface.

5. **Compatibility**

   description

   The tool should be compatible with other software tools and systems, such as Compilers, Template Code generators.

6. **Maintainability**

   description

   The tool should be easy to maintain and update, with clear documentation.

7. **Extensibility**

   description

   the tool should be extensible, allowing developers to add new features and functionality as needed.

2.4 Functional Requirements specifications

2.4.1 System Stakeholders

- **Automotive Engineers**: these are the primary stakeholders who are responsible for designing and developing automotive systems. They rely on the configuration tool to simplify and optimise the configuration process.
- **Software developers (users)**: these are responsible for developing the software components that make up an Autosar-based System. They rely on the configuration tool to ensure that the software components are configured properly.
- **Project Managers**: these are responsibles for overseeing the development of automotive systems and ensure that they meet project goals. They rely on the configuration tool to make sure that the configuration process is efficient.

2.4.2 Actors

- **AI Model**: takes the configuration description as an input and generates the corresponding Arxml file.
- **GUI** : allow users to make configuration for different modules and generate the corresponding configuration, header and source files.
- **Developers (users)**: has the ability to choose between AI interface or the Graphical user interface to make the configurations.

2.4.3 Use Cases

- **generate an Arxml file.**

  in order to generate the configuration Arxml file, the user should follow the following **requirements**:

  1. **open** the tool GUI.
  2. **select** the module.
  3. **make** configuration (set values to parameters)
  4. **click** Generate Xml icon.
  5. the configuration Arxml file is saved in the **instance location.**

- **generate source and header files.**

  in order to generate .c and .h codes, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **select** the module.
  3. **make** configuration (set values to parameters)
  4. **click** Generate Xml icon.
  5. **click** Generate code.
  6. the generated source and header files are saved where the configuration Arxml file is saved.

- **edit files.**

  in order to view and edit the files, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **click** File Editor.

- **check dependency.**

  in order to validate the configured parameters based on dependency, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **select** the module.
  3. **make** configuration (set values to parameters).
  4. **click** Generate Xml icon.
  5. **click** Dependency, the files can’t be generated if the dependency of one parameter is not correct.

- **split files.**

  in order to split an Arxml file based on the short name, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **click** split files.
  3. **choose** the file path.
  4. choose the **short name**.

5. The split file is saved where the configuration Arxml file is saved.

- **merge files.**

  in order to merge two Arxml files, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **click** Merge files.
  3. **select** the first file path, the second file path.
  4. **select** the path where the merge file will be saved at.
  5. **click** Merge.
  6. the merge file is saved where the configuration Arxml is saved.

- **compile files.**

  in order to compile the project with the generated files, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **select** the module.
  3. **make** configuration (set values to parameters)
  4. **click** Generate Xml icon.
  5. **click** Generate code.
  6. **click** compile.
  7. The tool will generate .hex and .elf files at the **build** directory where **CMakeLists.txt** file exists.

- **generate files using AI.**

  in order to use the AI feature for configuration Arxml generation, the user should follow these **requirements**:

  1. **open** the tool GUI.
  2. **click** on intelligent generation.
  3. **write** the prompt or use one of the examples.
  4. **set** number of beams.
  5. **click** generate.
  6. **click** json or xml to view the code.
  7. the user has the ability to generate the configuration Arxml, generate source and header files and compile.

2.5 System Development Tools

2.5.1. AI libraries and frameworks:

The tool may require access to popular python libraries and frameworks such as: Huggingface, Pytroch, Transformers, Beautifulsoup, Sklearn, Json, Wandb, Evaluate, Numpy, Pandas.

2.5.2 Integrated Development Environments:

The tool is developed by NetBeans IDE.

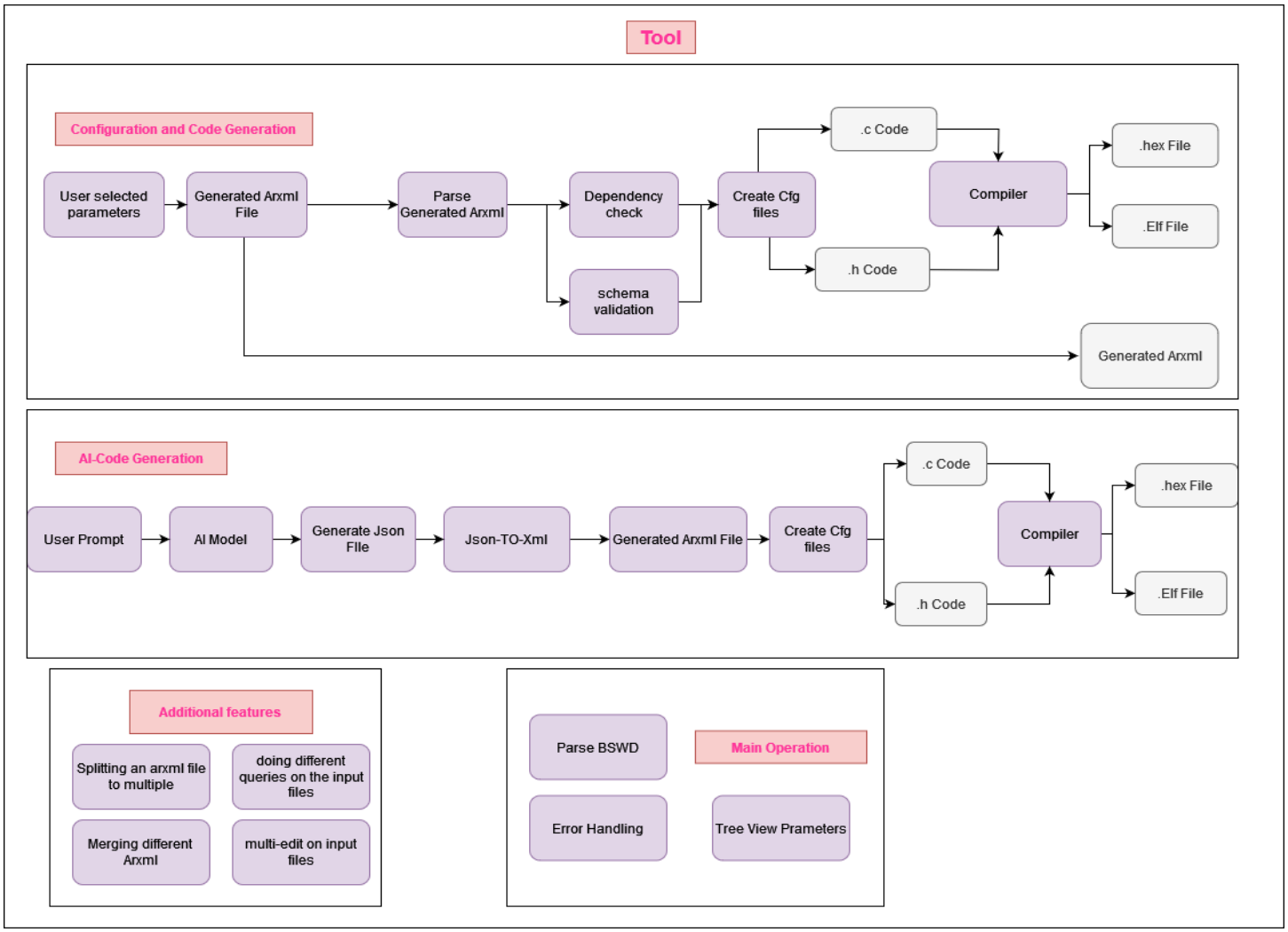
2.5.3 Cloud based development environments:

The ai feature is developed and tested through cloud environments such as : Google Clab, Kaggle.

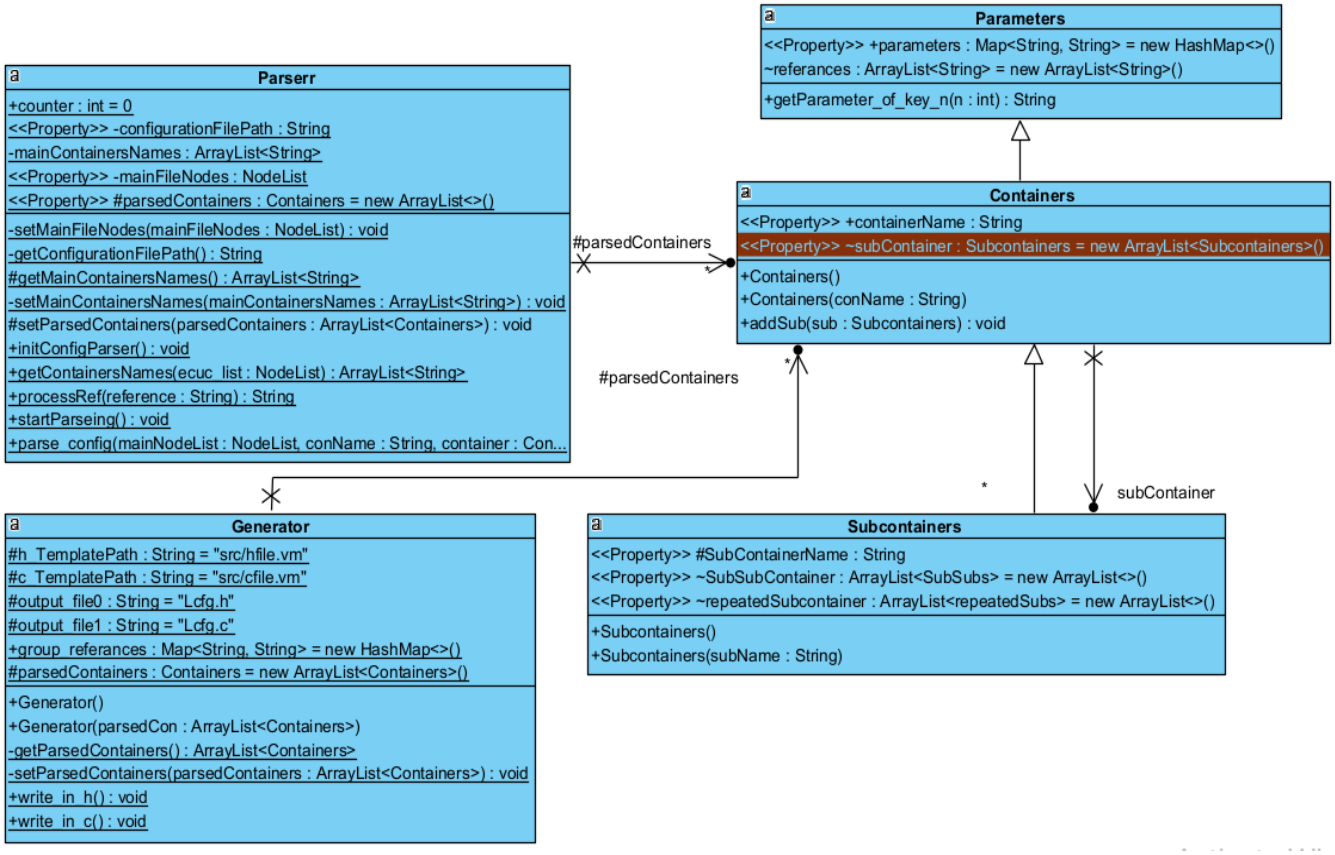
2.5.4 Collaborative Environments:

The tool uses collaborative environments that enables the tool developers to share code and make contributions easily ,such as **Github**.

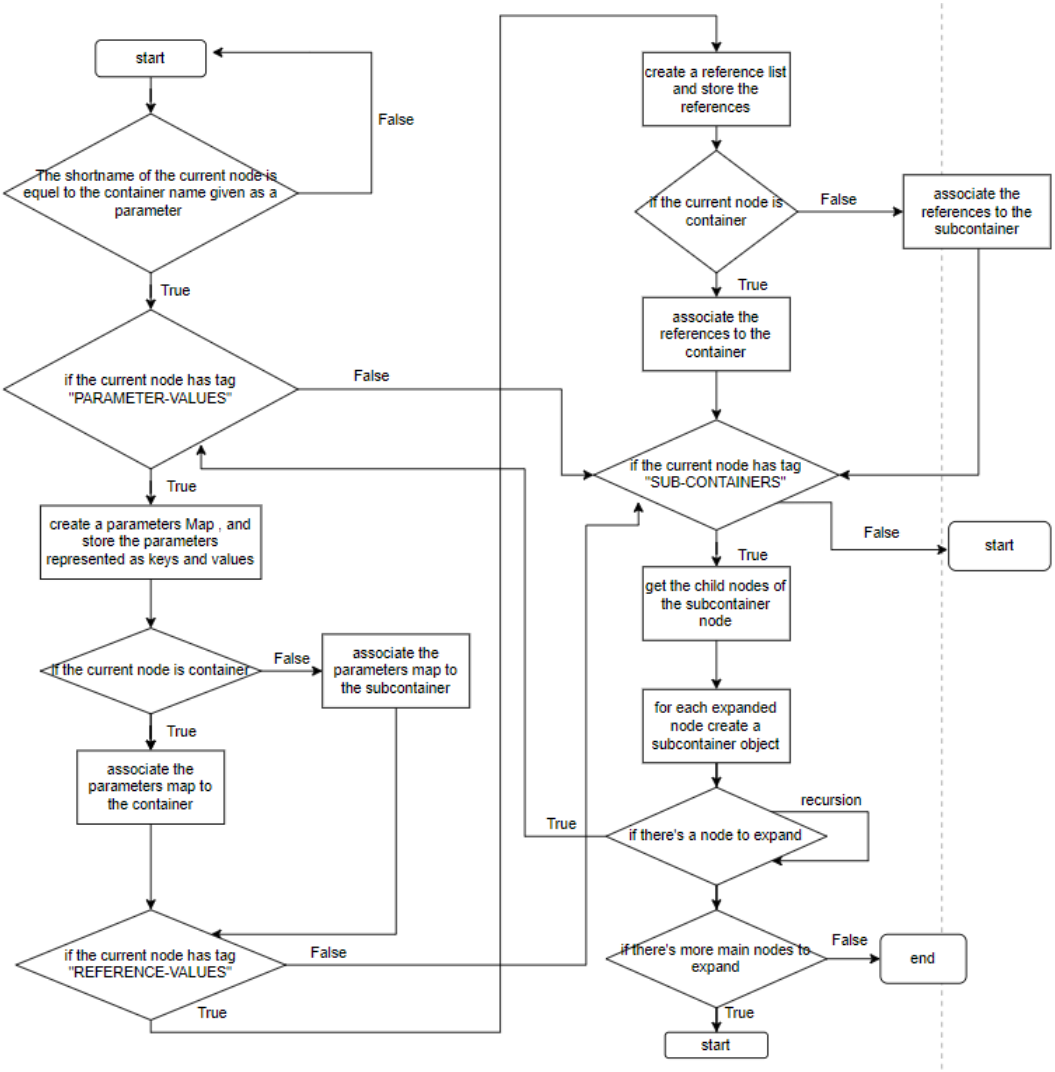
2.6 Software Design



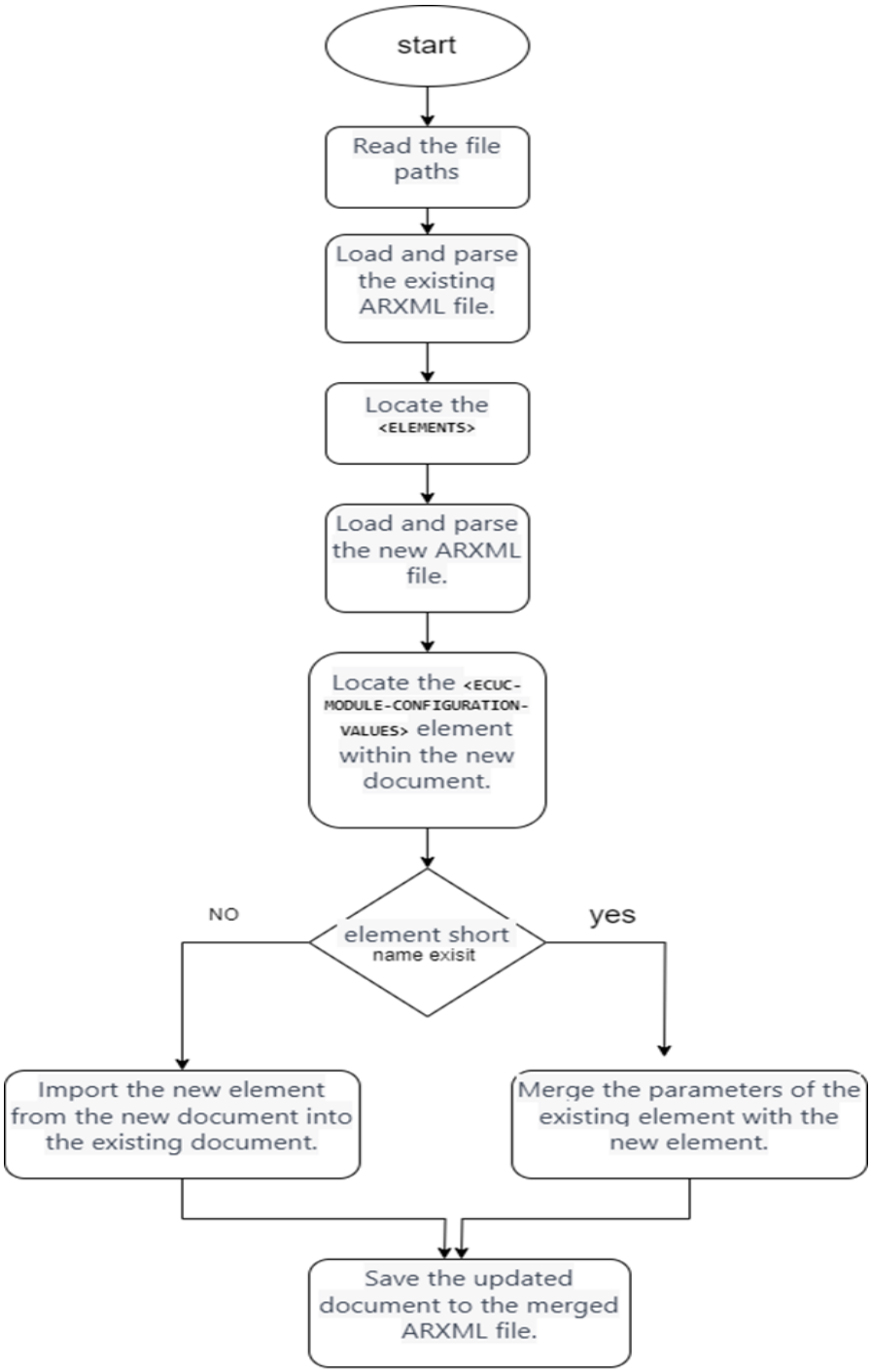
- **configuration parser class diagram.**



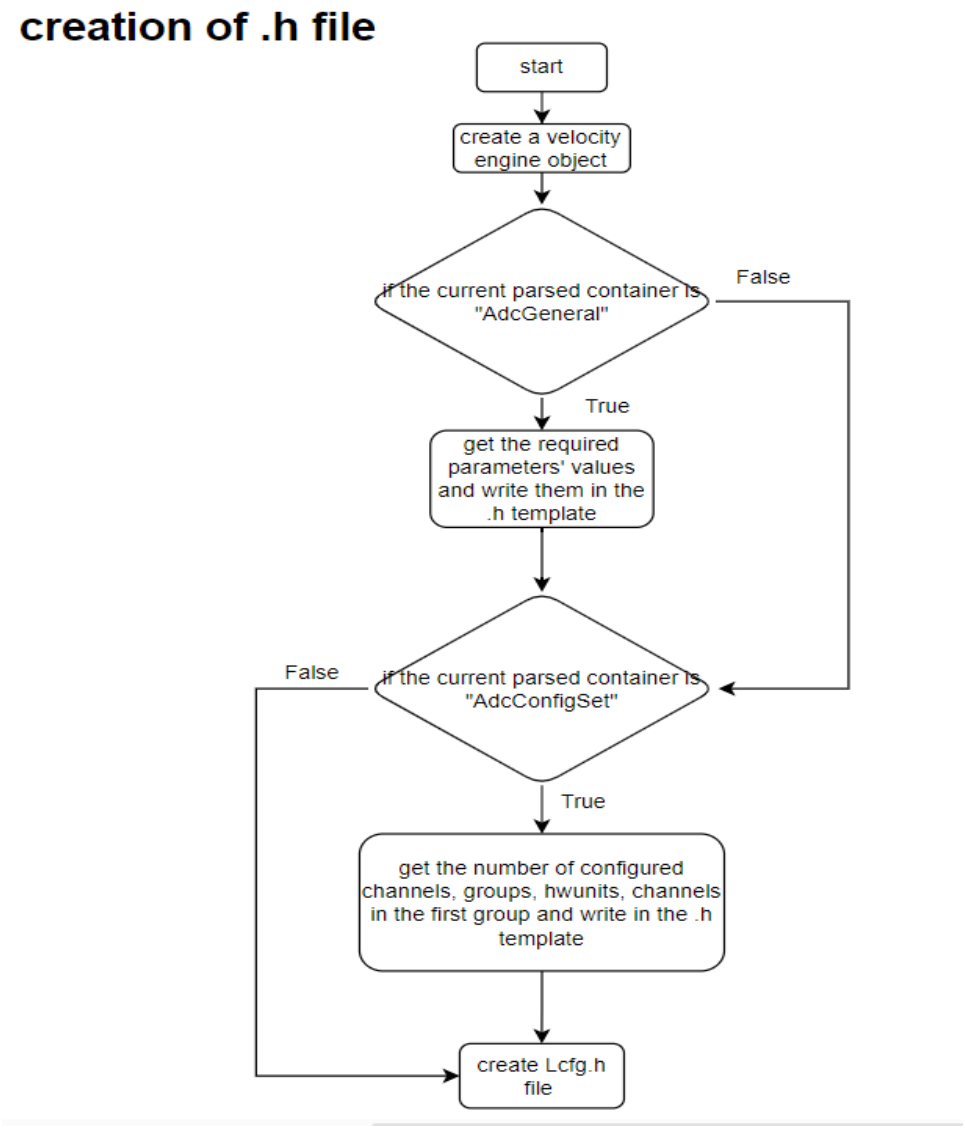
- **configuration parser code flow.**



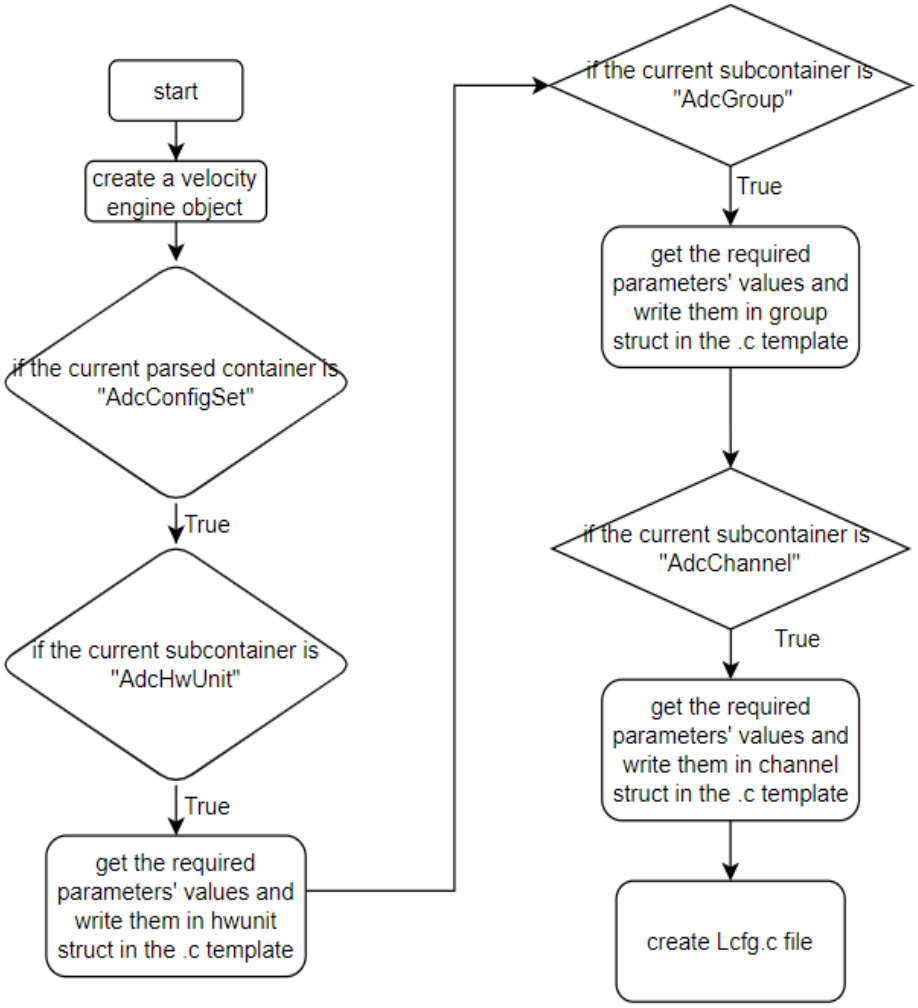
Parser Code Flow.



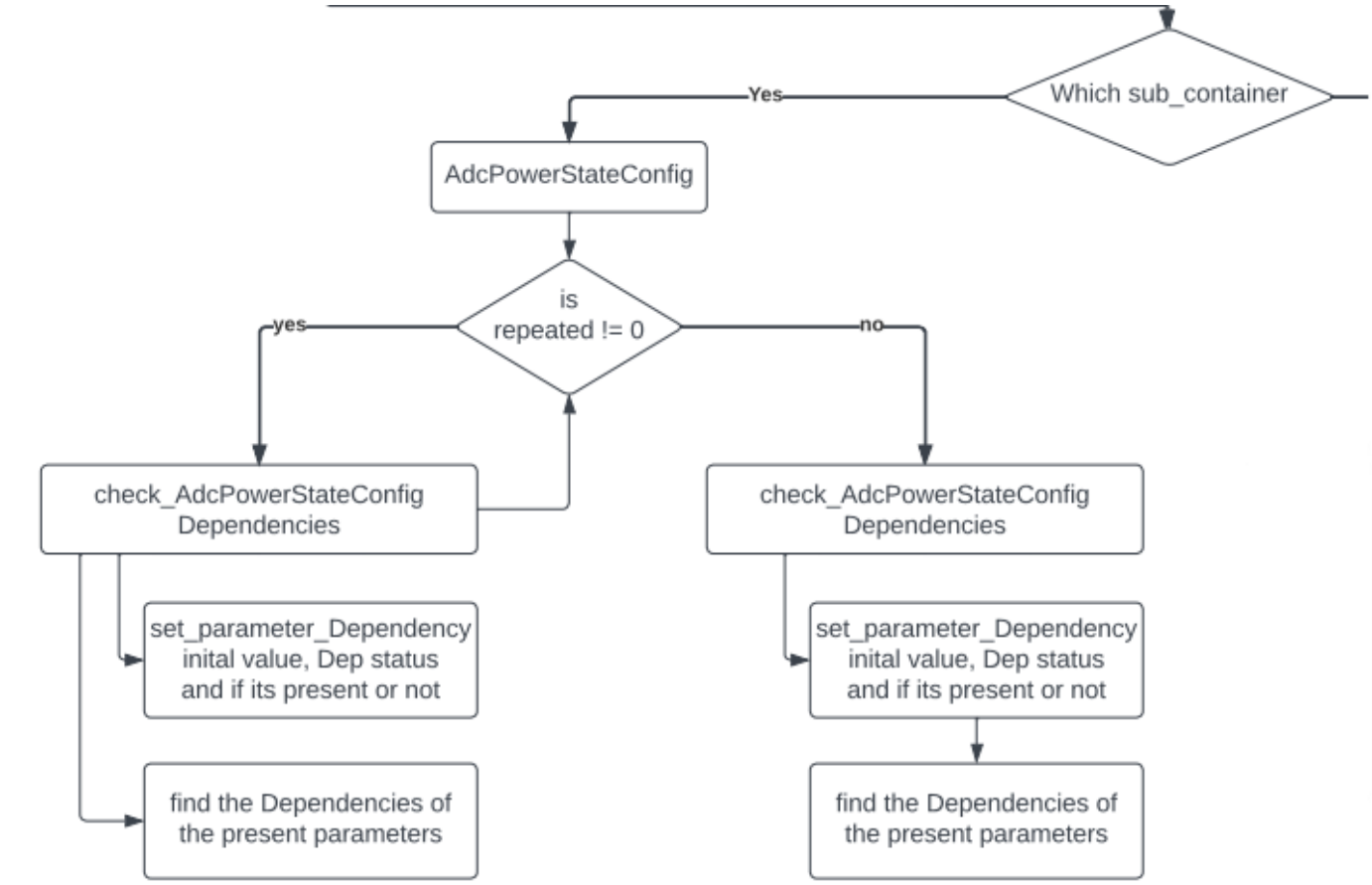
- **Generator code flow.**

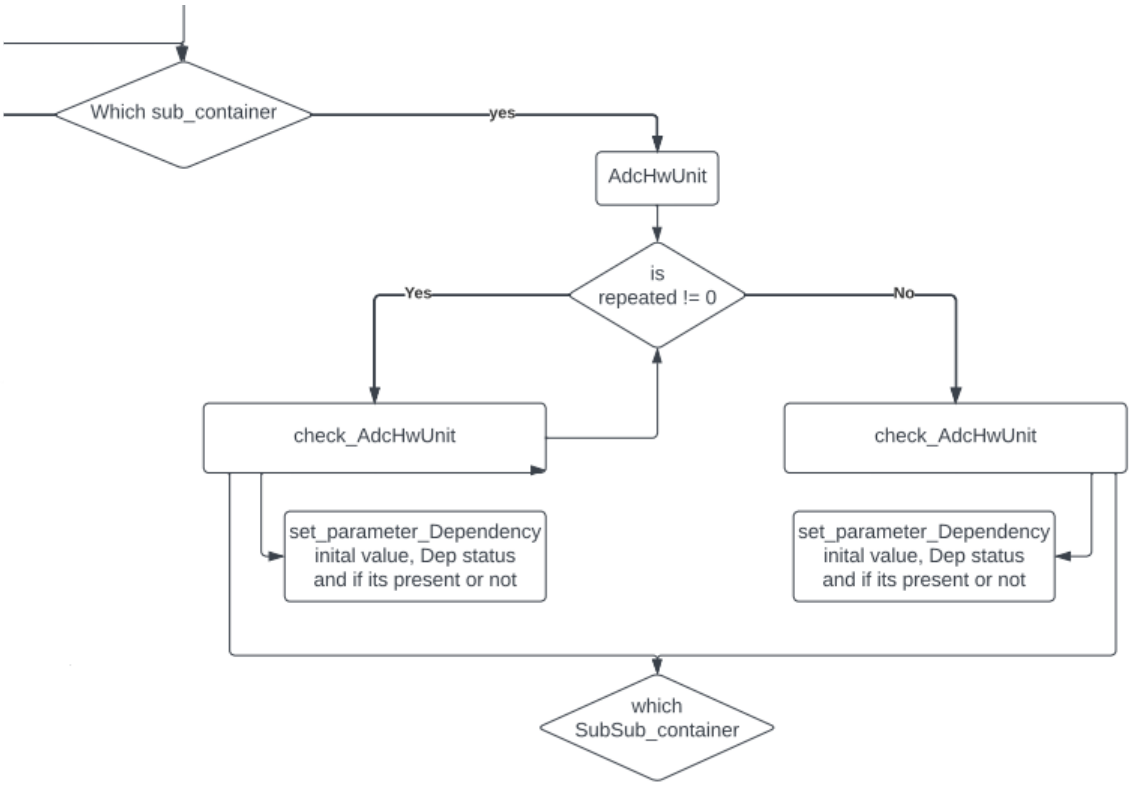


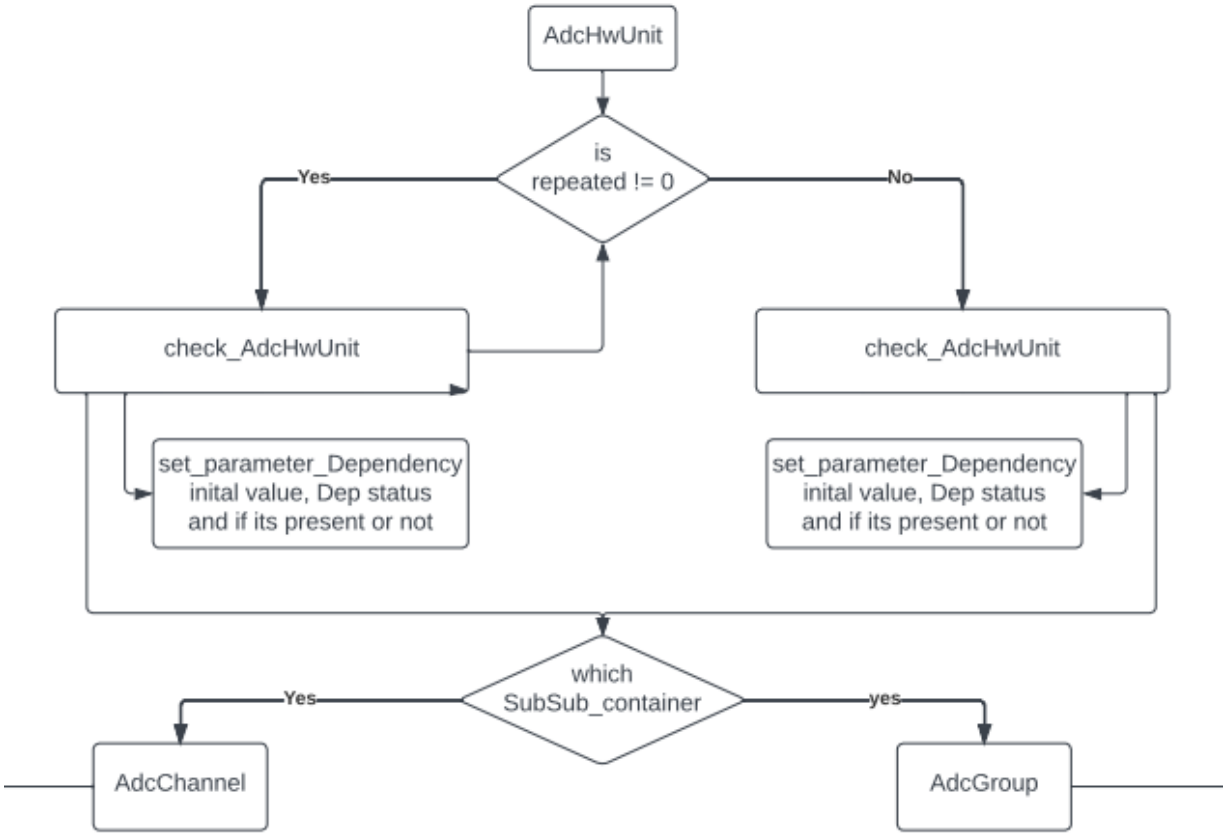
creation of .c file

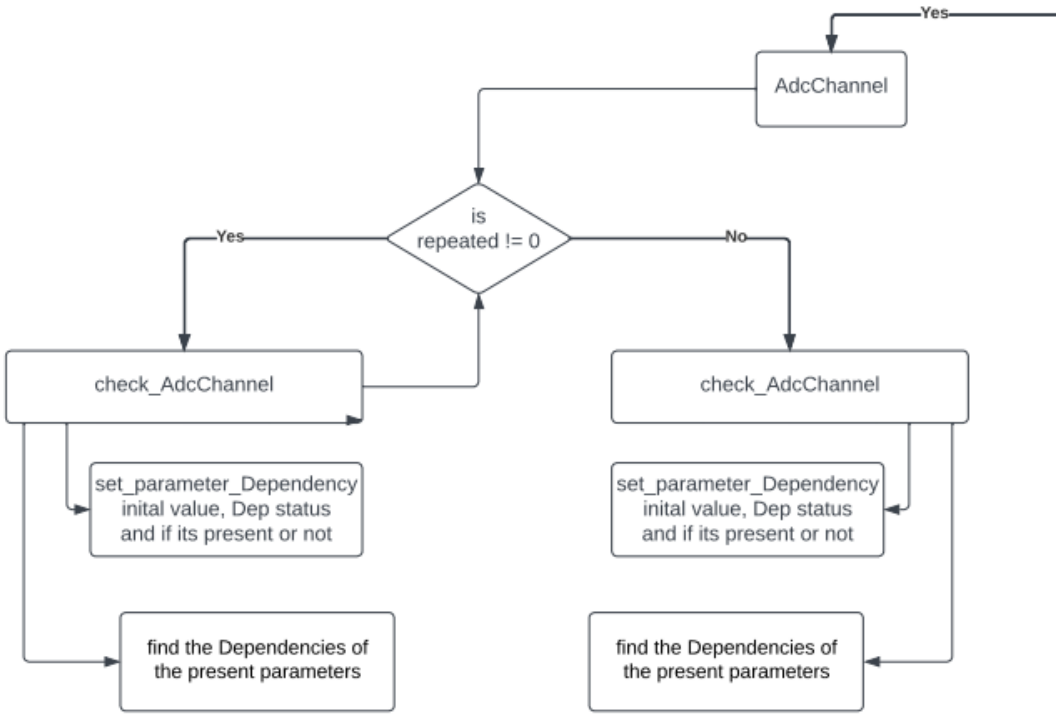


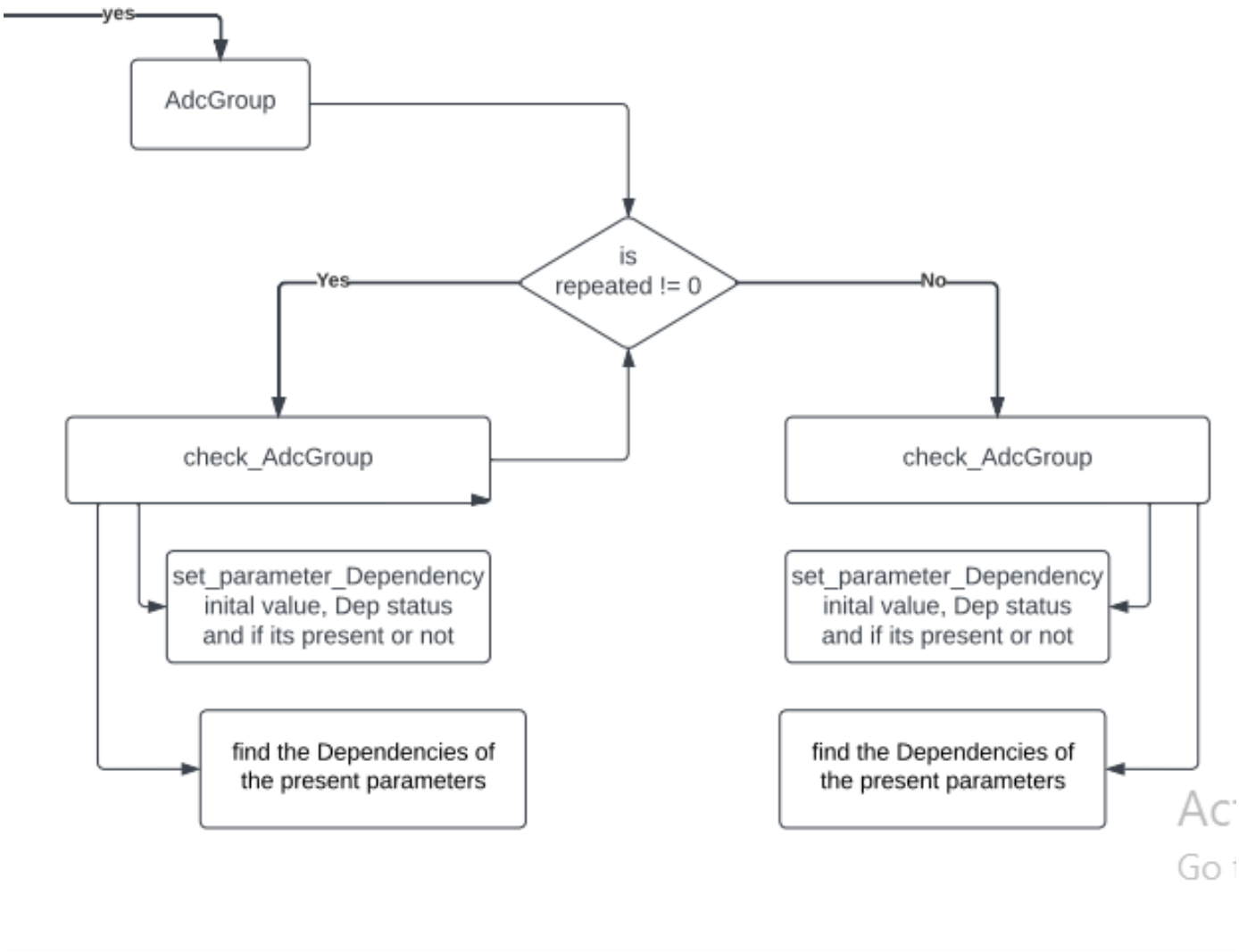
Consistency checks diagrams.


Which sub_container
Yes
AdcPowerStateConfig
is
repeated != 0
yes
no
check_AdcPowerStateConfig
Dependencies
check_AdcPowerStateConfig
Dependencies
set_parameter_Dependency
inital value, Dep status
and if its present or not
set_parameter_Dependency
inital value, Dep status
and if its present or not
find the Dependencies of
the present parameters
find the Dependencies of
the present parameters

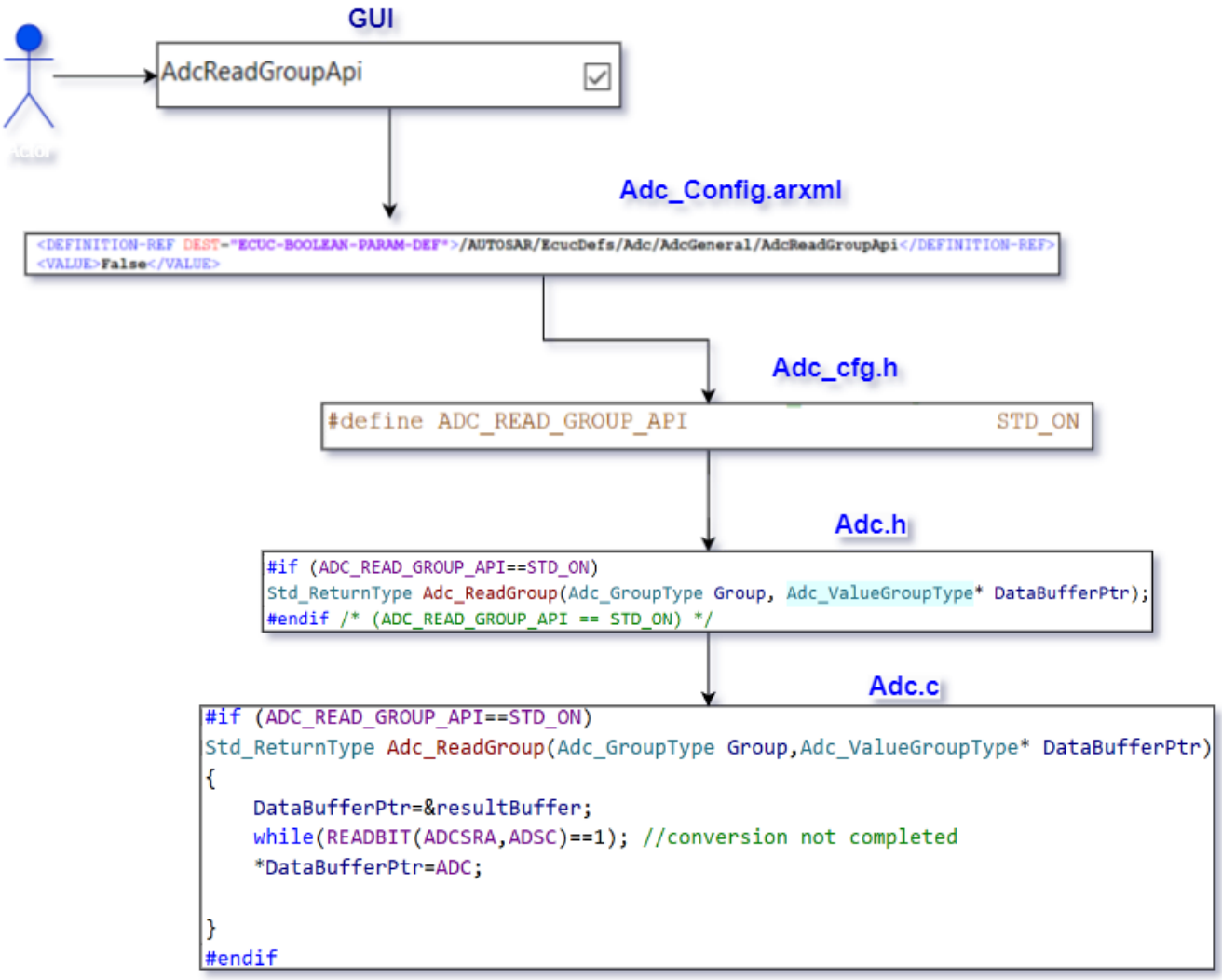

Which sub_container
yes
AdcHwUnit
is
repeated != 0
Yes
No
check_AdcHwUnit
check_AdcHwUnit
set_parameter_Dependency
inital value, Dep status
and if its present or not
set_parameter_Dependency
inital value, Dep status
and if its present or not
which
SubSub_container

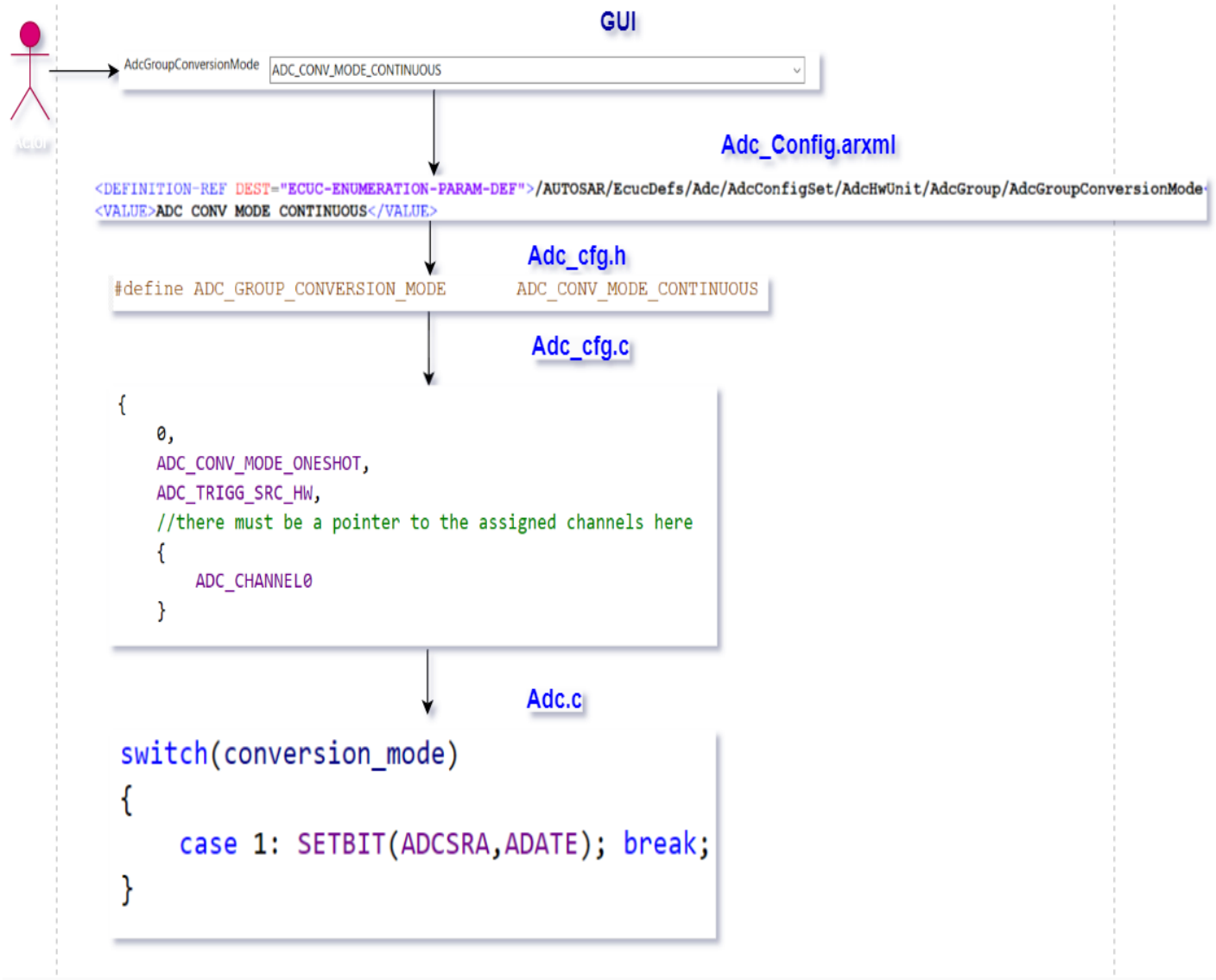

AdcHwUnit
is
repeated != 0
Yes
No
check_AdcHwUnit
check_AdcHwUnit
set_parameter_Dependency
inital value, Dep status
and if its present or not
set_parameter_Dependency
inital value, Dep status
and if its present or not
which
SubSub_container
Yes
yes
AdcChannel
AdcGroup


Yes
AdcChannel
is
repeated != 0
Yes
No
check_AdcChannel
check_AdcChannel
set_parameter_Dependency
inital value, Dep status
and if its present or not
set_parameter_Dependency
inital value, Dep status
and if its present or not
find the Dependencies of
the present parameters
find the Dependencies of
the present parameters

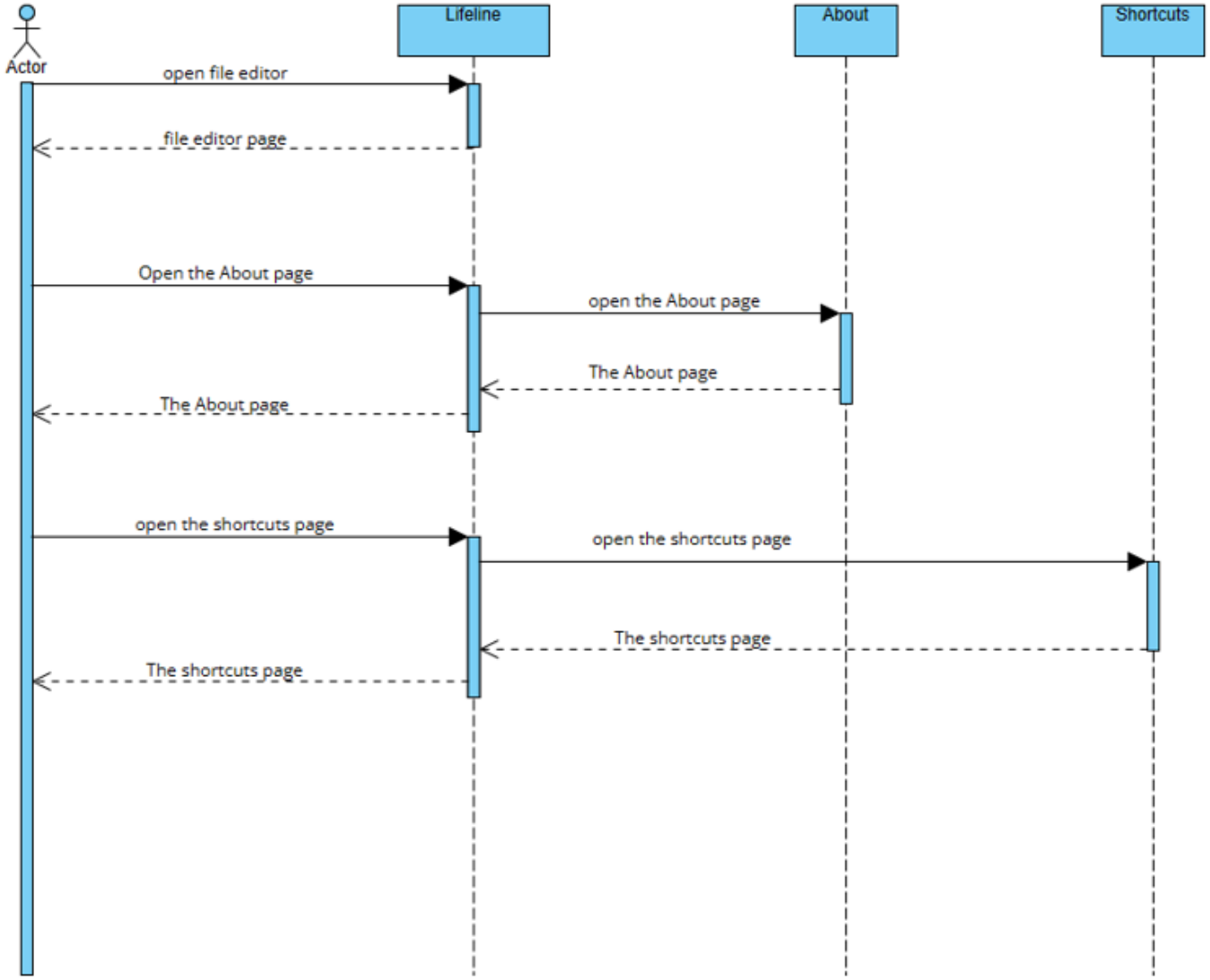

yes
AdcGroup
is
repeated != 0
Yes
No
check_AdcGroup
check_AdcGroup
set_parameter_Dependency
inital value, Dep status
and if its present or not
set_parameter_Dependency
inital value, Dep status
and if its present or not
find the Dependencies of
the present parameters
find the Dependencies of
the present parameters

Generator sequence diagrams.

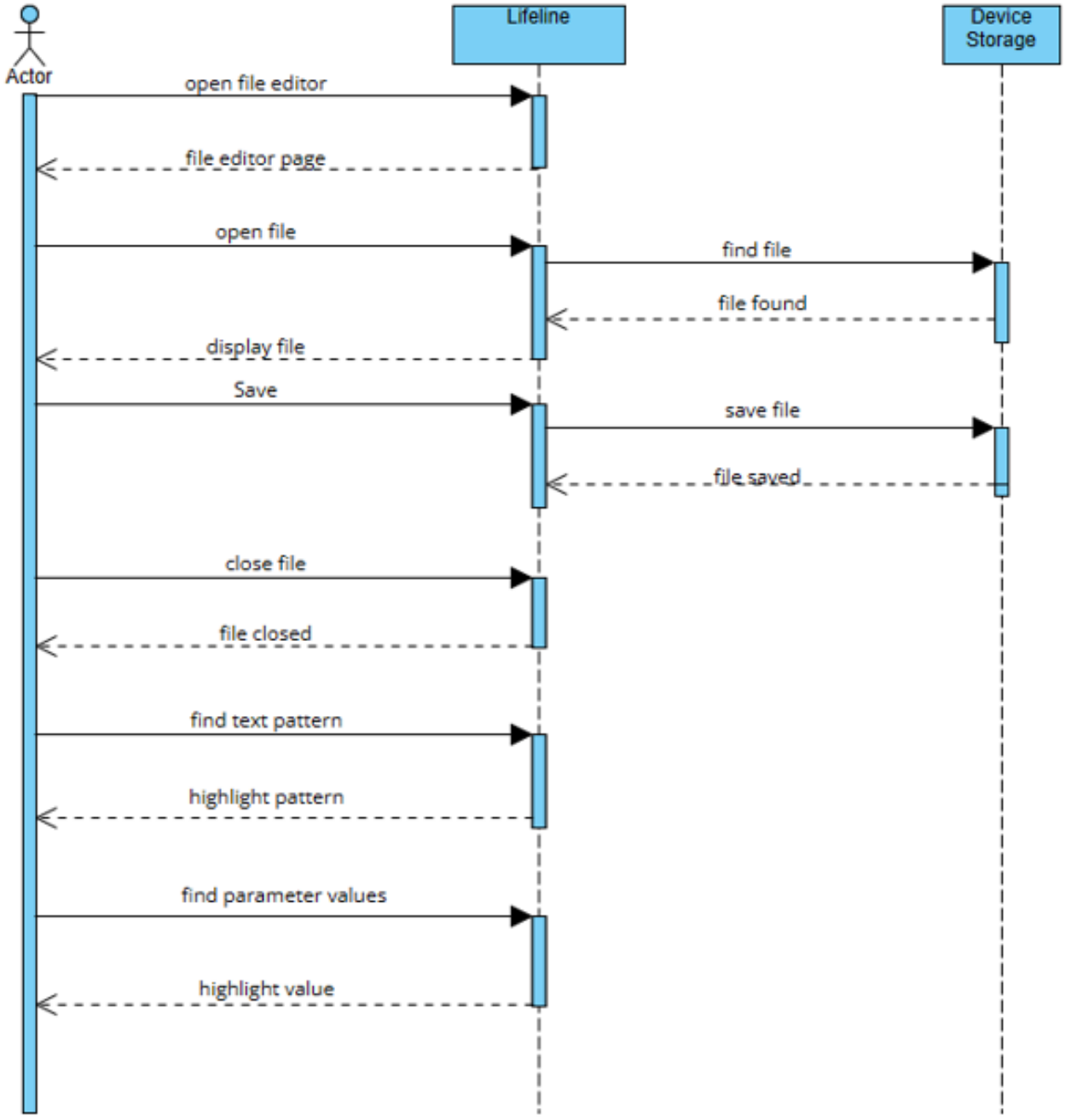



GUI
AdcGroupConversionMode
ADC_CONV_MODE_CONTINUOUS
Adc_Config.arxml
<DEFINITION-REF DEST="ECUC-ENUMERATION-PARAM-DEF">/AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcGroupConversionMode
<VALUE>ADC CONV MODE CONTINUOUS</VALUE>
Adc_cfg.h
#define ADC_GROUP_CONVERSION_MODE ADC_CONV_MODE_CONTINUOUS
Adc_cfg.c
{
0,
ADC_CONV_MODE_ONESHOT,
ADC_TRIGG_SRC_HW,
//there must be a pointer to the assigned channels here
{
ADC_CHANNEL0
}
Adc.c
switch(conversion_mode)
{
case 1: SETBIT(ADCSRA,ADATE); break;
}

GUI sequence diagrams.



GUI sequence diagrams.



2.5 MVC Design Pattern

The MVC design pattern was employed in this project to enhance the organisation, modularity, and maintainability of the ARXML generator. MVC divides the application into three major components: the “Model”, the “View”, and the “Controller”. Each component has its distinct responsibilities, which helps in separating concerns and promoting code reusability.

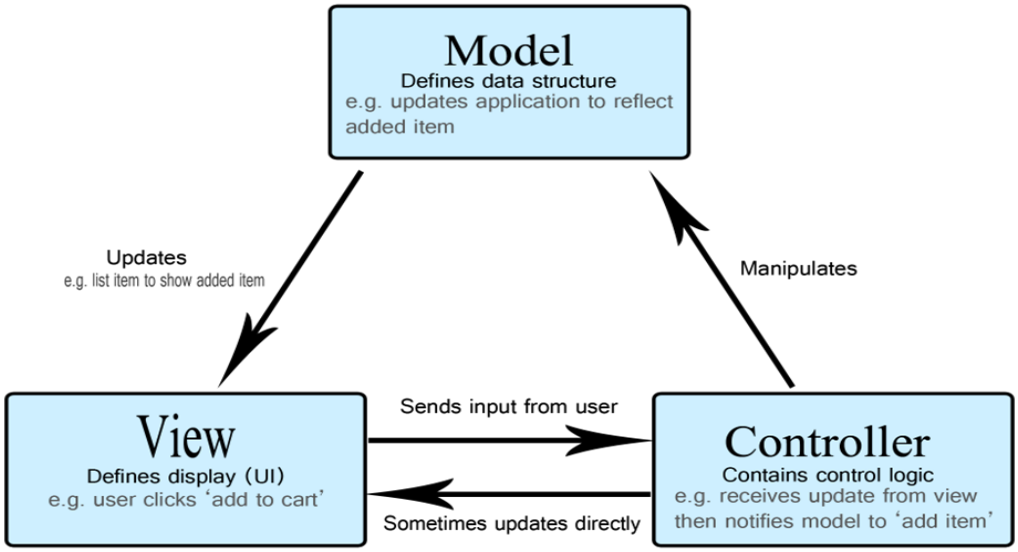


Figure 4: MVC description.

2.5.1. Model

The **Model** component represents the data and logic of the application. In the context of the ARXML generator, the Model encapsulates the core functionalities related to ARXML generation, including data structures, algorithms, and business rules. It manages the state and behaviour of the application,

ensuring consistency and integrity during the generation process. The Model component provides an interface for interacting with the underlying data and contains the necessary methods and operations to manipulate and generate ARXML files.

2.5.2. View

The View component handles the presentation and user interface of the application. In this project, the View is responsible for displaying the user interface elements, receiving user input, and presenting the generated ARXML files to the user. It provides a visual representation of the data and facilitates user interaction with the ARXML generator. The View component is designed to be user-friendly, intuitive, and aesthetically pleasing, ensuring a seamless user experience.

2.5.3. Controller

The Controller component acts as the intermediary between the Model and the View. It receives user input from the View and communicates with the Model to trigger appropriate actions and update the application's state. In the ARXML generator, the Controller manages the flow of data and coordinates the interactions between the user interface, the ARXML generation logic, and the code generation module. It ensures that user input is properly processed, validates the input data, and triggers the generation process based on the user's commands.

The MVC design pattern promotes loose coupling between the components, enabling independent development, testing, and

maintenance of each component. It enhances code reusability and modularity, allowing for easy extension and modification of specific components without affecting others. By separating concerns and enforcing a clear separation of responsibilities, the MVC design pattern simplifies the development process and enhances the overall maintainability of the ARXML generator.

The implementation of the MVC design pattern in this project followed established best practices and design principles. The Model, View, and Controller were implemented as distinct modules, with well-defined interfaces and interactions. Communication between components was established through well-defined interfaces and event-driven mechanisms, ensuring a decoupled and flexible architecture.

In our project:

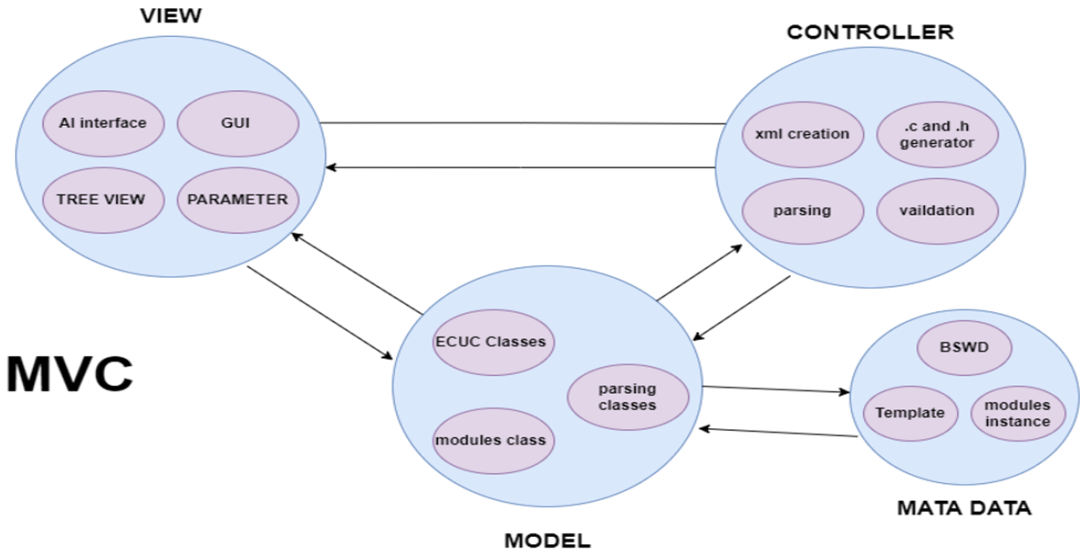


Figure 5: MVC components of the tool.

Overall, the use of the MVC design pattern in the ARXML generator project promotes code organisation, modularity, and maintainability. It provides a solid foundation for future enhancements, scalability, and extensibility of the application.

Chapter 3 : Software Module

3.1 ARXML Creation

According to AUTOSAR Methodology, the configuration process contains 4 steps in more detail:

- Configure System (parsing BSWD).
- Extract ECU-Specific Information (split it into modules and present each one in GUI).
- Configure ECU (choose ECU and set parameter values).
- Generate Executable.

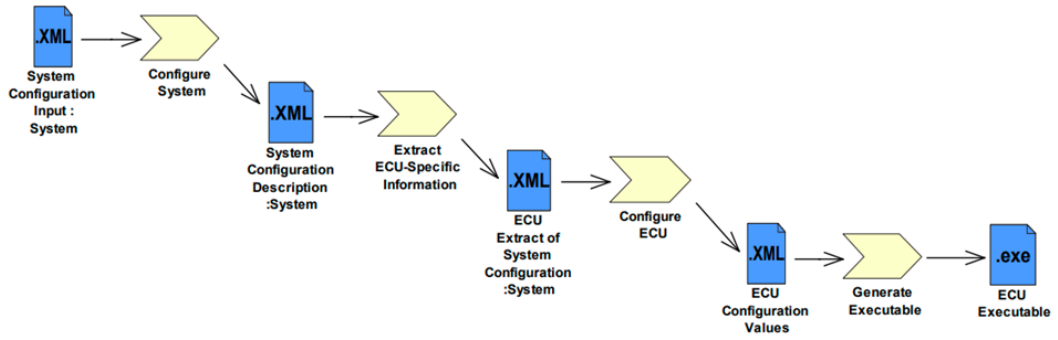


Figure 6: xml creation to ECU executable generation flow.

The configuration process of an ECU starts with the splitting of the System Description into several descriptions, each containing all information about one single ECU. This ECU extract is the basis for the ECU Configuration step. Within the ECU Configuration process, every single module of the AUTOSAR Architecture can be configured for the special needs of this ECU. Because of a quite complex AUTOSAR Architecture, modules, and interdependencies between the modules, tool support is required: **AUTOSAR ECU Configuration Editor(s)**.

The tool strategy and tooling details for the ECU Configuration are out of the scope of this specification. Nevertheless, tools need knowledge about ECU Configuration Parameters and their constraints such as configuration class, value range, multiplicities, etc. This description is the input for the tools. The description of configuration parameters is called **ECU Configuration Parameter Definition**. To make sure, that all tools are using the same output format within the configured values of the parameters, the ECU Configuration Value description is also part of this specification The ECU Configuration Value description may be on one hand the input format for other configuration tools (within a toolchain of several configuration editors) and on the other hand, it is the basis of generators. The configured parameters are generated into ECU executables. This is the last step of the configuration process and again out of the scope of this specification.

3.1.1 ECU Extract of System Description

ECU Configuration can only be started once a plausible System Configuration Description and the corresponding ECU extract has been generated (see Figure). Details on the System Configuration Description. The System Configuration Description contains all relevant system-wide configurations, such as:

- ECUs present in the system.
- Communication systems interconnecting those ECUs and their configuration.

- Communication matrices (frames sent and received) for those communication systems.
- Definition of Software Components with their ports and interfaces and connections (defined in the SWC Description and referenced in the System Configuration Description).
- Mapping of SWCs to ECUs The ECU Extract of the System Configuration is a description in the same format as the System Configuration Description, but with only those elements included that are relevant for the configuration of one specific ECU.

3.1.2 ECU Configuration

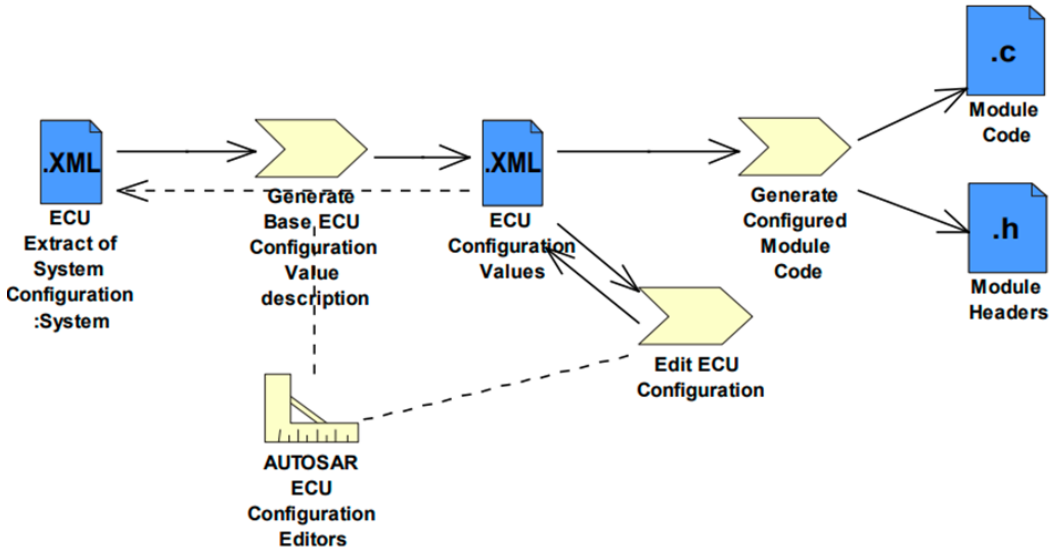


Figure 7: ECU configuration flow.

here's the steps to for ecu configuration:

- Generate Base ECU Configuration Value description.

- Edit ECU Configuration.
- Generate Configured Module Code.

All three activities use a single work product, the ECU Configuration Values, which contains (i.e. references) all the configuration information for all BSW modules on the ECU. In order to better understand the three different activities an introduction to configuration classes is essential. In a real implementation of a BSW module, all configuration parameters are most likely not the same configuration class. I.e it will be a mix of parameters with different configuration classes within a BSW module. These three activities are introduced in detail in later sections, but the first is an introduction to ECU Configuration Value description and configuration classes.

3.1.3 ECU Configuration Value description

The ECU Extract of System Configuration only defines the configuration elements that must be agreed upon between ECUs. In order to generate a working executable that runs on the ECU, much more configuration information must be provided. The remaining part of the configuration is about configuring all BSW modules within the ECU. Typical BSW modules within an ECU can be RTE, Com, Can, OS, NVRAM, etc. There are also dependencies between BSW modules to consider when configuring the ECU. When the configuration is done, the generation of configuration data takes place. I.e. There are both configuration editors and configuration generators involved in the process. In order to obtain consistency within the overall configuration of the ECU, AUTOSAR has defined a single

format, the ECU Configuration Value description to be used for all BSW modules within an ECU. Both configuration editors and configuration generators are working toward ECU Configuration Value descriptions. [ecuc_sws_1028] This one description (ECU Configuration Value description) collects the complete configuration of BSW modules in a single ECU. Each module generator may then extract the subset of configuration data it needs from that single format.

3.1.3 Configuration Classes

The development of BSW modules involves the following development cycles: compiling, linking, and downloading the executable to ECU memory. Configuration of parameters can be done in any of these process steps: pre-compile time, link time, or even post-build time. According to the process step that does the configuration of parameters, the configuration classes are categorised below:

- pre-compile time.
- link time.
- post-build time.

The configuration in different process steps has some consequences for the handling of ECU configuration parameters. If a configuration parameter is defined as precompile time, after compilation this configuration parameter can not be changed anymore. Or if a configuration parameter is defined at post-build time the configuration parameter has to be stored at a known memory location. Also, the format in which the BSW module is

delivered determines in what way parameters are changeable. A source code delivery or an object code delivery of a BSW module has different degrees of freedom regarding the configuration. The configuration class of a parameter is typically not fixed in the standardised parameter definition since several variants are possible. However, once the module is implemented the configuration class for each of the parameters is fixed in that implementation. Choosing the right configuration class from the available variants is depending on the type of application and the design decisions taken by the module implementer. Different configuration classes can be combined within one module. For example, for post-build time configurable BSW implementations only a subset of the parameters might be configurable post-build time. Some parameters might be configured as precompile time or link time. File formats used for describing the configuration classes:

- Arxml (An xml file standardised by AUTOSAR).
- .exe (An executable that can be downloaded to an ECU).
- .hex (A binary file that can be downloaded to an ECU , but it can not execute on its own).
- .c (A C-source file containing either source code or configuration data).
- .h (A header file for either source code or configuration data).
- .obj (An object file for either source code or configuration data).

3.1.4 Edit ECU Configuration

After getting data and presenting it in GUI the second step in the process of ECU configuration is to edit the configuration parameters for all BSW modules. [ecuc_sws_1030] Once the section for a specific BSW module has been generated in the base ECU Configuration Value description, it can be edited with AUTOSAR ECU Configuration Editors. Those editors may operate with user interaction, semi-automatically or automatically, depending on the BSW module and implementation. A straightforward approach to editing the ECU Configuration Value description.

Editing the ECU Configuration is a process that has some aspects which put specific requirements on tools and work procedures. One aspect is the iterative process when editing ECU configuration parameters and another aspect is support for configuration management.

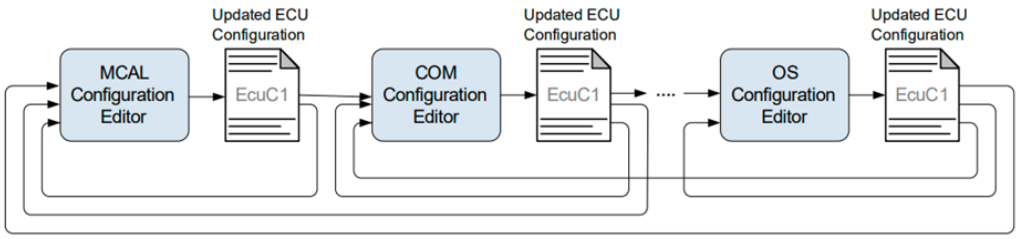


Figure 8: the process of updating an ECU configuration.

The previous figure shows how a set of the tools might be used in a chain with iteration in order to achieve a successful ECU Configuration. Tools are sequentially called within a toolchain to create an ECU Configuration Value description. Iteration cycles must be implemented by repeated activation of different

configuration tools for specific aspects of the BSW. Dependencies between tools, as well as the configuration workflow, might need to be expressed explicitly. Configuration tools are required only to support a single standardised interface, the ECU Configuration Value Template. Tools supporting the methodology and the iterations needed for ECU configuration can be designed based on different strategies. gives some information about this topic.

3.1.5 Configuration Metamodel.

The metamodel for the configuration of ECU artefacts uses a universal description language so that it is possible to specify different kinds of configuration aspects. This is important as it is possible to describe AUTOSAR-standardised and vendor-specific ECU Configuration Parameters with the same set of language elements. This eases the development of tools and introduces the possibility to standardise vendor-specific ECU Configuration Parameters at a later point in time. In general, the configuration language uses containers and actual parameters. Containers are used to group corresponding parameters. Parameters hold the relevant values that configure the specific parts of an ECU. Due to the flexibility that has to be achieved by the configuration language the configuration description is divided into two parts:

- ECU Configuration Parameter Definition.
- ECU Configuration Values.

3.1.6 ECU Configuration Template Structure

The goal of the ECU Configuration Value Template is to specify an exchange format for the ECU Configuration Values of one ECU. The actual output of ECU Configuration editors is stored in the ECU Configuration Value description, which might be one or several XML files. But the ECU Configuration editors need to know how the content of an ECU Configuration Values should be structured (which parameters are available in which container) and what kind of restrictions are to be respected (e.g. the ECU Configuration Parameter is an integer value in the range between 0 and 255). This is specified in the ECU Configuration Parameter Definition which is also an XML file. The relationship between the two file types is shown in figure.

For the ECU Configuration editors there are basically two possible approaches to implement these definitions. Either the ECU Configuration Parameter Definition is read and interpreted directly from the XML file or the defined structures are hard-coded into the tool2. For the development of the ECU Configuration Parameter Definition and the ECU Configuration Value description a model-based approach has been chosen which already has been used during the development of other AUTOSAR template formats.

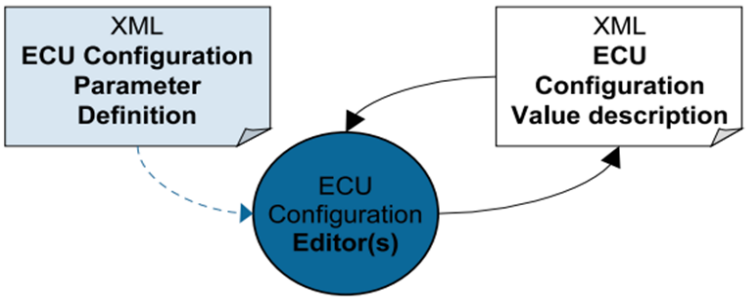


Figure 9:ECU configuration template structure.

3.1.7 ECU Configuration Parameter Definition

The two major building blocks for the specification of ECU Configuration Parameter Definitions are containers and parameters/references. With the ability to establish relationships between containers and parameters and the means to specify references, the definition of parameters has enough power for the needs of the ECU Configuration.

3.1.7.1 Container Definition

The container definition is used to group other parameter container definitions, parameter definitions and reference definitions. There are two specialisations of a container definition. The abstract class **Ecu Container Def** is used to gather the common features (see figure).

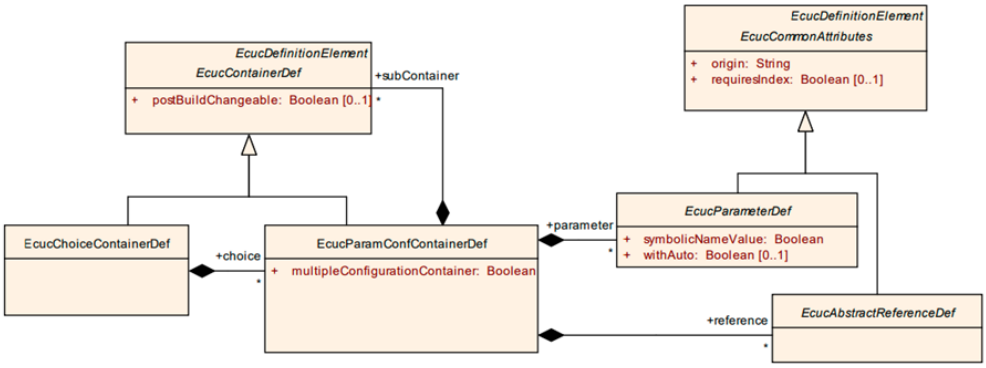


Figure 10 :an abstract class diagram represents the container definition.

3.1.8 Object Streaming ARXML Creation:

In this part we will discuss the main core of our project the ARXML file creation, in this part, we use the object streaming concept that is done by using JAVA programming language with XBlind library, **object streaming** the following figure shows the change from object to Xml or from ARXML to object.

We use JAXB :

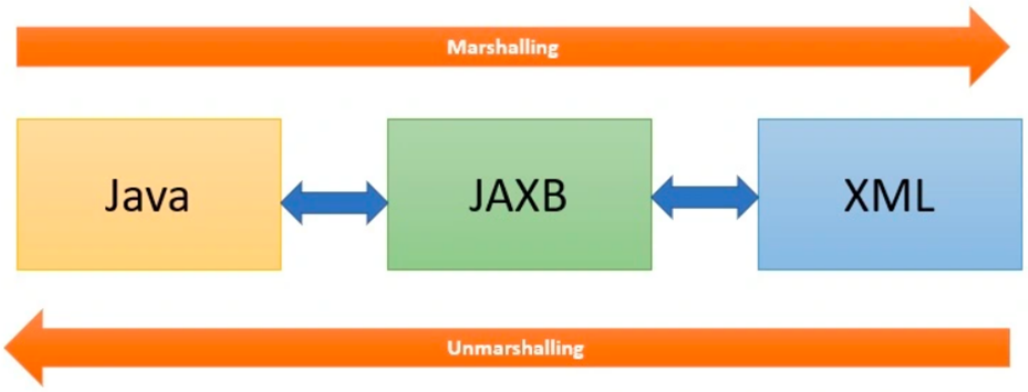


Figure 11: java architecture for xml binding.

JAXB (Java Architecture for XML Binding) is a Java technology that allows for seamless conversion between Java objects and XML representations. It provides a set of APIs and concepts that enable efficient and standardised **marshalling** (Java Object to XML) and unmarshaling (XML to Java Object) operations. In the context of creating ARXML files, JAXB can be used to facilitate the generation of ARXML from Java objects and vice versa.

Here are some key points to discuss in your report about using JAXB for ARXML file creation:

1. Purpose of JAXB: Explain that JAXB is specifically designed to simplify the process of working with XML data in Java applications. It provides a convenient way to map XML schema definitions (XSD) to Java classes, allowing for easy conversion between XML and Java object representations.
2. **Marshalling** (Java Object to XML): Describe how JAXB enables the conversion of Java objects to ARXML. When marshalling, JAXB serialises Java objects into ARXML format based on the defined XML schema. It handles the traversal of the object graph and converts object properties into corresponding XML elements and attributes.
3. Unmarshaling (XML to Java Object): Discuss how JAXB facilitates the conversion of ARXML to Java objects. During unmarshalling, JAXB parses the ARXML input based on the XML schema and creates corresponding Java objects. It populates the object properties with data from the XML elements and attributes.
4. XML Schema Definition (XSD): Highlight the importance of having a well-defined XML schema for ARXML files. Explain that JAXB requires an XSD to generate the Java classes that represent the XML structure. The XSD serves as a contract that defines the structure and constraints of the ARXML data.
5. JAXB Annotations: Explore the use of JAXB annotations in Java classes to customise the XML representation. JAXB annotations allow fine-grained control over the mapping between Java objects and ARXML elements/attributes.

They can be used to define element names, namespaces, data types, and handle other XML-specific requirements.

6. Code Generation: Discuss the code generation process in JAXB. Explain that in JAXB we make Java classes from an XSD using build plugins in popular build tools like Maven or Gradle. These generated classes serve as the foundation for marshalling and unmarshalling ARXML data.
7. Data Binding and Validation: Explain how JAXB performs data binding between XML and Java objects. JAXB provides mechanisms for handling complex data types, collections, inheritance, and more. It also supports validation of the XML data against the defined XML schema to ensure data integrity and compliance.
8. Integration with ARXML-Based Projects: Emphasise that JAXB can seamlessly integrate with ARXML-based projects. Developers can use JAXB to create and manipulate ARXML files programmatically, allowing for efficient configuration management and interaction with other AUTOSAR tools and frameworks.
9. Benefits of Using JAXB for ARXML: Summarise the benefits of utilising JAXB for ARXML file creation. These include reduced development effort by leveraging automated code generation, improved maintainability through strongly-typed Java objects, and simplified data interchange between Java applications and ARXML-based systems.

By discussing these points, you can provide a comprehensive overview of how JAXB enables the creation of ARXML files by

seamlessly converting Java objects to XML and vice versa. Highlighting the advantages and integration capabilities of JAXB will emphasise its value in AUTOSAR-based projects.

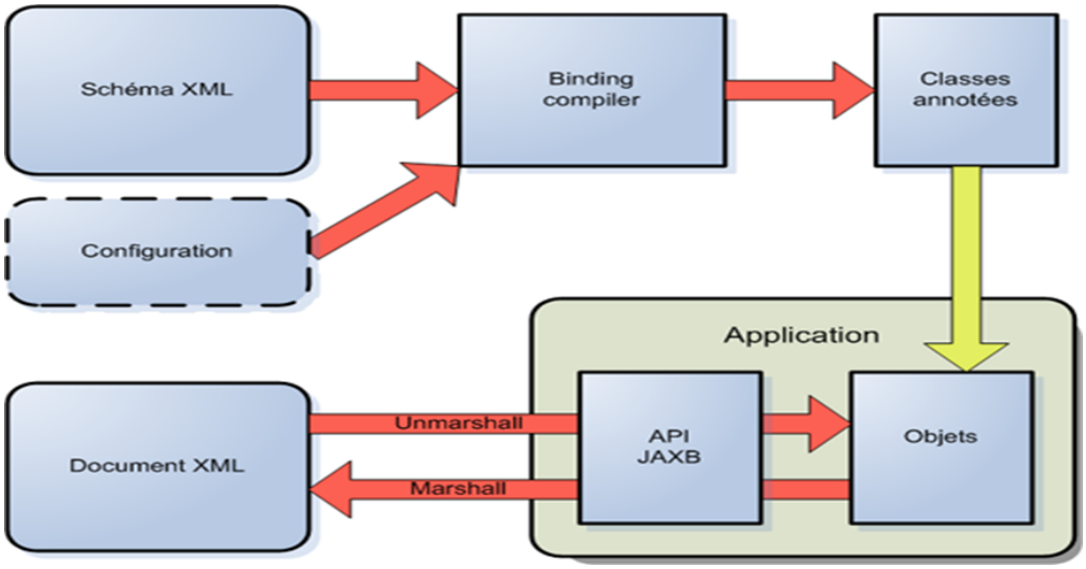


Figure 12: how JAXB works.

3.1.9 ARXML Files Merging

In this section, we will discuss the implementation of the ARXML file merging functionality in the graduation project. The purpose of this functionality is to combine two ARXML files into a single merged file. This operation is performed using the “ARXMLMerger”class, which encapsulates the merging logic.

The ARXML file merging functionality is an essential part of Our configuration tool. It enables the combination of two ARXML files, preserving their structure and merging the relevant elements and parameters.

Execution:

The ARXML file merging functionality can be executed by calling the `mergeARXMLFiles(existingFilePath, newFilePath, mergedFilePath)` method, providing the file paths of the existing and new ARXML files, as well as the desired output file path

Error Handling:

Proper error handling is implemented to handle exceptions that may occur during the merging process. If an error occurs, an error message will be displayed indicating the cause of the failure.

By implementing the ARXML file merging functionality, our system enables the seamless integration of two ARXML files, providing a consolidated representation of their contents and a simple GUI.

The MergeGui class represents the graphical user interface (GUI) for the ARXML merger tool. It provides a simple and intuitive way for users to select existing ARXML files, Second ARXML files, specify the merged file path, and initiate the merging process.

The class defines event listeners for each button to handle user interactions. When the "Select Existing File" button is clicked, a file chooser dialog is opened, allowing the user to select the existing ARXML file. The selected file path is then displayed in the label: First FilePath.

When the "Select Second File" button is clicked, the user can select the new ARXML file, and its path is displayed in the lblSecondFilePath label.

The "Save Merged File" button opens a file chooser dialog where the user can specify the path to save the merged file. The selected path is displayed in the lblMergedFilePath label.

Finally, when the "Merge" button is clicked, the ARXMLMerger.mergeARXMLFiles() method is called to merge the selected existing and new ARXML files. If the merging process is successful, a message dialog is displayed indicating the successful merge. If any errors occur during the merging process, an error message dialog is displayed.

Overall, the MergeGui class provides a user-friendly interface for selecting ARXML files and merging them, enhancing the user experience and simplifying the merging process.

3.1.10 ARXML SPLITTER

Or we can define it as The “export_part_of_file_by_tagname_and_short_name” method allowing you to export a specific part of an ARXML file based on a given tag name and short name value.

Here's how the code works:

1. It starts by setting up the necessary document builders and parsers using the “DocumentBuilderFactory” and “DocumentBuilder” classes.

2. The method parses the input ARXML file using “db.parse(new File(fileName))”, where “fileName” represents the path to the input file.
3. It compiles an XPath expression to select the matching nodes based on the provided tag name and short name value.
4. The “XPathExpression” is evaluated using “expr.evaluate(doc, XPathConstants.NODESET)”, where “doc” represents the parsed ARXML document. This returns a “NodeList” containing the matching nodes.
5. A new “Document” object is created to hold the extracted data.
6. A root node, “<CONTAINERS>”, is created and appended to the new document.
7. Each matching node from the original document is imported into the new document using “newDoc.importNode(node, true)”.
8. The imported nodes are appended as child nodes to the root node in the new document.
9. The new document is written to an XML file using “Transformer” and “StreamResult” objects. The output file path is provided as “outputFileName”.
10. The transformer is configured to format the output XML with proper indentation.
11. Finally, the `transformer.transform(source, result)` call writes the new document to the output file.

By using this method, you can extract a specific subset of the ARXML file based on the provided tag name and short name

value. This allows you to isolate and export relevant data from the ARXML file for further analysis or processing.

3.2 Generator

3.2.1 introduction

The generator is the tool that automates the process of generating C code and header files for Autosar-based software systems. It takes as input the Autosar compliant xml files which contains the user's configuration of a module, parse this file to get the required parameters' values, apply consistency checks to make sure that parameters dependency is correct and finally generate the code.

Before we start to talk about the generator, a small reminder that the module that we're trying to implement through our project is the **ADC module**.

Figure 13: simplified illustration of the generator input and outputs.

disclaimer: what we mean by **config** is the user configuration of a pacific module, which in our case will be the ADC module.

but the question here is, what does config.h and config.c files contain?

the answer can be simply shown in the following figures:

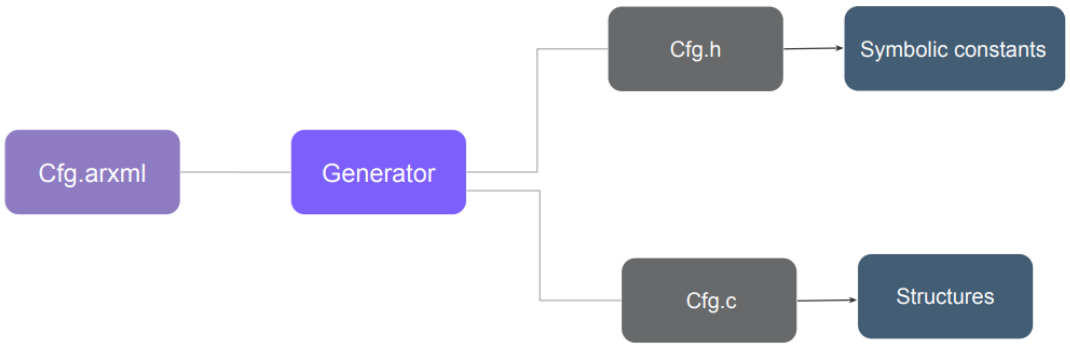


Figure 14: what a config.c and config.h files would contain.

The configuration header file would simply contain a set of hash of defines of the parameters' names and their corresponding values, where the configuration source file would contain a set of structures for a set of specific sub containers that can have multiple configurations such as: Adc channel, Adc group,etc.

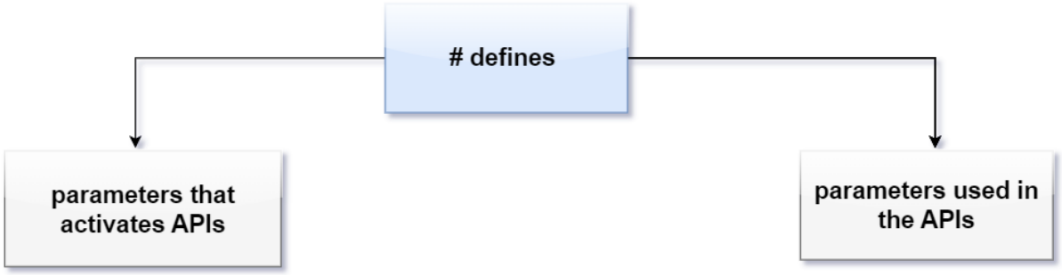


Figure 15: the types of symbolic constants exist in the generated header file.

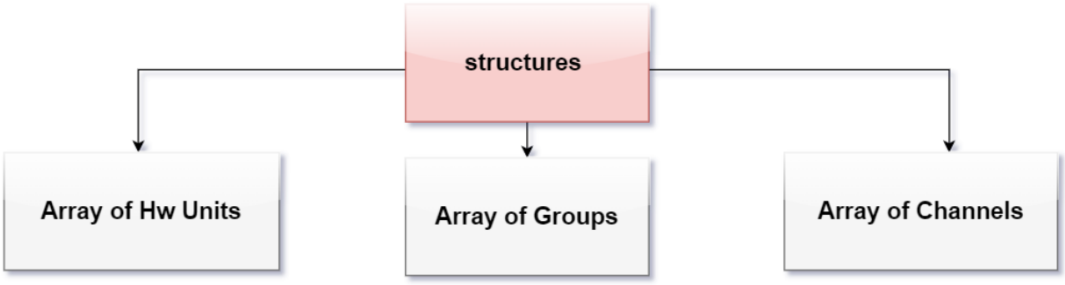


Figure 16: the structures that appear in the generated source file.

but before we dive into how the generator is designed, let take a look on what the ADC driver contains based on the autosar standards:

3.2.2 ADC driver

based on the autosar specification, the following figure shows the files included in an ADC driver:

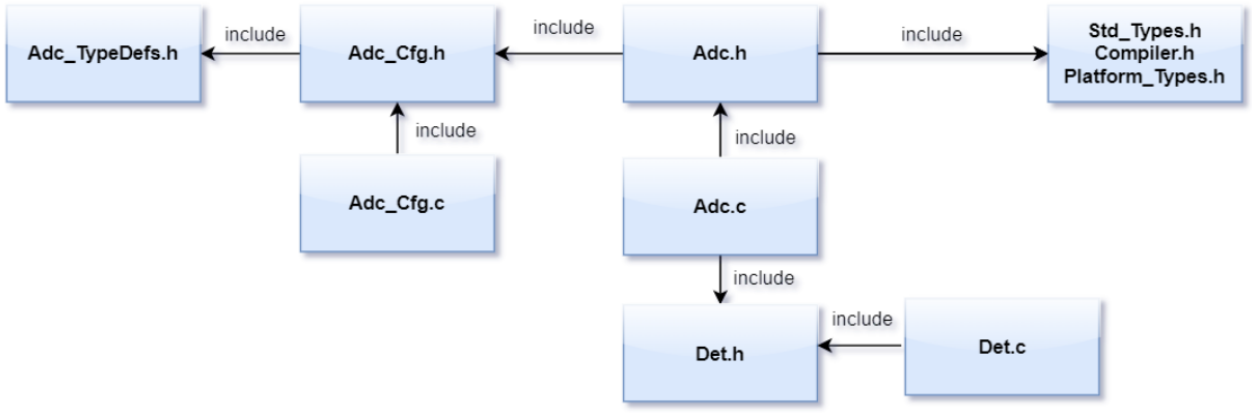


Figure 17: ADC driver file include structure.

we have three types of files in the Adc driver:

1. **generated files:**

   typically the following files can be generated in an ADC driver:

   - **Adc_Lcfg.h, Adc_Lcfg.c**: contains link time parameters (constants, extern variables)
   - **Adc_Cfg.h, Adc_Cfg.c**: contain precompile parameters.
   - **Adc_PBfg.h, Adc_PBcfg.h:** contain post build parameters.

2. **files that come from the compiler's supplier:**
    - **Std_Types.h:** this file includes symbolic constants for the vendor id, module id, autosar version, ..etc.
    - **Compiler.h:** this file contains the configuration of the target 's memory and pointers and so on.
    - **Platform_Types.h:** this file contains definitions such as Autosar integer data types, cpu types and bit/ bytes order.
3. **files that comes from other modules:**
    - **Det.h, Det.c:** these files handle the Adc functions' errors, such as calling a function before initialising the Adc, the .h file contains macros that handle these errors and the .c file contains the functions that use these macros based on the reported error.
4. **others:**
    - **Adc_TypeDefs.h:** this header contains the type definitions, structs and enumerations.
    - **Adc.h:** contains the symbolic constants required by the Adc.c file.
    - **Adc.c:** contains the functions needed to make Adc work.

disclaimer: the current version of the tool only supports the generation of Lcfg.h, Lcfg.c files.

3.2.3 generator phases

the generator phases is shown in the following figure:

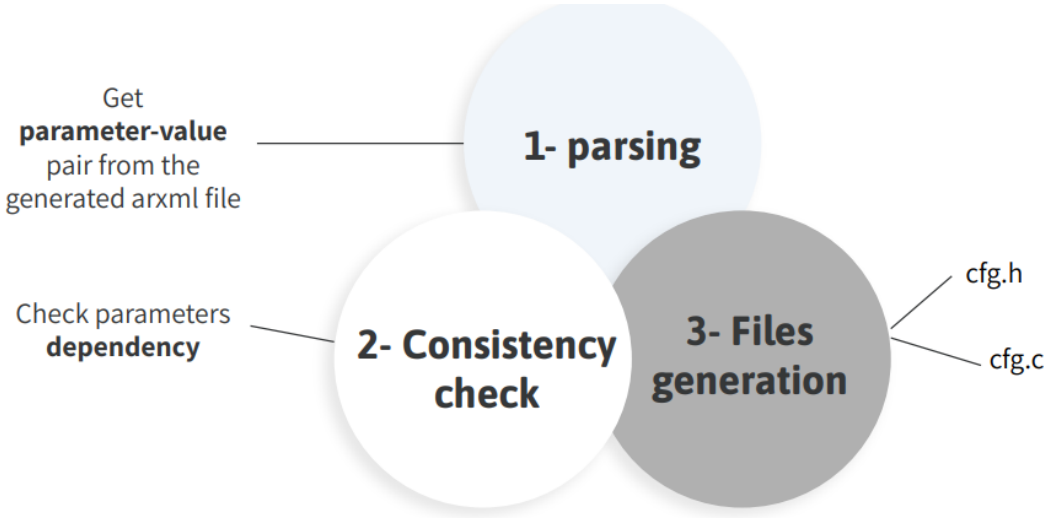


Figure 18: the generator phases

3.2.3.1 Parser

The parser used in this phase is different from the MD parser, as it's used to get the parameter's values of the user's configuration file.

for example, in Adc module, we have three main containers, each container may have parameters and may have subcontainers, the parser role here is to get these parameters and their corresponding values out of these main containers and save them in a data structure in the form of key (that represents the parameter's name) and a value (represents the parameter's value).

So we have three main java classes:

- Parameters. Class: this class has a hash map that is used to store the parameters' values in the form of key and value.
- Containers. Class: this class inherits the Parameters class.
- Subcontainers. Class: this class inherits the Containers class, so each object in this class has a parent container name, the parent parameters and finally its own name and parameters.

Disclaimer: for more information about the containers, subcontainers and parameters of the Adc, you find them in the following autosar documentation:
Sws_currentVersion.pdf

Before we get to know the steps of parsing the configuration file, we can have a look on the structure of the xml file:

```
<CONTAINERS>
    <ECUC-CONTAINER-VALUE>
        <SHORT-NAME>AdcGeneral</SHORT-NAME>
        <DEFINITION-REF>AUTOSAR/EcucDefs/Adc/AdcGeneral</DEFINITION-REF>
        <PARAMETER-VALUES>
          <ECUC-BOOLEAN-PARAM-DEF>
            <DEFINITION-REF>AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcHwTriggerApi</DEFINITION-REF>
              <VALUE>False</VALUE>
```

Figure 19: a code snippet of Adc.arxml

<CONTAINERS> is the parent tag or node, it has sub tags or child nodes as following:

<ECUC-CONTAINER-VALUE>: this tag has underneath all the data and parameters related to this container (AdcGeneral)

<SHORT-NAME>: the name of the container.

<DEFINITION-REF>: the path to this container in the current file.

<PARAMETER-VALUES>: this tag has underneath all the parameters related to the current container.

<ECUC-BOOLEAN-PARAM-DEF>: the datatype of the current parameter, which is Boolean here.

<DEFINITION-REF>: the path to this parameter in the current file.

<VALUE>: the value of this parameter.

So the process will be as following:

For each Main Container in the configuration Arxml:

1. Ask for its **<PARAMETER-VALUES>** tag, if it exists, store them in a map of parameters.
2. Ask for its **<REFERENCE-VALUES>** tag, if it exists, store these references in an array list of strings
3. Ask for its **<SUB-CONTAINERS>** tag, if it exists, get the child nodes of the current node, and for each child node repeat starting from step 1 (recursion).
4. When finishing the expansion of the **<SUB-CONTAINERS>** tag, the for loop counter is incremented to get the next Main Container, and steps are repeated again until we finish parsing all the main containers.

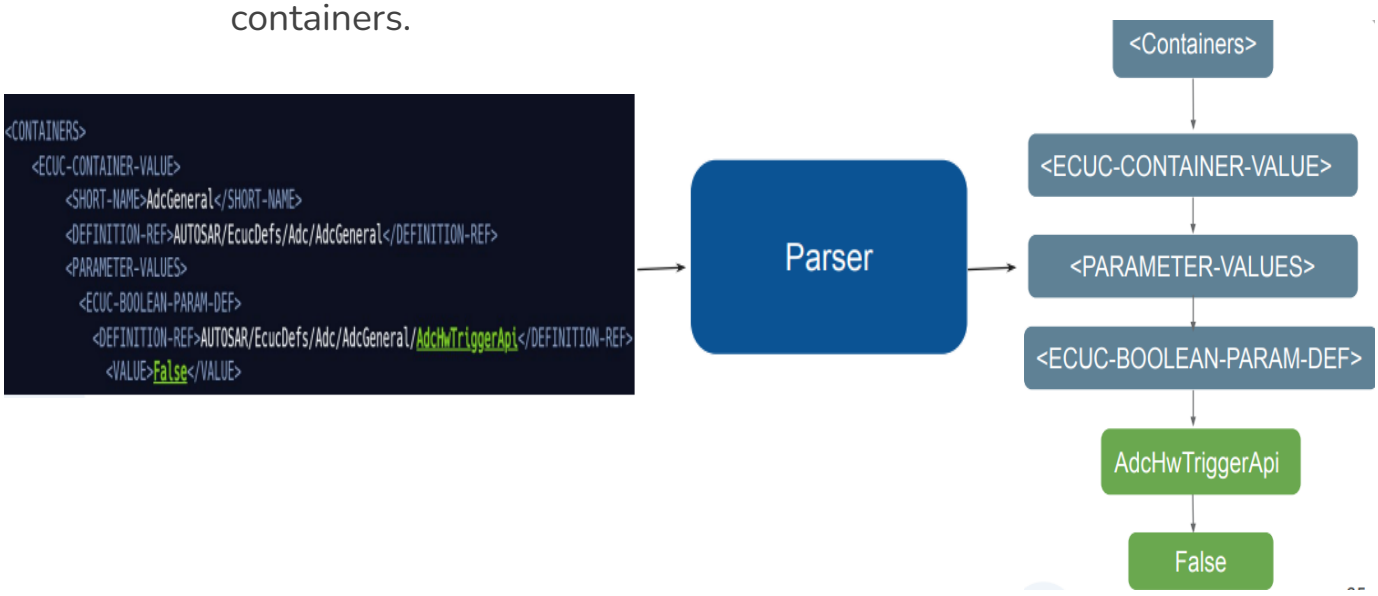


Figure 20:how the parser expands tha Arxml tags to reach a parameter's value.

here's an example of the parser's output:

```
Key: AdcResultAlignment -->  Value: ADC_ALIGN_LEFT
Key: AdcHwTriggerApi -->  Value: False
Key: AdcEnableQueuing -->  Value: False
Key: AdcEnableStartStopGroupApi -->  Value: False
Key: AdcPriorityImplementation -->  Value: ADC_PRIORITY_NONE
Key: AdcEnableLimitCheck -->  Value: False
Key: AdcGrpNotifCapability -->  Value: False
Key: AdcDevErrorDetect -->  Value: False
Key: AdcDeInitApi -->  Value: False
Key: AdcReadGroupApi -->  Value: False
Key: AdcVersionInfoApi -->  Value: False
##############################################
Key: AdcHwUnitId -->  Value:
##############################################
Key: AdcPowerStateReadyCbkRef -->  Value: 4
Key: AdcPowerState -->  Value: 3
##############################################
Key: AdcGroupAccessMode -->  Value: ADC_ACCESS_MODE_SINGLE
Key: AdcGroupId -->  Value: 2
Key: AdcStreamingNumSamples -->  Value: 1
Key: AdcGroupConversionMode -->  Value: ADC_CONV_MODE_CONTINUOUS
Key: AdcGroupTriggSrc -->  Value: ADC_TRIGG_SRC_HW
Key: AdcStreamingBufferMode -->  Value: ADC_STREAM_BUFFER_CIRCULAR
```

Figure 21 : the configuration's parser output.

3.2.3.2 Consistency check

The ADC (Analog-to-Digital Converter) Driver is an important component of the AUTOSAR Driver interface, as it allows application software to read analog signals from sensors and convert them into digital data that can be processed by the system. The ADC Driver is designed to be highly configurable, allowing it to support a wide range of hardware devices and application requirements. One of the key features of the ADC Driver is its support for parameter dependencies, which allows certain configuration parameters to be linked together in a way that ensures consistency and reduces the risk of errors and inconsistencies.

In this partition, we will explore the parameter dependencies that are present in the Specification of ADC Driver AUTOSAR CP R21-11 Document, with a focus on the parameter dependencies of the AdcGeneral container, the AdcPowerStateConfig Subcontainer, the AdcChannel Subcontainer, and the AdcGeneral Subcontainer.

SWS Item	**ECUC_Adc_00462 :**
Module Name	*Adc*
Module Description	Configuration of the Adc (Analog Digital Conversion) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
AdcConfigSet	1	This container contains the configuration parameters and sub containers of the AUTOSAR Adc module.
AdcGeneral	1	General configuration (parameters) of the ADC Driver software module.
AdcPublishedInformation	1	Additional published parameters not covered by "Common" Published Information. Note that these parameters have "PUBLISHED-INFORMATION" configuration class setting, since they are published information.

Figure 22: the Adc Module and Containers.

Overview of the ADC Driver:

The ADC Driver is a software component that provides an interface between application software and hardware devices that support analog-to-digital conversion. The Driver is responsible for configuring and controlling the hardware device, and for converting the analog input signals into digital data that can be processed by the system.

The ADC Driver is designed to be highly configurable, with a large number of parameters that can be set to control its behaviour and performance. These parameters include things like the sampling rate, the resolution of the digital data, and the conversion mode.

One of the key features of the ADC Driver is its support for parameter dependencies. Parameter dependencies allow certain configuration parameters to be linked together in a way that ensures consistency and reduces the risk of errors and inconsistencies. This is important because the ADC Driver can be configured in many different ways, and it is essential that the configuration is correct and consistent with the requirements of the application software and the hardware device.

AdcGeneral Container

One of the dependencies present in the AdcGeneral container is AdcDeInitApi. This parameter determines whether the Adc_DeInit() service is available. If AdcDeInitApi is set to true, then Adc_DeInit() can be used. However, if it is set to false, then Adc_DeInit() cannot be used. This function is responsible for resetting the ADC module and releasing the resources used by the driver. By disabling this function, the driver cannot reset the ADC module properly, which can lead to issues and cause the driver to malfunction.

Another important dependency present in the AdcGeneral container is AdcEnableQueuing. This parameter determines whether the queuing mechanism is active in case the priority mechanism is disabled. If the priority mechanism is enabled, then the queuing mechanism is always active, and the parameter ADC_ENABLE_QUEUING is not evaluated. If AdcPriorityImplementation equals ADC_PRIORITY_NONE, AdcEnableQueuing is evaluated, and if it is set to true, the queuing mechanism is enabled. This function is responsible for

queuing the conversion requests when the priority mechanism is disabled. By disabling this function, the driver cannot queue the conversion requests properly, which can lead to issues and cause the driver to malfunction.

SWS Item	**ECUC_Adc_00391 :**		
Name	AdcEnableQueuing		
Parent Container	AdcGeneral		
Description	Determines, if the queuing mechanism is active in case of priority mechanism disabled. Note: If priority mechanism is enabled, queuing mechanism is always active and the parameter ADC_ENABLE_QUEUING is not evaluated. true: Enabled. false: Disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: AdcPriorityImplementation: parameter is only evaluated for priority implementation ADC_PRIORITY_NONE.		

Figure 23: the AdcEnableQueuing parameter table.

AdcEnableStartStopGroupApi is another important dependency present in the AdcGeneral container. This parameter determines whether the Adc_StartGroupConversion() and Adc_StopGroupConversion() services can be used. If the parameter is set to true, then these services can be used, but if it is set to false, then these services cannot be used. These functions are responsible for starting and stopping the conversion of a group of channels. By disabling this function, the driver cannot start or stop the conversion of a group of channels properly, which can lead to issues and cause the driver to malfunction.

AdcHwTriggerApi is another important dependency present in the AdcGeneral container. This parameter determines whether the Adc_EnableHardwareTrigger() and Adc_DisableHardwareTrigger() services can be used. If the parameter is set to true, then these services can be used, but if it is set to false, then these services cannot be used. These functions are responsible for enabling or disabling the external hardware trigger. By disabling this function, the driver cannot enable or disable the external hardware trigger properly, which can lead to issues and cause the driver to malfunction.

AdcPowerStateAsynchTransitionMode is another important dependency present in the AdcGeneral container. This parameter determines whether the ADCDriver supports the asynchronous power state transition. If the parameter is set to true, then the support is enabled, but if it is set to false, then the support is disabled. This function is responsible for allowing the ADCDriver to transition to a low-power state asynchronously. By disabling this function, the driver cannot transition to a low-power state asynchronously, which can lead to issues and cause the driver to malfunction. This parameter also has a dependency on AdcLowPowerStatesSupport. This parameter shall only be configured if the parameter AdcLowPowerStatesSupport is set to true.

AdcReadGroupApi is another important dependency present in the AdcGeneral container. This parameter determines whether the Adc_ReadGroup() service is available. If the parameter is set to true, then this service can be used, but if it is set to false, then this service cannot be used. This function is responsible for

reading the converted values of a group of channels. By disabling this function, the driver cannot read the converted values of a group of channels properly, which can lead to issues and cause the driver to malfunction.

AdcVersionInfoApi is another important dependency present in the AdcGeneral container. This parameter determines whether the Adc_GetVersionInfo() service can be used. If the parameter is set to true, then this service can be used, but if it is set to false, then this service cannot be used. This function is responsible for getting the version information of the ADC driver. By disabling this function, the driver cannot get the version information of the ADC driver properly, which can lead to issues and cause the driver to malfunction.

It is important to note that these dependencies are closely related and can affect each other's functionality. For example, if AdcPriorityImplementation is not present or is set to a value other than ADC_PRIORITY_NONE, then AdcEnableQueuing will not be evaluated, and the queuing mechanism will not be activated. Similarly, if AdcLowPowerStatesSupport is not present or is set to false, then AdcPowerStateAsynchTransitionMode will not be evaluated, and the support for asynchronous power state transition will not be enabled.

AdcPowerStateConfig Subcontainer

The AdcPowerStateConfig subcontainer contains important dependencies that affect the functionality of the ADC driver in relation to different power states supported by the ADC hardware. One of the dependencies present in this subcontainer is AdcLowPowerStatesSupport. This parameter determines whether the ADCDriver supports low-power states. If AdcLowPowerStatesSupport is set to true, then the AdcPowerState parameter can be configured. However, if it is set to false, then the AdcPowerState parameter cannot be configured. This kind of parameter dependency ensures that the AdcPowerState parameter is only configured when it is necessary and that the ADCDriver is not overloaded with unnecessary configurations.

The AdcPowerState parameter is another important dependency present in the AdcPowerStateConfig subcontainer. Each instance of this parameter describes a different power state supported by the ADC hardware. It should be defined by the hardware supplier and used by the ADCDriver to reference specific hardware configurations that set the ADC hardware module in the referenced power state. At least the power mode corresponding to full power state shall be always configured. The type of this parameter is EcucIntegerParamDef, and the range is from 0 to 18446744073709551615. The default value is not applicable, and the value configuration class is pre-compile time for all variants. This parameter also has a scope dependency, which means that it shall only be configured if the AdcLowPowerStatesSupport parameter is set to true.

Another important dependency present in the AdcPowerStateConfig subcontainer is **AdcPowerStateReadyCbkRef**. Each instance of this parameter contains a reference to a power mode callback defined in a CDD or IoHwAbs component. The type of this parameter is **Ecu Function Name Def**, and the multiplicity is 1. The post-build variant value is false, and the value configuration class is pre-compile time for all variants. This parameter also has a scope dependency, which means that it shall only be configured if the AdcLowPowerStatesSupport parameter is set to true. This function is responsible for notifying the ADCDriver when the ADC hardware has successfully transitioned to the referenced power state. By disabling this function, the driver cannot receive the notification properly, which can lead to issues and cause the driver to malfunction.

SWS Item	**ECUC_Adc_00460 :**		
Name	AdcPowerStateReadyCbkRef		
Parent Container	AdcPowerStateConfig		
Description	Each instance of this parameter contains a reference to a power mode callback defined in a CDD or IoHwAbs component.		
Multiplicity	1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter shall only be configured if the parameter AdcLowPowerStatesSupport is set to true.		

Figure 24: the AdcPowerStateReadyCbkRef parameter table.

It is important to note that the AdcPowerState parameter and the AdcPowerStateReadyCbkRef parameter are closely related.

The AdcPowerState parameter is used to reference specific hardware configurations that set the ADC hardware module in the referenced power state, and the **AdcPowerStateReadyCbkRef** parameter is used to notify the ADCDriver when the ADC hardware has successfully transitioned to the referenced power state. Therefore, these dependencies should be evaluated and configured together to ensure the correct behaviour of the ADC driver.

The AdcChannel Subconatiner

The AdcChannel subcontainer has several parameter dependencies that must be met for the ADC driver to function correctly. These dependencies include the AdcEnableLimitCheck parameter, the AdcChannelLimitCheck parameter, the AdcChannelLowLimit parameter, the AdcChannelHighLimit parameter, and the AdcChannelRangeSelect parameter. Each of these parameters has a specific dependency on one or more of the other parameters. If any of these dependencies are not met, then a dependency error occurs. Therefore, it is essential to ensure that all the dependencies are met when configuring the ADC driver to ensure its proper functioning

The AdcChannelResolution parameter in the AdcChannel subcontainer has a dependency on the AdcMaxChannelResolution parameter in the same container. The AdcChannelResolution parameter specifies the channel resolution in bits and has a range of 1 to 63. It has a multiplicity of 0..1, meaning that it is optional and can have zero or one value. The ImplementationType of this parameter is Adc_ResolutionType.

The AdcMaxChannelResolution parameter is a global parameter that specifies the maximum resolution of the ADC. This parameter has a dependency scope of local, which means that it is only available within the AdcChannel subcontainer. The AdcChannelResolution parameter depends on the AdcMaxChannelResolution parameter, which means that the value of AdcChannelResolution must be less than or equal to the value of AdcMaxChannelResolution. If this dependency is not met, then it is not the right value, and a dependency error occurs.

Next, the AdcChannelLimitCheck parameter in the AdcChannel subcontainer depends on the AdcEnableLimitCheck parameter in the same container. The AdcChannelLimitCheck parameter enables or disables limit checking for an ADC channel. It has a multiplicity of 0..1, meaning that it is optional and can have zero or one value. The ImplementationType of this parameter is EcucBooleanParamDef.

SWS Item	**ECUC_Adc_00453 :**		
Name	AdcChannelLimitCheck		
Parent Container	AdcChannel		
Description	Enables or disables limit checking for an ADC channel.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: AdcEnableLimitCheck: not available if limit checking is not globaly enabled. AdcGroupDefinition: ADC channels with limit checking feature enabled have to be assigned to ADC groups which consist exactly of one limit checking enabled ADC channel.		

Figure 25: the AdcEnableLimitCheck parameter table.

The AdcEnableLimitCheck parameter is a global parameter that enables or disables limit checking for all ADC channels. This parameter has a dependency scope of local, which means that it is only available within the AdcChannel subcontainer. The AdcChannelLimitCheck parameter depends on the AdcEnableLimitCheck parameter, which means that the AdcChannelLimitCheck parameter is not available if the AdcEnableLimitCheck parameter is not globally enabled. If this dependency is not met, then a dependency error occurs.

The AdcChannelLowLimit parameter in the AdcChannel subcontainer also depends on the AdcEnableLimitCheck and AdcChannelLimitCheck parameters. The AdcChannelLowLimit parameter specifies the low limit used for limit checking and has a range of 0 to 18446744073709551615. Like the other parameters, it has a multiplicity of 0..1, meaning that it is optional and can have zero or one value. The ImplementationType of this parameter is EcucIntegerParamDef.

The AdcChannelLowLimit parameter depends on the AdcEnableLimitCheck and AdcChannelLimitCheck parameters. The AdcEnableLimitCheck parameter enables or disables limit checking for all ADC channels, while the AdcChannelLimitCheck parameter enables or disables limit checking for a specific ADC channel. The AdcChannelLowLimit parameter is not available if either the AdcEnableLimitCheck or AdcChannelLimitCheck parameters are not enabled. Additionally, the value of the AdcChannelLowLimit parameter must be greater than or equal

to the AdcChannelHighLimit parameter. If these dependencies are not met, then a dependency error occurs.

Similarly, the AdcChannelHighLimit parameter in the AdcChannel subcontainer depends on the AdcEnableLimitCheck, AdcChannelLimitCheck, and AdcChannelLowLimit parameters. The AdcChannelHighLimit parameter specifies the high limit used for limit checking and has a range of 0 to 18446744073709551615. It also has a multiplicity of 0..1, meaning that it is optional and can have zero or one value. The ImplementationType of this parameter is EcucIntegerParamDef.

The AdcChannelHighLimit parameter depends on the AdcEnableLimitCheck, AdcChannelLimitCheck, and AdcChannelLowLimit parameters. Like the AdcChannelLowLimit parameter, it is not available if either the AdcEnableLimitCheck or AdcChannelLimitCheck parameters are not enabled. Additionally, the value of the AdcChannelHighLimit parameter must be less than or equal to the AdcChannelLowLimit parameter. If these dependencies are not met, then a dependency error occurs.

Finally, the AdcChannelRangeSelect parameter in the AdcChannel subcontainer depends on the AdcEnableLimitCheck and AdcChannelLimitCheck parameters. The AdcChannelRangeSelect parameter specifies which conversion values are taken into account related to the borders defined with AdcChannelLowLimit and AdcChannelHigh

Limit. It has a multiplicity of 0..1, meaning that it is optional and can have zero or one value. The ImplementationType of this parameter is Adc_ChannelRangeSelectType.

SWS Item	**ECUC_Adc_00456 :**		
Name	AdcChannelRangeSelect		
Parent Container	AdcChannel		
Description	In case of active limit checking: defines which conversion values are taken into account related to the boarders defined with AdcChannelLowLimit and AdcChannelHighLimit. Implementation Type: Adc_ChannelRangeSelectType		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	ADC_RANGE_ALWAYS	Complete range - independent from channel limit settings.	
	ADC_RANGE_BETWEEN	Range between low limit and high limit - high limit value included.	
	ADC_RANGE_NOT_BETWEEN	Range above high limit or below low limit - low limit value included.	
	ADC_RANGE_NOT_OVER_HIGH	Range below high limit - high limit value included.	
	ADC_RANGE_NOT_UNDER_LOW	Range above low limit.	
	ADC_RANGE_OVER_HIGH	Range above high limit.	
	ADC_RANGE_UNDER_LOW	Range below limit - low limit value included.	
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: AdcEnableLimitCheck: not available if limit checking is not globally enabled. AdcChannelLimitCheck: not available if channel specific limit check is not enabled.		

Figure 26: the AdcChannelRangeSelect parameter table.

The AdcChannelRangeSelect parameter depends on the AdcEnableLimitCheck and AdcChannelLimitCheck parameters. It is not available if either the AdcEnableLimitCheck or AdcChannelLimitCheck parameters are not enabled. The AdcChannelRangeSelect parameter has an **EcucEnumerationParamDef** type, which means that it can take

one of several predefined values. The possible values for this parameter are ADC_RANGE_ALWAYS, ADC_RANGE_BETWEEN, ADC_RANGE_NOT_BETWEEN, ADC_RANGE_NOT_OVER_HIGH, ADC_RANGE_NOT_UNDER_LOW, ADC_RANGE_OVER_HIGH, and ADC_RANGE_UNDER_LOW. The Adc_ChannelRangeSelectType implementation type specifies the type of the AdcChannelRangeSelect parameter.

AdcGroup Subcontainer

The AdcGroup subcontainer contains several parameters that define the behaviour and characteristics of a group of ADC channels. These parameters include AdcGroupAccessMode, AdcGroupConversionMode, AdcGroupPriority, **AdcHwTrigSignal**, AdcNotification, AdcStreamingBufferMode, and AdcStreamingNumSamples.

AdcGroupAccessMode is a parameter that defines the type of access mode to group conversion results. It can have two values: ADC_ACCESS_MODE_SINGLE and ADC_ACCESS_MODE_STREAMING. ADC_ACCESS_MODE_SINGLE is the single-value access mode, which means that the ADC driver will only provide a single conversion result for the group when requested. ADC_ACCESS_MODE_STREAMING is the streaming access mode, which means that the ADC driver will continuously provide conversion results for the group until the user stops the conversion process. The implementation type for AdcGroupAccessMode is Adc_GroupAccessModeType.

AdcGroupConversionMode is a parameter that defines the type of conversion mode supported by the driver. It can have two values: ADC_CONV_MODE_CONTINUOUS and ADC_CONV_MODE_ONESHOT.
ADC_CONV_MODE_CONTINUOUS is the continuous conversion mode, which means that the conversions for an ADC channel group are performed continuously after a software API call (start). The conversions themselves are running automatically without the need for additional software or hardware triggers.
ADC_CONV_MODE_ONESHOT is the one-shot conversion mode, which means that the conversion of an ADC channel group is performed once after a trigger. The implementation type for AdcGroupConversionMode is Adc_GroupConvModeType.

AdcGroupPriority is a parameter that defines the priority level of the AdcGroup. It is an optional parameter with a range of 0 to 255. The implementation type for AdcGroupPriority is Adc_GroupPriorityType.

AdcHwTrigSignal is a parameter that configures on which edge of the hardware trigger signal the driver should react, i.e. start the conversion (only if supported by the ADC hardware). It can have three values: ADC_HW_TRIG_BOTH_EDGES, ADC_HW_TRIG_FALLING_EDGE, and ADC_HW_TRIG_RISING_EDGE. ADC_HW_TRIG_BOTH_EDGES is used to react on both edges of the hardware trigger signal (only if supported by the ADC hardware).
ADC_HW_TRIG_FALLING_EDGE is used to react on the falling edge of the hardware trigger signal (only if supported by the

ADC hardware). ADC_HW_TRIG_RISING_EDGE is used to react on the rising edge of the hardware trigger signal (only if supported by the ADC hardware). The implementation type for AdcHwTrigSignal is Adc_HwTriggerSignalType.

AdcNotification is a parameter that defines a callback function for each group. It is an optional parameter that allows the user to specify a function that will be called when the conversion for the group is complete. The implementation type for AdcNotification is **EcucFunctionNameDef**.

AdcStreamingBufferMode is a parameter that configures the streaming buffer as either a linear buffer or a ring buffer. It can have two values: ADC_STREAM_BUFFER_CIRCULAR and ADC_STREAM_BUFFER_LINEAR.
ADC_STREAM_BUFFER_CIRCULAR is used to configure the streaming buffer as a ring buffer that wraps around if the end of the stream buffer is reached. ADC_STREAM_BUFFER_LINEAR is used to configure the streaming buffer as a linear buffer that stops the conversion as soon as the stream buffer is full. The implementation type for AdcStreamingBufferMode is Adc_StreamBufferModeType.

AdcStreamingNumSamples is a parameter that defines the number of ADC values to be acquired per channel in streaming access mode. It is an optional parameter that is only valid for streaming access mode with a range of 1 to 255. In single-access mode, this parameter assumes a value of 1, since only one sample per channel is processed. The implementation type for AdcStreamingNumSamples is Adc_StreamNumSampleType.

3.2.3.3 generator

After we've parsed the configuration file, and made sure that parameters' dependency is all checked and no errors with the user's configuration, we want to generate the Lcfg.h and Lcfg.c files.

In order to generate a file that has a static structure but only its content changes with the user's input, we've gone for Velocity Engine.

Velocity is a Java-based templating engine, it's an open source web framework designed to be used as a view component in the MVC architecture and it provides an alternative to some existing technologies such as JSP.

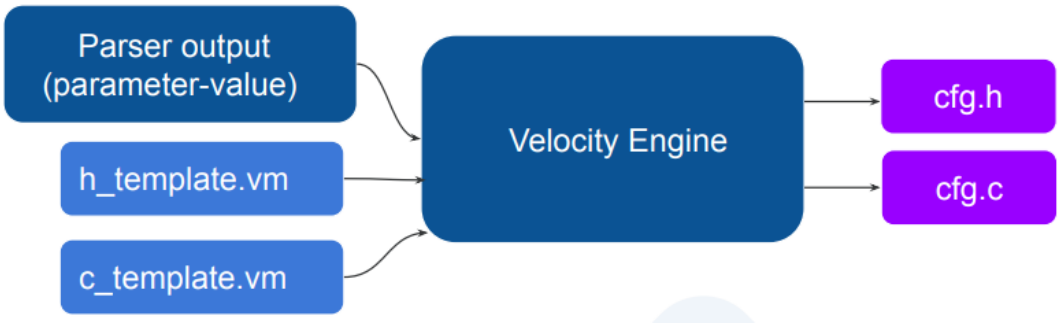


Figure 27: simplified illustration of velocity's inputs and outputs

Here's how things would work:

For the Lcfg.h file generation

1. Initialise the velocity engine.
2. Create a template file (.vm) that contains the static text that will appear in the .h file, which is represented as symbolic constants that holds the names of the parameters.

3. For each parameter in the template, place in front of it a placeholder ($value), these place holders are where we will put the corresponding parsed values at.
4. Get the parsed values from their maps and place them in their corresponding positions in the template file.
5. Put the data in a context object.
6. Merge the template with context data and generate the Lcfg.h file.

For the Lcfg.c file generation

1. Initialise the velocity engine.
2. Create a template file (.vm) that contains the static text that will appear in the .c file, which is represented as set of structures each represents an array of a configured Subcontainer, and each array has a set of values that corresponds to their parameters in the Adc_TypeDefs.h.
3. For each structure's inner content, place a placeholder.
4. For each place holder, Get the parsed values from their maps and write their values in a string that represents the structure content.
5. Put the data in a context object.
6. Merge the template with context data and generate the Lcfg.c file.

The following figure explain more about how things would look like:

In the following example we have the template file on the left and the generated Lcfg.h on the right:

```
\#define ADC_DEINIT_API                          $value0
\#define ADC_DEV_ERROR_DETECT                    $value1
\#define ADC_ENABLE_LIMIT_CHECK                  $value2
\#define ADC_ENABLE_QUEUING                      $value3
\#define ADC_ENABLE_START_STOP_GROUP_API         $value4
\#define ADC_GRP_NOTIF_CAPABILITY                $value5
\#define ADC_HW_TRIGGER_API                      $value6
\#define ADC_LOW_POWER_STATES_SUPPORT            $value7
\#define ADC_POWER_STAT_ASYNCH_TRANSITION_MODE   $value8
\#define ADC_PRIORITY_IMPLEMENTATION             $value9
\#define ADC_READ_GROUP_API                      $value10
\#define ADC_RESULT_ALIGNMENT                    $value11
```

```
#define ADC_DEINIT_API                          False
#define ADC_DEV_ERROR_DETECT                    False
#define ADC_ENABLE_LIMIT_CHECK                  False
#define ADC_ENABLE_QUEUING                      False
#define ADC_ENABLE_START_STOP_GROUP_API         False
#define ADC_GRP_NOTIF_CAPABILITY                False
#define ADC_HW_TRIGGER_API                      False
#define ADC_LOW_POWER_STATES_SUPPORT            False
#define ADC_POWER_STAT_ASYNCH_TRANSITION_MODE   False
#define ADC_PRIORITY_IMPLEMENTATION             ADC_PRIORITY_NONE
#define ADC_READ_GROUP_API                      False
#define ADC_RESULT_ALIGNMENT                    ADC_ALIGN_LEFT
```

In this example we have the template file on the left and the generated Lcfg.c on the right:

```
\#include "Lcfg.h"
\#include "Adc_TypeDefs.h"


Adc_ChannelConfigType Adc_ChannelConfig[ADC_CHANNEL_NUMBER]=
{

    $text0

};



Adc_GroupConfigType Adc_ChannelGroupConfig[ADC_GROUP_NUMBER]=
{

    $text1

};
```

```
#include "Lcfg.h"
#include "Adc_TypeDefs.h"

Adc_ChannelConfigType Adc_ChannelConfig[ADC_CHANNEL_NUMBER]=
{

    {ADC_CHANNEL,} ,
    {ADC_CHANNEL000,},
    {ADC_CHANNEL001,}

};

Adc_GroupConfigType Adc_ChannelGroupConfig[ADC_GROUP_NUMBER]=
{
   {0,
   ADC_CONV_MODE_CONTINUOUS,
   ADC_TRIGG_SRC_HW,
   {ADC_CHANNEL,},} ,

};
```

3.3 Compiler integration

3.3.1 introduction

In order to design an independent tool, we need to integrate 3rd party tools such as compilers.

As we're targeting **AVR atmega32**, we need to provide the required environments and tools, in order to generate. elf and. hex files, or even a .exe to burn this file on a hardware.

We'll just go for generating a. elf and. hex files and use them in a simple project in a simulator such as Porteus.

In order to integrate an **AVR-GCC** compiler, we considered the following requirements:

1. Install WinAVR, and set its binary files' paths in the environment variables.
2. Install cmake.
3. Install MinGW

3.3.2 CMAKE

CMAKE is an open source external tool that has the ability to generate Make files, if we have a makefile, then we can easily use the instructions in this file such as build, clear, run, and compile our source and header files to generate a hex file.

In order to use the CMAKE tool, we considered the following:

1. Create a directory that has a **source folder** which contains all the source files that need to be compiled in our project, including the generated Lcfg.c file.
2. create a **header folder** which contains all the header files that needs to be compiled in our project, including the generated Lcfg.h file.
3. Create a CMakeLists.txt file, which contains the following:
    - A line that specifies the version of the CMAKE tool.
    - A line that tells the compiler that the current CMAKE will act as a cross compiler.
    - A line that forces the compilation to be on avr-gcc compiler not on the normal gcc.
    - A line that will be used later to apply formalisation to the output file (formalise the output from elf to hex)
    - Assign flags to the compiler in order to apply the required optimization.
    - Assign a name to the project
    - Set the source files to a variable to be seen later while the compilation of main.c.
    - Include the header files
    - Generate the .elf file.
    - Formalise the .elf to .hex.
    - And finally add a command to flash the .hex on atmega32.

```
cmake_minimum_required(VERSION 3.26)
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_C_COMPILER avr-gcc CACHE STRING "c compiler" FORCE)
set(CMAKE_OBJCOPY   avr-objcopy CACHE STRING "avr-objcopy" FORCE)
set(CMAKE_C_FLAGS "-mmcu=atmega32 -O1 -DF_CPU=1000000UL")

project(adc C)
set(SRC_FILES main.c
source/LCD.c
source/ADC.c
source/Lcfg.c)
add_executable(${PROJECT_NAME}.elf ${SRC_FILES})
target_include_directories(${PROJECT_NAME}.elf PUBLIC include/)

add_custom_command(TARGET ${PROJECT_NAME}.elf POST_BUILD
    COMMAND ${CMAKE_OBJCOPY} -O ihex ${PROJECT_NAME}.elf ${PROJECT_NAME}.hex
    COMMENT "Creating ${PROJECT_NAME}.hex file...")

add_custom_target(flash
    avrdude -c usbasp -p atmega32 -U flash:w:${PROJECT_NAME}.hex
    DEPENDS ${PROJECT_NAME}.hex
    COMMENT "Flashing ${PROJECT_NAME}.hex to MCU...")
```

Figure 28: an example of CMakeLists.txt file.

4. Then, we will use a java builder object to write in the current directory the CAMKE commands:
    - Mkdir build
    - Cd build
    - Cmake –G “MinGW Makefiles” ..
    - make
5. The elf and hex files are generated.

Here’s our simple project on proteus after the compilation process has finished:

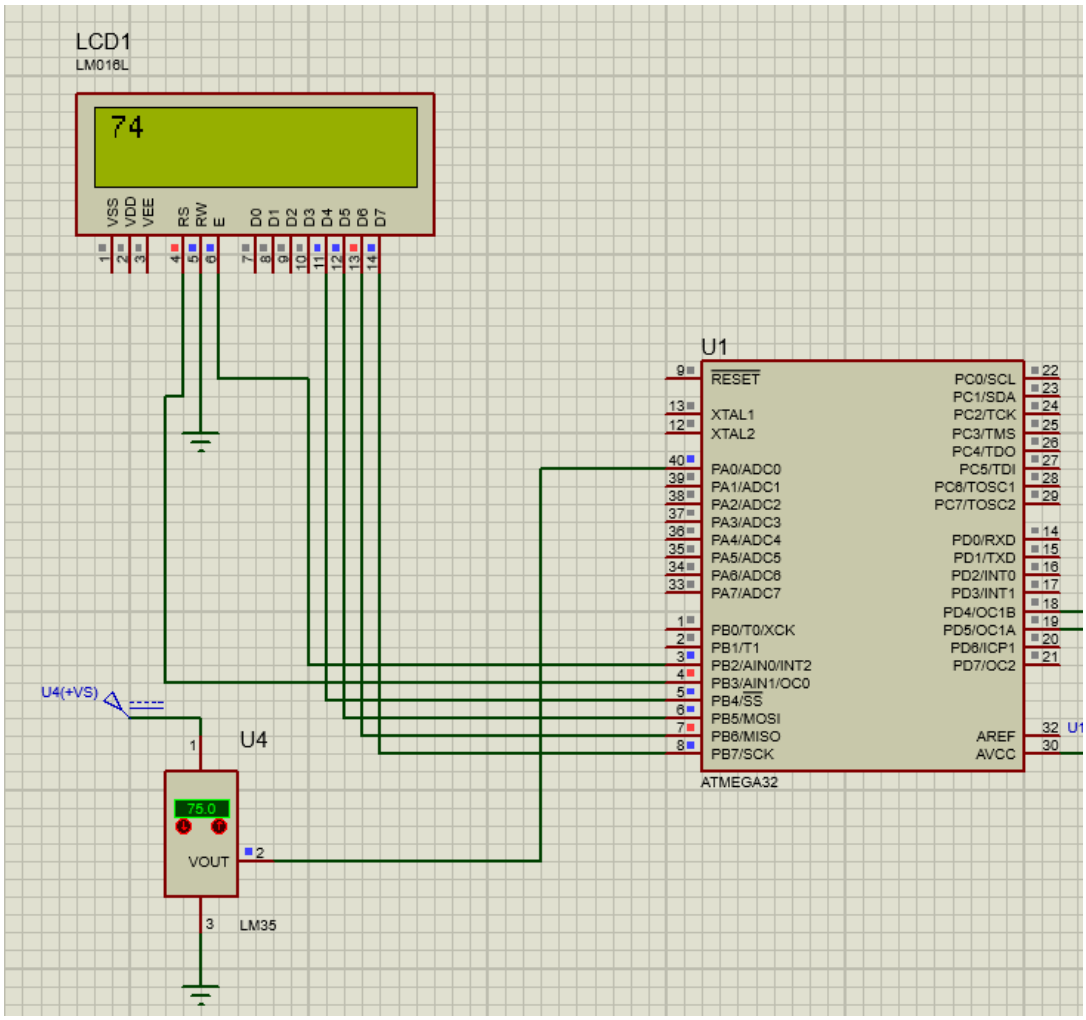


Figure 29: our simple Adc project on proteus.

The project is reading the temperature sensor's analog value, converting this value using ADC to get the digital reading, then showing this value on the LCD in a readable format.

Chapter 4 : AI Module

4.1 Introduction

4.1.1 Our task

Our task is to design an ai interface that can read the user input in various formats and convert it into Arxml files.

This task is complex and challenging as it requires expertise in NLP and machine learning. The goal of this task is to create a user-friendly interface that allows users to input information in natural language, regardless of the format, and have that information transformed into an ARXML file.

To accomplish this task, it's required to train a LM on a large corpus of ARXML data to develop the understanding of the XML pattern, syntax and also provide the model with more examples of user configuration.

our very first solutions to this task were as following:

4.1.2 approaches to solve the problem

- **sentence similarity.**

  Sentence similarity is a technique used to compare two sentences and determine how similar they are. To apply sentence similarity to our task, we first need to define a set of predefined examples that represent the different input formats and their corresponding ARXML.

  Once the examples are defined, we can use a similarity metric, such as **cosine similarity** or **jaccard similarity**, to compare the user input to the predefined examples. The example with the

highest similarity score will be selected as the best match for the input.

for example, suppose the user input is **“create an Adc module with Adc deinit enabled and has one channel and two Adc groups”**, we can compare this input to set of examples including:

“create an Adc module with Adc deinit enabled and has one channel and one Adc group”

“please create Adc that has Adc dev error detect enabled and has one channel and two Adc groups”

“configure an Adc module with Adc deinit enabled and has one channel and two Adc groups”

suppose using cosine similarity, we can calculate the cosine score for each example and then select the example with the highest given score.

Then, we can use NER or regular expressions to extract the required information such as : containers’ names, their parameters (if exists), their subcontainers (if exists) and their corresponding parameters.

The problem with this solution is that it requires a lot of data and coding in order to generate the final Arxml configuration file,hence, the generation time increases. Another problem is that the short names of subcontainers and their corresponding parameters may be hard to recognize or identify, as these subcontainers can have multiple configurations, thus, their

parameters' names can be shown in the user's input multiple times.

- **question and answering.**

  question and answering is an NLP technique that involves answering a question based on a given passage of text. To apply q&a in our task, we need to break down the input data into smaller pieces, such as individual words or phrases , and match these pieces to a set of predefined questions.

  for example, an input like "**create an Adc module with Adc deinit enabled and has one channel and two Adc groups"** will be broken into smaller pieces such as : "**create an Adc module**", "**with Adc deinit enabled**", "**and has one channel**", "**and two Adc groups**", then we match these pieces to a questions like:

  "What is the module name to be configured?"

  "What are the parameters in AdcGeneral to be set?"

  "how many Adc channels are configured?"

  "How many Adc groups are configured?"

  ...

  The problem with this approach is it requires a large amount of high quality training data and this data shouldn't be biassed or incorrect.

- **use Transformers.**

  Transformers, particularly the state-of-the-art models such as GPT-3, can be a powerful solution for this task. This is because transformers are designed to handle sequential data, such as natural language, and can learn powerful representations of the input data that capture complex relationships between the input and output.

  Why is using transformers our preferred solution?

  - It has different types of language models trained on data close to ours.
  - We can fine tune a language model easily on our data for the task of **text to code** generation.
  - a language model such as GPT-3 can understand the syntax and semantics of the xml files.
  - attention mechanism: transformers use the attention mechanism to assign importance to different parts of the input text while the encoding and decoding phases. This allows the model to focus on relevant tokens and understand the relationships between different parts of the instruction, which is valuable when translating natural language to code.
  - Code generation complexity: Generating code from natural language instructions is a challenging task that requires a deep understanding of both programming languages and human language. Transformers, with their ability to capture complex patterns and relationships, are well-suited to handle this complexity

and generate code that aligns with the given textual instructions.

4.1.3 What is NLP?

Natural Language Processing uses both **linguistics** and **mathematics** to connect the languages of humans with the language of computers. Natural language comes in two forms, text or speech. through NLP algorithms, these natural forms of communication are broken down into data that can be understood by a machine.

The study of the official and unofficial rules of language is called **linguistics**. The issue of using formal linguistics to create NLP models is that the rules for any language are complex, especially when we want to convert it into formal mathematical rules. In addition to this, there's another problem which is that human language is full of shortcuts, inconsistencies and errors.

Because of the limitations of formal linguistics, **computational linguistics** has become a growing field. Using large datasets, linguists can discover more about how human language works and use those findings to inform natural language processing. This version of NLP, **statistical NLP,** has come to dominate the field of natural language processing. Using statistics derived from large amounts of data, statistical NLP bridges the gap between how language is supposed to be used and how it is actually used.

disclaimer: statistical NLP is predicting the next word in a sequence based on the previous words that precede it.

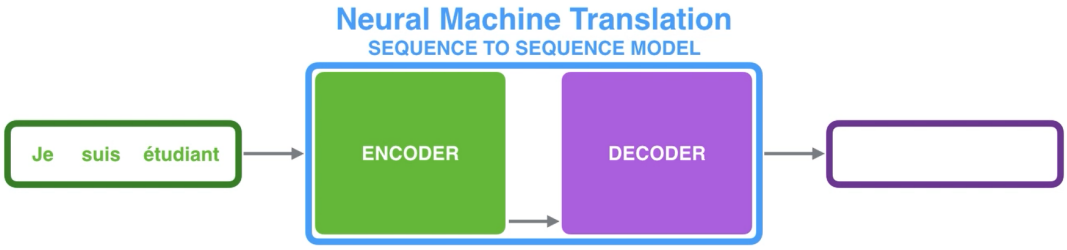
4.1.4 How Transformers work

Transformers are a type of neural network architecture that have been gaining popularity. Transformers were recently used by **OpenAI** in their language models, and also used recently by **DeepMind** for AlphaStar.

Transformers were developed to solve the problem of **sequence transduction** or **neural machine translation** (we mean here any task that transforms an input sequence into output sequence) this includes speech recognition, text to speech transformation.

machine translation

The model is composed of an **encoder** and a **decoder**. the encoder processes each item in the input sequence, it compiles the information it captures into a **vector** (called the **context**). After processing the entire input sequence, the encoder sends the context over the decoder, which begins producing the output sequence item by item.



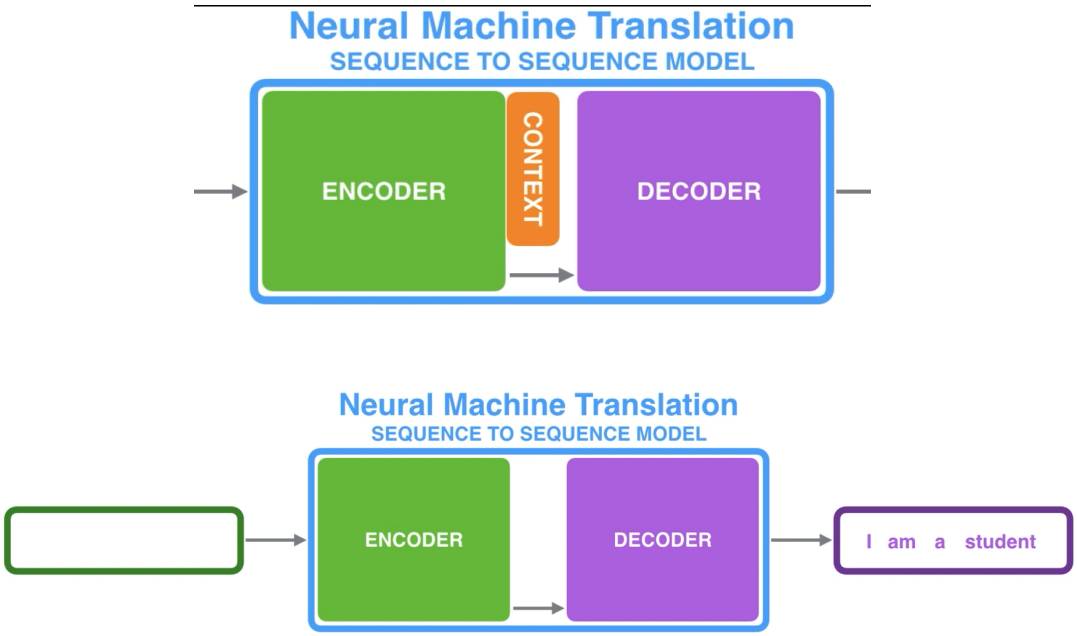


Figure 30: an example of how machine translation works.

For models to perform sequence transduction, it is necessary to have some sort of memory and the model needs to figure out the dependencies and communications between the input sequence. **RNNs** and **CNNs** have been used to deal with this problem because of their properties.

disclaimer: The context is a vector (an array of numbers, basically) in the case of machine translation. The encoder and decoder tend to both be recurrent neural networks.

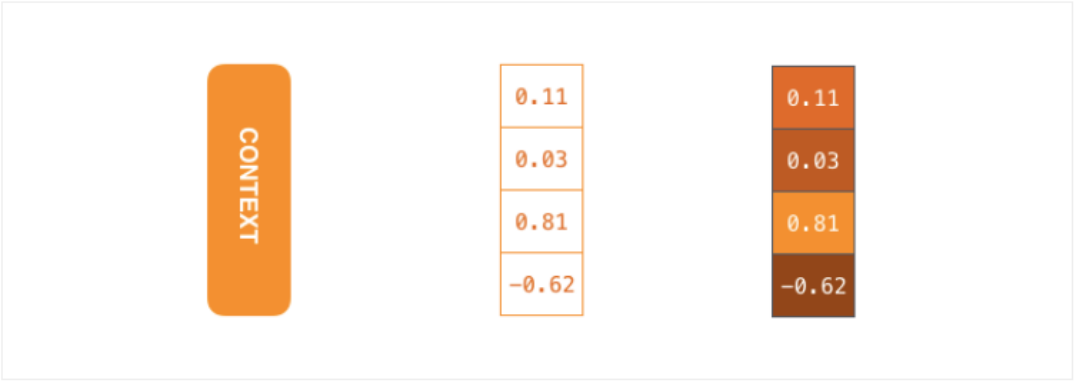


Figure 31: an example of a vector.

By design, a RNN takes two inputs at each step: an input (in the case of encoder, it's one word from the input sentence), and a hidden state. the word needs to be represented by a vector.

To transform a word into a vector, we apply something called "**word embeddings**" algorithms. These turn words into vector spaces that capture a lot of the meaning/ semantic information of the words.

(e.g. king-man+woman=queen).

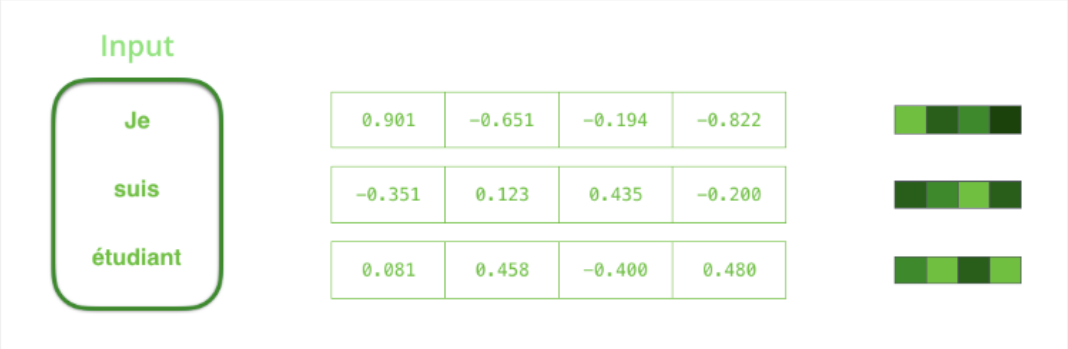


Figure 32: turning the input words into vectors before processing them (word embedding).

Recurrent Neural Networks(RNNs)

Recurrent Neural Networks have loops in them, allowing information to persist.

in the first step, we have the **input vector#1** which represents the word embeddings of a word in the input and a **hidden state#0** , then the network outputs a **hidden state#1** and an **output vector#1** , the next step takes the second input vector and the **hidden state #1** and outputs **output vector#1** and **hidden state #2** and so on.

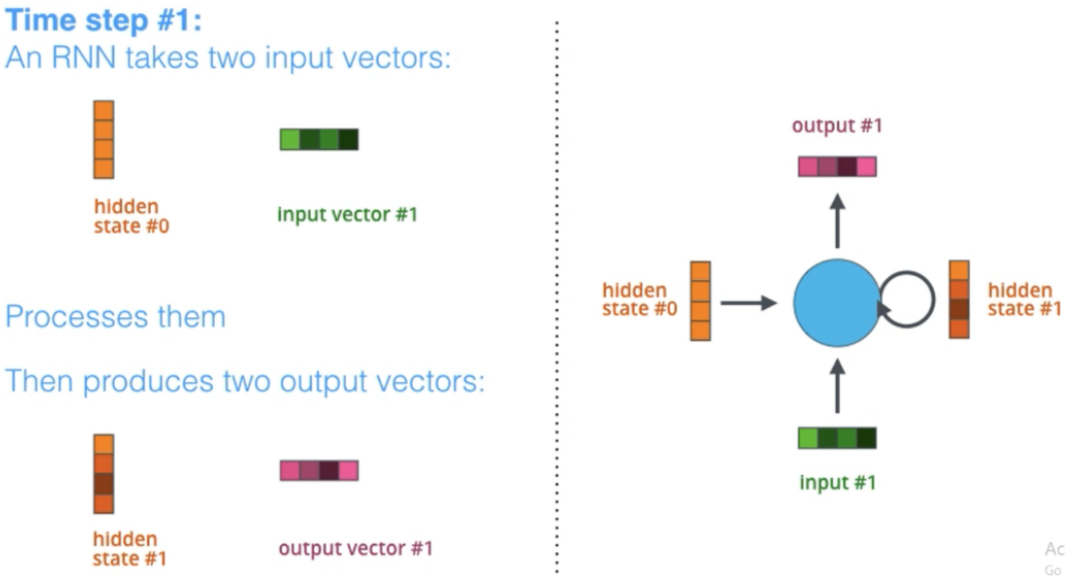


Figure 33: an example of an RNN network.

In the figure below, we can see a part of the RNN network, **A**, processing some input x and outputs h. a loop allows information to be passed from one step to the next.

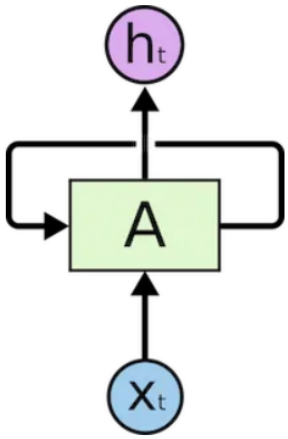


Figure 34: an RNN cell.

an RNN can be thought of as multiple copies of the same network,A, each network passing a message to a successors:

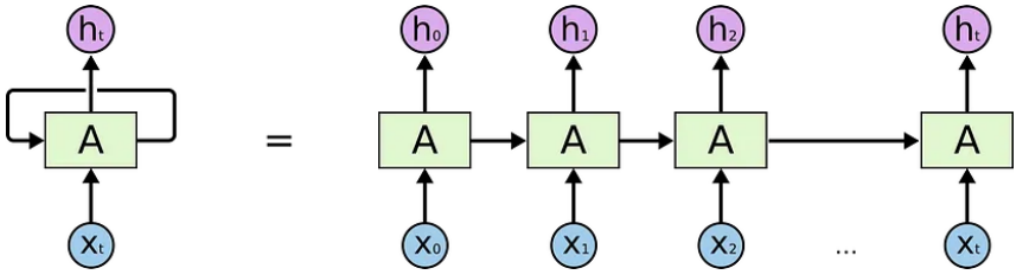


Figure 35: how an RNN network would look like.

If we want to translate some text, we set each input as the word in that text, then the RNN passes the information of the previous words to the next network that can use and process that information.

This is how a sequence to sequence model works using recurrent neural networks, each word is converted to a **vector** then it's given to a network that will output an **output vector** and **hidden state** given to the next network.

But what is the problem here?

Consider having a language model that tries to predict the next word in a sequence, and the sequence is something like this: "the clouds in the sky", it's very simple isn't it! The RNNs can learn to use past information and figure out what is the next word in this sentence.

but there's some cases where we need more context. For example, let's say that you're trying to predict the last word of the text: "I grew up in France … I speak fluently …". Recent information suggests that the last word is a language, but if we

want to know which language is spoken fluently, we need to go back to the context where “France” exists.

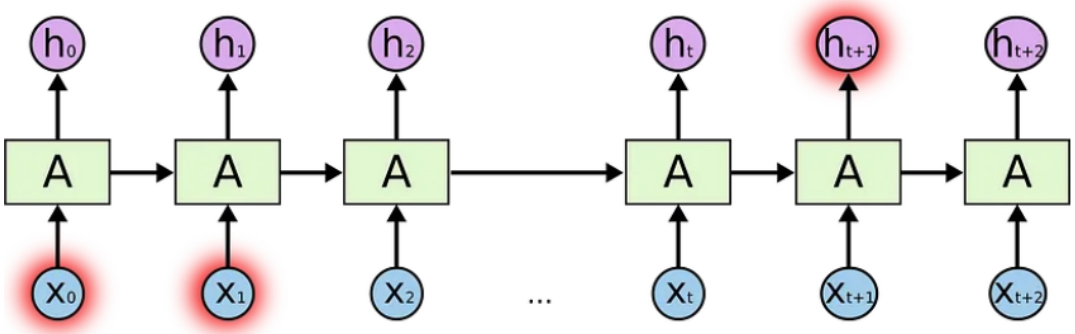


Figure 36: how long the distance between words in a sequence is considered a problem.

RNNs become very ineffective when the gap between the relevant information and the point where it is needed becomes very large. so, to solve this problem, a special type of RNN called “LSTM” tries to solve this problem.

Long-Short Term Memory (LSTM)

LSTM can selectively remember things that are important and forget things that aren’t so important. internally, a LSTM looks like the following:

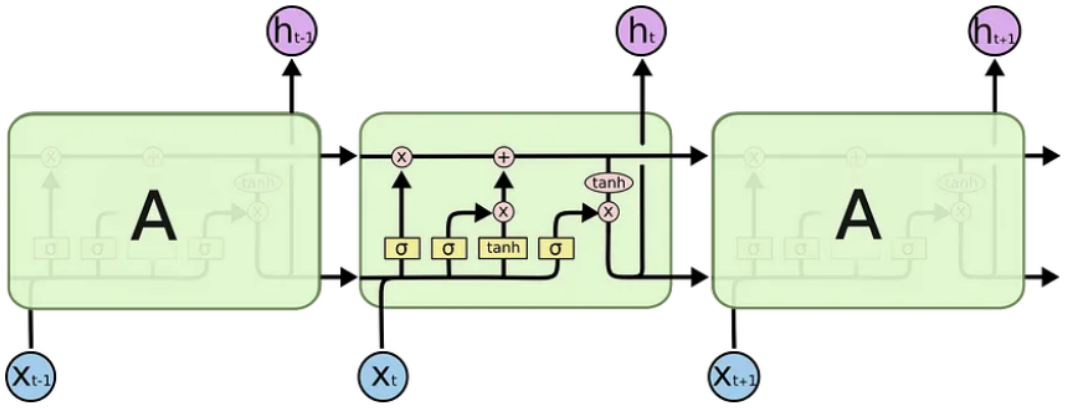


Figure 37: an example of the LSTM network.

Each cell takes as inputs **x_t** (a word in the case of a sentence to sentence translation), the **previous cell state** and the **output of the previous cell**. It manipulates these inputs and based on them, it generates a new cell state, and an output.

the problem here:

Actually, the same problem that happens in RNN, happens here. The LSTM model often forgets the content of distant positions in a sequence when the sequences are long.

in summary, LSTM and RNN have the following problems:

- It's hard to maintain parallelization when processing sentences , since you have to process word by word.
- “distance” between positions is linear.
- no explicit modelling of long and short range dependencies.

Attention with RNN.

researchers created a technique for paying attention to specific words in a sequence. Attention is a technique that is used in a neural network. For RNN, instead of only encoding the whole sentence in the hidden state, each word has its own corresponding hidden state that is passed all the way to the decoding stage.

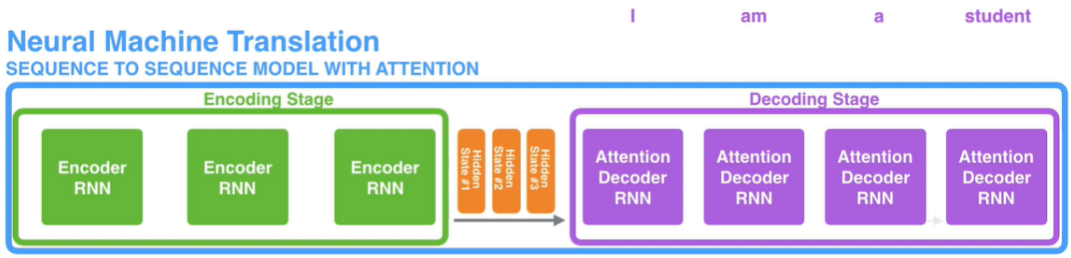


Figure 38: RNN with attention.

to understand how things would look like, let's take this example:

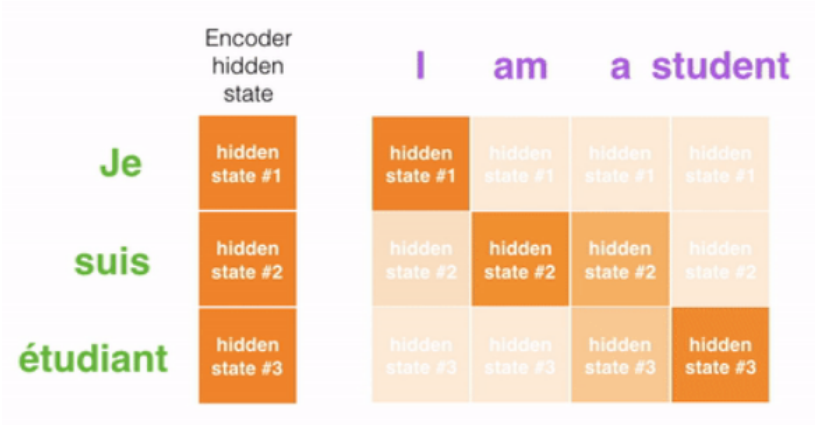


Figure 39: an example of translation with attention RNN.

In this example, all the hidden states are given to the decoder, where each output stage gives each hidden state a degree (weight) of how important this piece of information is useful in predicting the current output.

The problem here is that RNN with attention didn't solve the problem of sequential processing. for a large corpus of text, this increases the time spent translating the text.

Convolutional Neural Networks (CNNs)

with convolutional neural networks, with them we can:

- parallelize processing.
- exploit local dependencies
- The distance between positions is logarithmic.

In CNN, each word on the input can be processed at the same time and does not necessarily depend on the previous words to be translated. Not only that, but the "distance" between the output word and any input for a CNN is in the order of **log N** that is **the size of the height of the tree generated from the output to the input** .That is much better than the distance of the output of a RNN and an input, which is on the order of N.

The problem is that Convolutional Neural Networks do not necessarily help with the problem of figuring out the problem of dependencies when translating sentences. That's why **Transformers** were created, they are a combination of both CNNs with attention.

Transformers

To solve the problem of parallelization, Transformers try to solve the problem by using encoders and decoders together with **attention models.** Attention boosts the speed of how fast the model can translate from one sequence to another.

Let's take a look at how **Transformers** work. Transformer is a model that uses **attention** to boost the speed. More specifically, it uses **self-attention.**

internally, it consists of six encoders and six decoders.

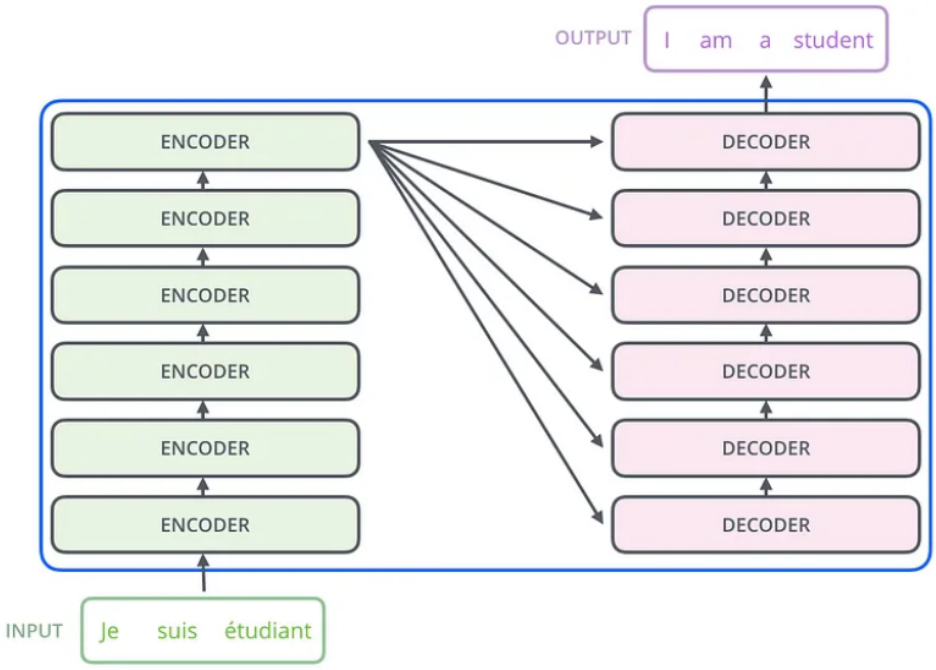


Figure 40: the transformer's encoders and decoders.

where each encoder consists of two layers: **Self-Attention** and a **feed-forward neural network**. The self attention helps the encoder look at other words in the input sequence as it encodes a specific word.

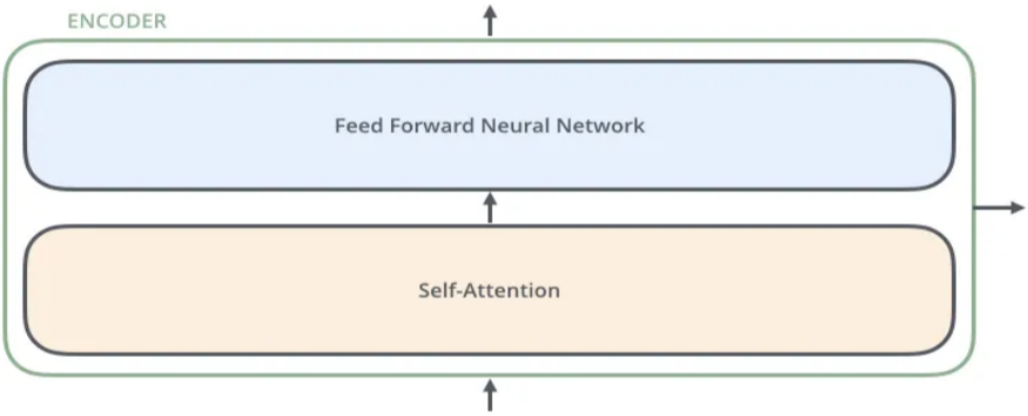


Figure 41: an example of a transformer's encoder.

while the **Decoder** also has both the self attention and feed forward layers but between them, we have an "**Encoder-Decode Attention**" layer, which helps the decoder focus on relevant parts of the input sequence.

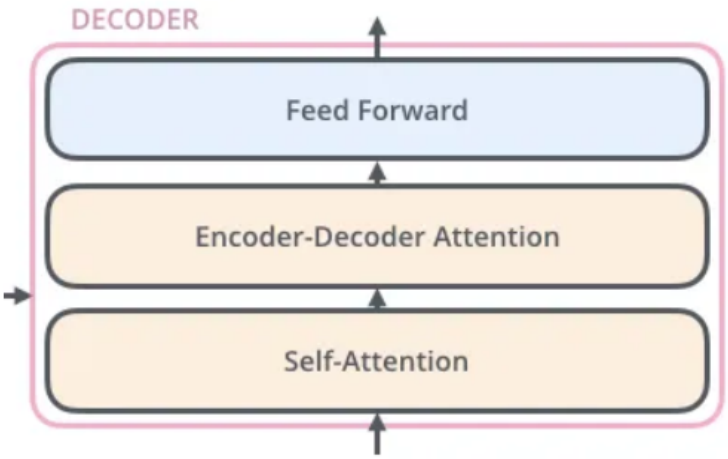
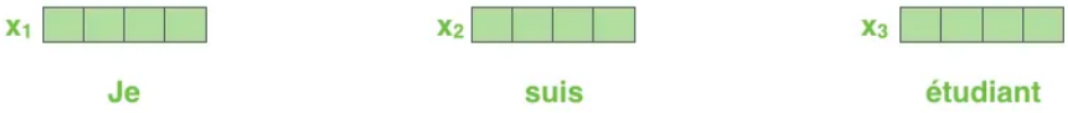


Figure 42: an example of a transformer's decoder.

self attention

to understand self attention, consider the following example:



assume that each word is embedded into a vector of size 512. we'll represent those vectors with these simple boxes.

The embedding only happens in the bottom-most encoder. The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512.

In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that's directly below. After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.

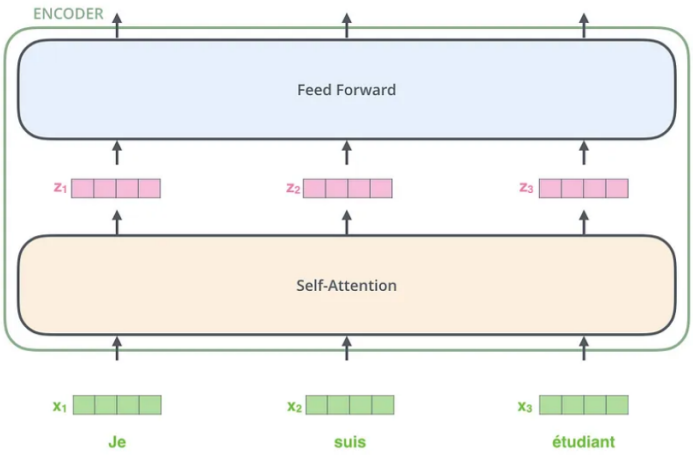


Figure 43: an example of self attention.

Here we begin to see one key property of the Transformer, which is that the word in each position flows through its own path in the encoder. There are dependencies between these paths in the self-attention layer. The feed-forward layer does not have those dependencies, however, and thus the various paths can be executed in parallel while flowing through the feed-forward layer.

4.1.5 How does tokenization work in NLP?

Tokenization is used in natural language processing to split paragraphs and sentences into smaller units that can be more easily assigned meaning.

The first step of the NLP process is gathering the data (a sentence) and breaking it into understandable parts (words).

Here's an example of a string of data:

"What restaurants are nearby?"

In order for this sentence to be understood by a machine, tokenization is performed on the string to break it into individual parts. With tokenization, we'd get something like this:

'**what**' '**restaurants**' '**are**' '**nearby**'

This may seem simple, but breaking a sentence into its parts allows a machine to understand the parts as well as the whole. This will help the program understand each of the words by themselves, as well as how they function in the larger text. This is especially important for larger amounts of text as **it allows the machine to count the frequencies of certain words as well as where they frequently appear.** This is important for later steps in natural language processing.

kinds of tokenization:

- **word tokenization**

Word tokenization is the most common version of tokenization. It takes natural breaks such as spaces and splits the data into respective words using delimiters (characters like “,” or “;”).

While this is the simplest way to do tokenization, it has some drawbacks as it’s hard to separate unknown words or (OOV) words.This is often solved by replacing unknown words with a simple token that communicates that a word is unknown. This is a rough solution, especially since 5 ‘unknown’ word tokens could be 5 completely different unknown words or could all be the exact same word.

- **character tokenization**

  Character tokenization was created to address some of the issues that come with word tokenization. Instead of breaking text into words, it completely separates text into characters. This allows the tokenization process to retain information about OOV words that word tokenization cannot.

  Character tokenization **doesn’t have the same vocabulary issues as word tokenization as the size of the ‘vocabulary’ is only as many characters as the language needs.** For English, for example, a character tokenization vocabulary would have about 26 characters.

  While character tokenization solves OOV issues, it isn‘t without its own complications. By breaking even simple sentences into characters instead of words, **the length of**

the output is increased dramatically. With word tokenization, our previous example “what restaurants are nearby” is broken down into four tokens. By contrast, character tokenization breaks this down into 24 tokens, a 6X increase in tokens to work with.

Character tokenization also adds an additional step of understanding the relationship between the characters and the meaning of the words. Sure, character tokenization can make additional inferences, like the fact that there are 5 “a” tokens in the above sentence. However, this tokenization method moves an additional step away from the purpose of NLP, interpreting meaning.

- **sub word tokenization**

Sub word tokenization is similar to word tokenization, but it breaks individual words down a little bit further using specific linguistic rules. One of the main tools they utilise is breaking off affixes. Because prefixes, suffixes, and infixes change the inherent meaning of words, they can also help programs understand a word’s function. This can be especially valuable for out of vocabulary words, as identifying an affix can give a program additional insight into how unknown words function.

The sub word model will search for these sub words and break down words that include them into distinct parts. For example, the query **“What is the tallest building?”** would be broken down into **‘what’ ‘is’ ‘the’ ‘tall’ ‘est’ ‘build’ ‘ing’**

How does this method help the issue of OOV words? Let's look at an example:

Perhaps a machine receives a more complicated word, like **'machinating'** (the present tense of the verb 'machinate' which means to scheme or engage in plots). It's unlikely that machinating is a word included in many basic vocabularies.

If the NLP model was using word tokenization, this word would just be converted into just an unknown token. However, if the NLP model was using sub word tokenization, it would be able to separate the word into an **'unknown'** token and an **'ing'** token. From there it can make valuable inferences about how the word functions in the sentence.

But what information can a machine gather from a single suffix? The common 'ing' suffix, for example, functions in a few easily defined ways. It can form a verb into a noun, like the verb 'build' turned into the noun 'building'. It can also form a verb into its present participle, like the verb 'run' becoming 'running.'

If an NLP model is given this information about the 'ing' suffix, it can make several valuable inferences about any word that uses the sub word **'ing**.' If 'ing' is being used in a word, it knows that it is either functioning as a verb turned into a noun, or as a present verb. This dramatically narrows down how the unknown word, 'machinating,' may be used in a sentence.

- **Byte Pair Encoding Tokenization**

  This is the type of tokenization we're using in our task.

  BPE is a simple form of **data compression** algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data. it ensures that **the most common words are represented in the vocabulary as a single token while the rare words are broken down into two or more subword tokens and this is in agreement with what a subword-based tokenization algorithm does.**

  here's an example of does it work:

  Suppose we have a corpus that has the words (after pre-tokenization based on space) — **old**, **older**, **highest**, and **lowest** and we count the frequency of occurrence of these words in the corpus. Suppose the frequency of these words is as follows:

  {"old": 7, "older": 3, "finest": 9, "lowest": 4}

  Let us add a special end token "</w>" at the end of each word.

{"old</w>": 7, "older</w>": 3, "finest</w>": 9, "lowest</w>": 4}

The “</w>” token at the end of each word is added to **identify a word boundary so that the algorithm knows where each word ends**.

consider the following steps:

1. split each of the previous words into characters and count their frequency of occurrence.

Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13
11	t	13
12	w	4

2. look for the most frequent pairing, merge them, and update the table as following:
    - calculate the frequency of the merged pairs.
    - calculate the frequency of other related tokens in the table where: having e (has count 16) and s (has count 13), the frequency of e will be (16-13=3), the frequency of s will be (3-3=0) and from the above pre tokenization, the es has frequency of (9+4=13)

Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13
12	w	4
13	es	9 + 4 = 13

- We will now merge the tokens “es” and “t” as they have appeared 13 times in our corpus.

Number	Token	Frequency
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13

- Let’s work now with the “</w>” token. We see that the byte pair “est” and “</w>” occurred 13 times in our corpus.

Number	Token	Frequency
1	</w>	23 - 13 = 10
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13 - 13 = 0
15	est</w>	13

- The steps are repeated until we reach the preferable number of tokens or no more frequent pairs exist.

Number	Token	Frequency
1	</w>	10
2	o	4
3	l	4
4	e	3
5	r	3
6	f	9
7	i	9
8	n	9
9	w	4
10	est</w>	13
11	old	10

from the previous example ,The tokens with 0 frequency count have been removed from the table. We can now see that the total token count is 11, which is less than our initial count of 12.

This is a small corpus but in practice, the size reduces a lot. This list of 11 tokens will serve as our **vocabulary**.

disclaimer: merging is applied in order to generate vocabs, while splitting is applied in order to tokenize the input.

the benefits of this type of tokenization over other methods are:

1. decreased total number of tokens.
2. decreased total vocabulary.

4.2 Model Creation Flow.

Creating a robust and efficient language model is a multi-step process that involves various stages of data processing, model training, evaluation, optimization, and deployment. The transformer architecture, known for its ability to capture complex linguistic patterns, offers a powerful framework for developing state-of-the-art language models. In this section, we will explore the key steps involved in creating a transformer-based language model to help us in Text-to-Arxml generation tasks.

4.2.1 DataCreation

Data creation is a crucial step in developing a transformer-based language model, as it forms the foundation for training the model and enabling it to generate accurate and contextually relevant responses. However, in our project, such as working with Autosar XML configuration files, obtaining an available dataset can be a

challenging task due to the unique nature of the data and its dependency on user requirements. This situation restricts access to pre-existing datasets and necessitates the generation of a custom dataset tailored to the specific needs of the project.

To initiate the data creation process, it is essential to collaborate closely with users, domain experts, or relevant stakeholders to gather comprehensive input text descriptions. These descriptions can range from high-level requirements to detailed specifications, capturing the intended functionality, constraints, and desired behaviour of the Autosar XML configuration files.

4.2.1.1 How We Create Our data.

This part outlines the steps required to generate an example of data in the form of a Text to ARXML configuration file. The process involves utilising our Autosar configuration tool and a user-friendly graphical user interface (GUI) designed to facilitate data creation. By following these steps, users can effectively configure their desired parameters and generate ARXML configuration files.

Steps for Generating Text to ARXML Configuration File Synthesised Data:

Step 1: Utilise the Autosar Configuration Tool The user initiates the process by using our Autosar configuration tool. This tool allows them to set the parameters according to their requirements and subsequently generates an ARXML configuration file.

Step 2: Utilise the Data Creation GUI To aid in the data creation process, we have developed a user-friendly GUI. Using this GUI,

the user can input detailed prompts that describe the selected parameters they have chosen within the Autosar configuration tool.

Step 3: Generate an Example in the Synthesised Data Set With the ARXML configuration file generated from the Autosar configuration tool and the user's input prompts from the GUI,

Step 4: Repeat the Previous Steps for Data Generation To create a substantial amount of data, the user can **repeat** Steps 1 to 3 iteratively. By modifying the parameters in the Autosar configuration tool, generating ARXML configuration files, and providing corresponding prompts in the GUI, multiple examples can be synthesised and added to the data set.

Conclusion: Following the outlined steps, users can effectively generate data in the form of Text to ARXML configuration files. By leveraging our Autosar configuration tool, the data creation GUI, and the iterative process.

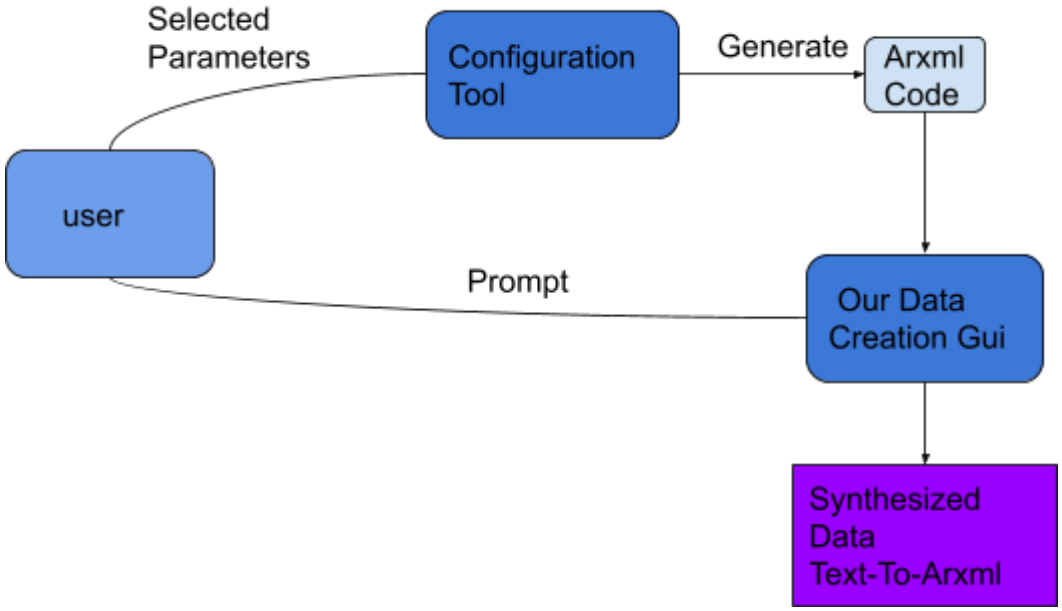


Figure 44: How we Create our DataSet.

We have used this approach to generate approximately **150** examples which contain the user detailed prompt against the generated Arxml file for the configuration tool. You can Look for previous figures to understand how we create the data, Let's look onto our data.

4.2.1.2 Prompt In our Data

For this task find the good way that represent the prompt the describe what he parameters user required is one of the hardest task in our project , We not need to use a prompt that contains a lot of informations about all the parameters that will generated , that will increase the number of tokens represent the data example, we have decided to let the user just type an normal text like "please create Adc module ..." this help us to take input without any complex restriction for the input.

here an example that we follow in our data:

" create Adc module , det init api enabled ,dev error detect enabled , enable queuing enabled ,enable stat group api ,hw trigger api ,result alignment to right ,one hardware unit ,one channel ,one group ,max channel resolution is 16"

4.2.2 Data Preprocessing

the crucial data preprocessing steps required to address the challenges associated with limited data availability in the creation of Text to ARXML configuration files. As the generation of a large-scale dataset is essential for effective fine-tuning of complex

architectures like transformers, we employ two key processes: data cleaning and data augmentation. These processes aim to enhance the dataset size and quality, thereby enabling more accurate and robust model training.

4.2.2.1 Data Cleaning.

Text data cleaning plays a critical role in preparing data for Natural Language Processing (NLP) tasks. This process involves several steps to ensure the quality and consistency of the text data, we have applied many processes that perform the text cleaning on our prompt.

steps we follows:

- **Lowercasing**

  Converting all text to lowercase helps ensure consistency and reduces the complexity of the vocabulary.

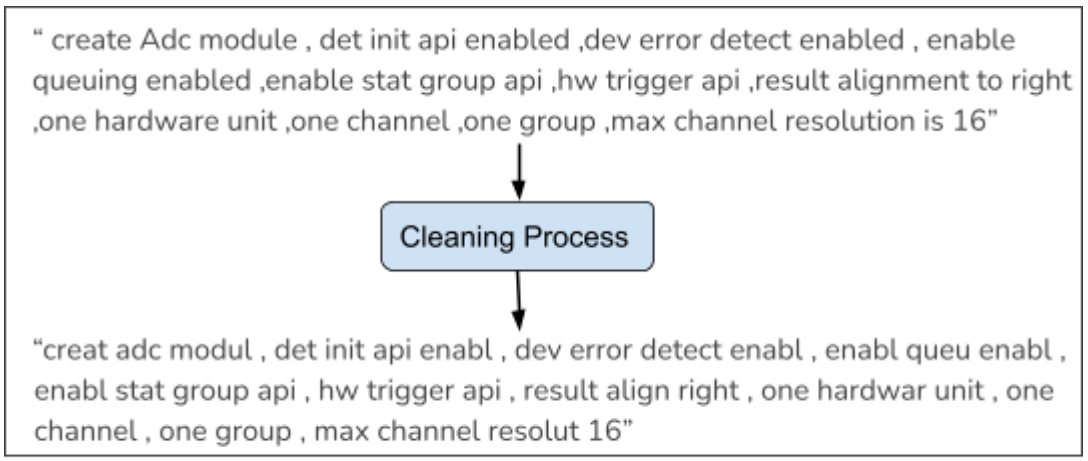
- **Stop Word Removal**

  Stop words are common words (e.g., "the," "is," "a") that do not carry significant meaning and can be safely removed.

- **Stemming**

  stemming removes prefixes and suffixes to obtain the root form of words.

The Following Example how the cleaning process affect the prompt text:



4.2.2.2 Data Augmentation.

Given the limited number of examples available, data augmentation techniques are employed to generate a larger number of diverse examples. This process involves:

Character Augmentation

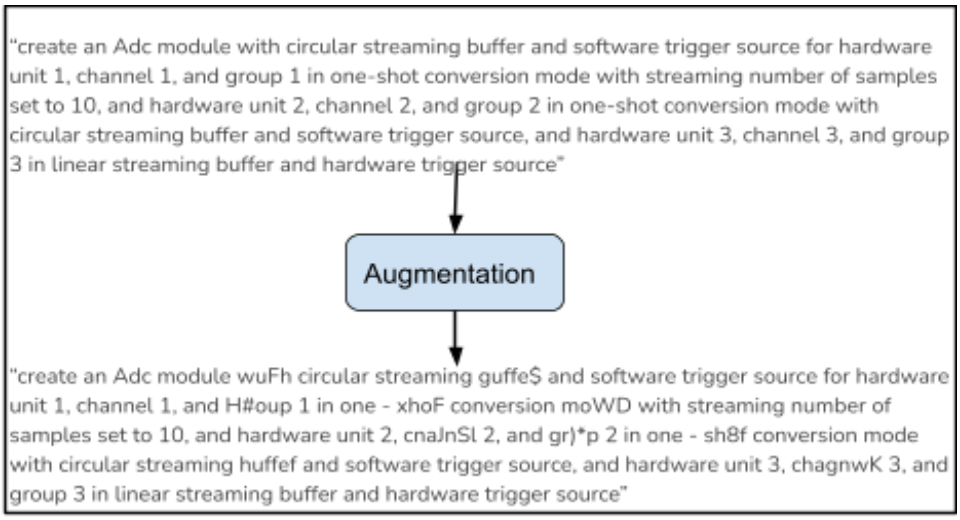
1. keyBoard Distant Augmentation
2. Swap Character Augmentation
3. Delete Character Augmentation
4. Substitute Character Augmentation

Word Augmentation

1. WordSpelling Augmenter
2. Word Swap RandomAugmenter
3. Word Delete Random Augmenter
4. Word Split Random Augmenter

We have applied this augmentation on our created synthesised data set ”Prompts” which are **150** examples after we created so we now have **500** examples which help us to train our model .

Note: we can't generate more augmented data that may lead the model to reach a large overfitting on our data since the data is repeated many times.



4.2.3 Model Training.

We fine tune a OpenAi Transformer **GPT-Neo 125m** model, there are several steps involved, including data preparation, tokenization, and model training.We, using Hugging Face Transformers to fine tune the model and W&B for performance visualisation can greatly enhance the training process. Here's a detailed description of each step.

To fine-tune the GPT-Neo model, I leveraged the advanced capabilities of the Hugging Face Transformers library. This library provided me with a comprehensive toolkit for customising and adapting the pre-existing GPT-Neo architecture to my task at hand.

By configuring the model architecture, tuning hyperparameters, and setting up the training process, I ensured that the model was optimised for generating accurate and coherent ARXML code from input text.

4.2.3.1 Prepare Data For Model Training.

For fine tune or model we need to prepare the synthesised dataset we have created,Our model required data in form of text start with **bos** token and end with **eos** token,this tokens represent the end and the start of every corpus passed to the model,so we need to prepare our data to become like this <bos>corpus<eos> , great now what about prompt and the code , the text generation model actually doesn't different what it’s corpus of data consists of it just text and try to predict the next token usually but how we could different between the prompt and the code in this corpus greate we need to separate between the code and prompt so we decided to use another token we put it “ space “ this falls between the prompt and the code to can separate between them in the predicted text so now we can find the start of the code what about its end ? Great, we used another tooken added after the code to now it’s end this token we put it as “ end “ token which represents the end of my predicted text so now how the data is feeded in the model like the following frame.

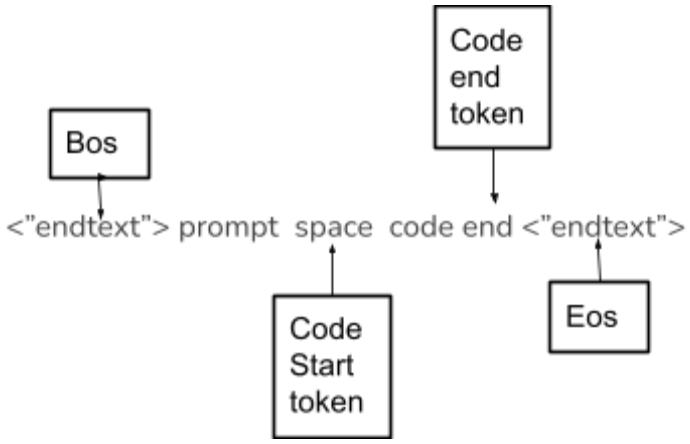


Figure 45: the frame of data passed to the gpt model.

4.2.3.2 Tokenize Our Data Set.

For tokenization we have use the pre training tokenizer of Open AI Gpt Neo Model to split our data into tokens we actually passed the data as corpus of text but here is some points to stop when we talk about the tokenization:

- **maxLength**:This is the important parameter when we talk about the tokenization process ,this mode defines the maximum sequence length that our model might ever be used with. Here we used **2048, this number also controls the size of the model.**
- **Vocab Size:**vocabulary size of the model. Defines the different tokens that can be represented by the *inputs_ids* passed to the forward method of the model. we used the default here **50257.**
- **truncation:**we apply the truncation process to remove any tokens that increased than 2048 so we can just pass to our model the 2048 token only.

- **ReturnOverflowingTokens:** we apply this for not losses the data that passed from the truncation process so if we have example exceed the 2048 token we divided into many examples each in size of 2048 tokens,as example : if we have an examples “prompt space code end “ with 4000 **tokens** will divide into two examples takes the first 2048 token as first example and the remaining 1952 **tokens** as the other example.
- **Padding:**what if our example is less than 2048 tokens by apply the padding ,padding add a <pad> token to complete the 2048 tokens and by using an attention mask we put in this tokens places to zeros so the model can't dive it any

We apply a tokenizer for our data which is train,test,validate then generate inputs ids it’s the numerical representations for each word token that the tokenizer generates.

What about the distribution of the number of tokens in our data after tokenization?

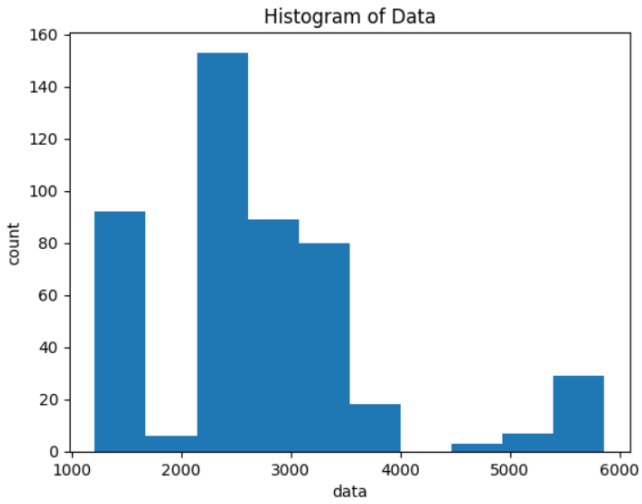


Figure 46: Numbers of Tokens in Text-To-Arxml Data Set.

From the above figure we can understands that data Text-to-Arxml contains large number of tokens and there is a lot of examples greater than 2048 token which may lead to not reach the end of the file which lead as to bad results.or required an bigger model with large amount of parameters that not enough for our memory.

4.2.3.3 Select Parameters and Train our model.

We fine tune the gpt neo 125M parameters model using our data text-to-arxml so we decided to start with 5 epochs training by applying gradient accumulation , gradient accumulation not the best thing in out case due to our small data set so we set it to one

gradient accumulation: it's combine the gradients of mini batch of data then sum all this gradients to get the result for on batch main used for large data and small amount of memory

this are the following parameters we use :

- learning_rate: 5e-05
- train_batch_size: 2
- eval_batch_size: 2
- seed: 42
- distributed_type: multi-GPU
- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- lr_scheduler_type: linear
- num_epochs:5

Result of Training and Validation

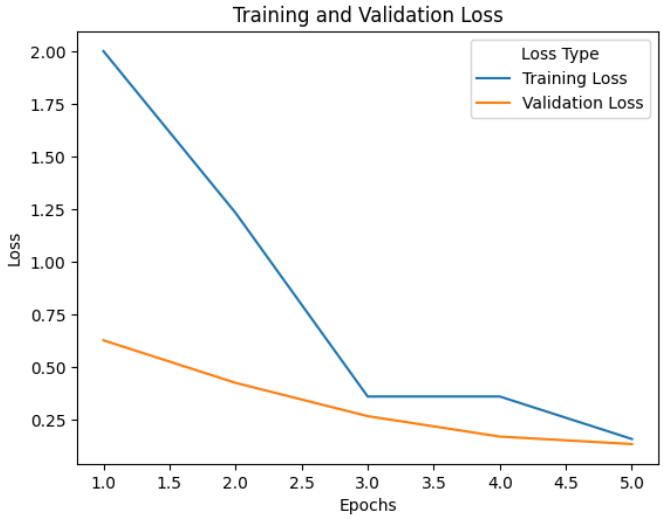


Figure 47: Result of training and validation Loss of Text-to-Arxml on Gpt neo 125M.

4.2.4 Model Evaluation

The evaluation in this task is very complex we need to check if the output code is the same syntax with the real one ,but here is very complex things we need to take care off like the parameters order for every container is it something we need to take care of or not if we should we can apple line by line comparison but we not need this ,there is no difference between the order of parameters in the same container so, we need something to take care of the structure of the code and the parameter values as match as possible.

Evaluation metrics we used:

- **Bleu metrics**: for these metrics which depend on the number of tokens in the out found in the real or we call it human reference which indicate here the real code we want to predict , calculate the modified precision by using the clipping for the

number of token repetition in the predicted code this called modified precision.

- bleu in our task is reaches to better result compared to the other metrics bleu is sensitive to any parameter value and losing container part since it’s depend on the length on the text in its evaluation.one of the point the seems so good for using rouge it’s not depend on the context or the meaning of the text which seems good when you used it in code evaluation.
- **Rouge metrics:** Rouge actually calculate results based on the number of words in references founded in the prediction then divide it by number of words for references this how we got the recall actually rouge also calculate the precision and F1 score , the last thing there's is are different type of rouge score , for rouge-1 we use just one gram , rouge-2 we use two grams and the last one is the rouge L which called the longest sequence of words (not necessarily consecutive, but still in order).Rouge very strong in catch the structure compare to the bleu score.

4.2.4.1 Results and Conclusion.

To get the results of the model we need to pass for it the prompt then compare the predicted with the result. Let's remember that our data number of tokens is larger than 3000 so since the model cant predict more than 2048 so this means we may lose some from our code output. Let's try to give our model the following prompt.

prompt:” create Adc module , det init api enabled ,dev error detect enabled , enable queuing enabled ,enable stat group api ,hw trigger

api ,result alignment to right ,one hardware unit ,one channel ,one group ,max channel resolution is 16”.

Model Results for testing:

- **Bleu Score: 0.722.**
- **rouge 1: 0.82.**
- **rouge 2: 0.82.**
- **rougeL: 0.82.**

Conclusion:

The first thing I got notice when I look to the output code I founded not complete this return us to the number of points we can't exceeded for our hardware limitation plus our model architecture which is 2048 , this example compare to others looks good since this scores not greate compare to result plus we can't find the file end that seems big problem how can I end the generation so all this constraint we founded at this point I next will talk how we passes it.

4.2.5 Optimization

Now we found our self with bad result and complete code what we want to do now is try to solve it step by step, first we need to to do something to can get better result but how we can do this for Large Language model like GPT we use something called PPO which help us to apply the concept of **RLHF** , this will lead to enhance the results of every generation of an piece of code , second this is how to solve the problem of number of tokens that our data consists off so for that we used something called deepspeed from microsoft which help us to train on cloud gpu after some configuration.

4.2.5.1 PPO

How the greatest trending tool now days works yes chat gpt actually the first time in open ai there train language mode Gpt-3 on this huge amount of data then goes to evaluation find that result is so bad but how they enhance the result , using reinforcement learning in the this task is something challenging, open ai have introduced paper called **PPO(proximal policy optimization) ,** we can abstract it by learning from my bad results trying to reach good result.

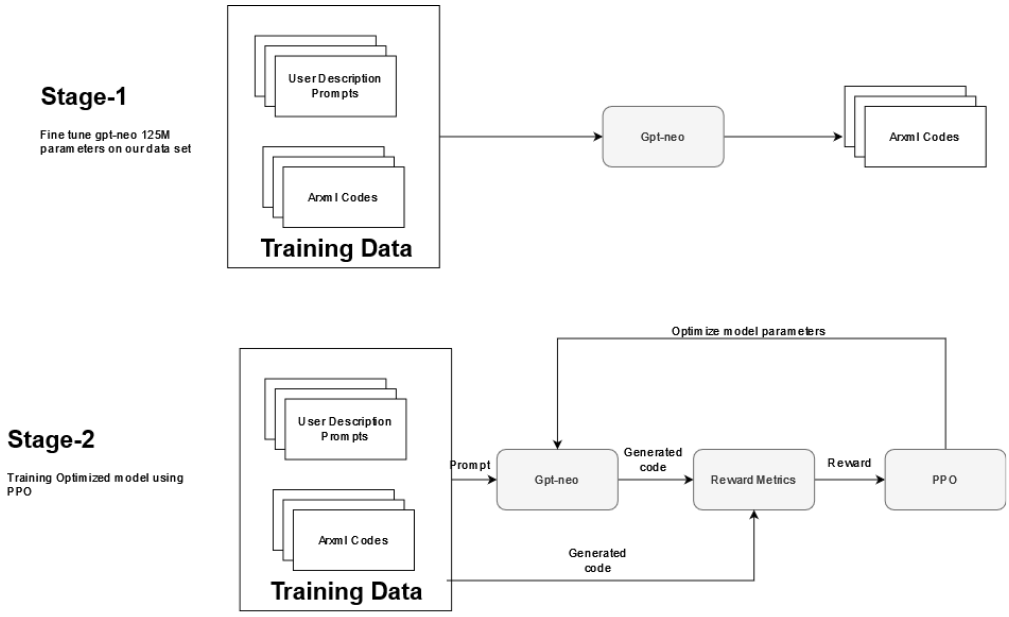


Figure 48: How we train our PPO model.

How we train our PPO model:

- **step1**: We have built our synthesised data into a text-to-arxml dataset.

- **step2**: We have to fine tune the Gpt neo 125M on our data and give it a name called the main model.
- **step3**: Copy this model to another one and save it as a reference model.
- **step4**: We choose one of the reward metrics which are **blue** to evaluate the model prediction compared to real.
- **step5**: We have used the TRL library to run the PPO on hugging face gpt transformer.we pass to it the main model and reference mode.
- **step6**: Start the training process for PPO at every step the main model will generate an output then will give it a score compared to the real one on bleu metric.
- **step7**: Send the result code and the metric evaluation for TRL to enhance the result of the model and use the reference model to not deviate too far from the result.
- **step8**:Evaluate the results of the new model on our test case.

Conclusion: For using something like PPO for training we need to train on a large number of epochs for this task which takes so much time and large amounts of memory so we are also limited by our free cloud hardware.

4.2.5.2 DeepSpeed

DeepSpeed is a deep learning optimization library that enables highly optimised training of large-scale deep learning models. This tool leverages a unique combination of model parallelism and pipeline parallelism to achieve a higher degree of performance optimization. By distributing the computation across multiple GPUs, DeepSpeed allows for faster training times and improved model accuracy.

We try to use this solution to train the model with a large number of sequence more than 2048 tokens so we can generate until the end of the file and by this way we can fix the errors ,but we need to test to check the results.

who deepspeed/Zero works:

At the begin deepspeed split the data on the available n GPUs ,every parameters takes the full F16 parameters and F16 Gradients,we split the model parameters across the GPUs then , every gpu broadcast each model parameter then train on it's data then all apply gradient accumulation at the main gpu then this done at the others then the losses are calculated then we run the backpropagation one by one until the new gradients calculated then that done on all GPUs until all the gradients are accumulated then the gradients are offloaded if you want to activate it .

Conclusion

We applied the deepspeed in our model and we have ability to run up **125.6 M** parameter model using up to **4000 sequence length but we found that the result for long sequence are not good and the code contains a lot of mistakes since the model architecture are not trained on**

4.2.6 Constraints

We now we got some of the bad results and we try to solve the problems but we not got an good result after the optimization so we set this constraints on our paper they are:

- Small amount of Data with bad quality prompt
- Large number of tokens that represent the data
- How to get the end of the code.

4.2.6.1 How we solve the constraints

We have an great idea from the best member in the team “sarah”,that we will not use the xml code directly why not use the **json code format** ,great let's try to build new data with good prompt that describe only the user non default values with direct way not need a lot of text we have use something like that:

prompt:” create Adc module , det init api enabled ,dev error detect enabled , enable queuing enabled ,enable stat group api ,hw trigger api ,result alignment to right ,one hardware unit ,one channel ,one group ,max channel resolution is 16”.

ok this prompt just describes what the user non default values that need to change anything not said here should not appear in the prompt.

For the data we have build **174 examples** using the same way we said above for data creation but we have change the code from **xml to json**,then after apply the above types of augmentations we got the **500 examples** so we now have ability to check the new distribution for the data.

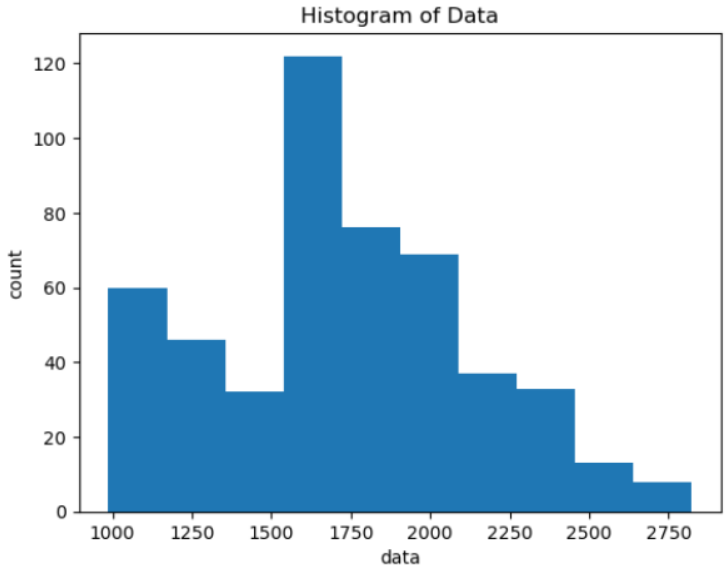


Figure 49: Numbers of Tokens in Text-To-Json Data Set.

Great, we have found the distribution that concentrated at 1500-1750 so we have moved from 3000-4000 to small range so this may lead to great results.

4.2.7 ReTraining and Evaluation

Now we will apply the same steps in the above training point but on the new data text to json which contains tokens that less than 2048

I have fine tune the same mode gpt neo 125M parameter model on the new data set on the following parameters:

- learning_rate: 0.0002
- train_batch_size: 2
- eval_batch_size: 2
- seed: 42
- optimizer: Adam with betas=(0.9,0.999) and epsilon=1e-08
- lr_scheduler_type: linear

- num_epochs: 15

Training and evaluation loss:

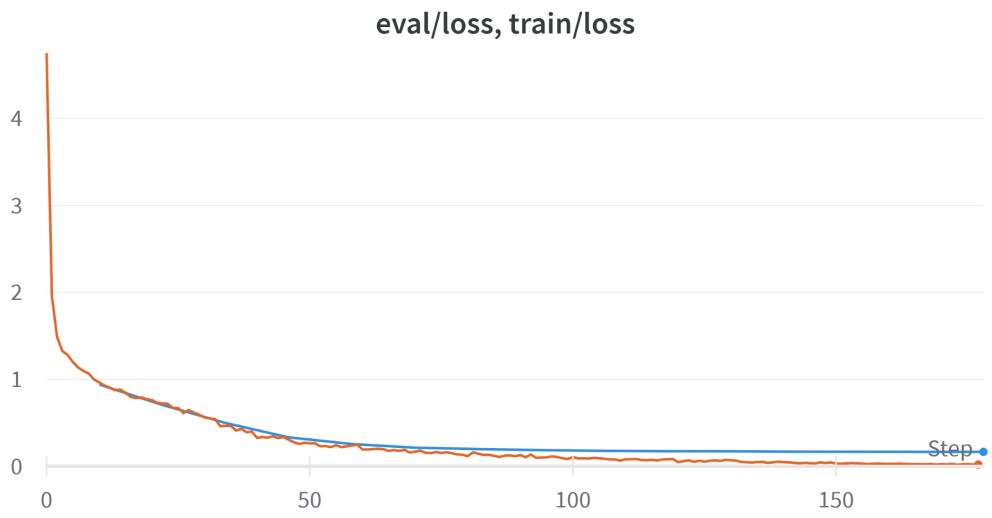


Figure 50: Result of training and validation Loss of Text-to-Json in Gpt neo 125M.

After Running the 8 Examples From Test Data set we got this scores:

- **bleu: 0.85**
- **rouge 1: 0.912**
- **rouge 2: 0.898**
- **rouge L: 0.872**

Results:

prompt:" create Adc module , det init api enabled ,dev error detect enabled , enable queuing enabled ,enable stat group api ,hw trigger api ,result alignment to right ,one hardware unit ,one channel ,one group ,max channel resolution is 16".

Code Output:


```
{
  "ECUC-MODULE-CONFIGURATION-VALUES": {
    "@Dest": "ECUC-MODULE-DEF",
    "@DEST": "BSW-IMPLEMENTATION",
    "SHORT-NAME": "Adc",
    "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc",
    "IMPLEMENTATION-CONFIG-VARIANT": "VARIANT-POST-BUILD",
    "MODULE-DESCRIPTION-REF": "AUTOSAR/EcucDefs/Adc",
    "CONTAINERS": {
      "ECUC-CONTAINER-VALUE": [
        {
          "SHORT-NAME": "AdcGeneral",
          "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral",
          "PARAMETER-VALUES": {
            "ECUC-BOOLEAN-PARAM-DEF": [
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcHwTriggerApi",
                "VALUE": "True"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcEnableQueuing",
                "VALUE": "True"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcEnableStartStopGroupApi",
                "VALUE": "False"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcEnableLimitCheck",
                "VALUE": "False"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcGrpNotifCapability",
                "VALUE": "False"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcDevErrorDetect",
                "VALUE": "False"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcDeInitApi",
                "VALUE": "True"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcReadGroupApi",
                "VALUE": "True"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcVersionInfoApi",
                "VALUE": "True"
              }
            ],
            "ECUC-ENUMERATION-PARAM-DEF": [
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcResultAlignment",
                "VALUE": "ADC_ALIGN_RIGHT"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcGeneral/AdcPriorityImplementation",
                "VALUE": "ADC_PRIORITY_HW"
              }
            ]
          }
        }
```

```
,
        {
          "SHORT-NAME": "AdcConfigSet",
          "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet",
          "SUB-CONTAINERS": {
            "ECUC-CONTAINER-VALUE": {
              "SHORT-NAME": "AdcHwUnit",
              "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit",
              "PARAMETER-VALUES": {
                "ECUC-INTEGER-PARAM-DEF": {
                  "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcHwUnitId",
                  "VALUE": "0"
                }
              },
              "SUB-CONTAINERS": {
                "ECUC-CONTAINER-VALUE": [
                  {
                    "SHORT-NAME": "AdcChannel",
                    "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcChannel",
                    "PARAMETER-VALUES": {
                      "ECUC-INTEGER-PARAM-DEF": {
                        "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcChannel/AdcChannelId",
                        "VALUE": "0"
                      }
                    }
                  },
                  {
                    "SHORT-NAME": "AdcGroup",
                    "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup",
                    "PARAMETER-VALUES": {
                      "ECUC-ENUMERATION-PARAM-DEF": [
                        {
                  "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcGroupAccessMode",
                          "VALUE": "ADC_ACCESS_MODE_SINGLE"
                        },
                        {
         "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcGroupConversionMode",
                          "VALUE": "ADC_CONV_MODE_CONTINUOUS"
                        },
                        {
        "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcGroupTriggSrc",
                          "VALUE": "ADC_TRIGG_SRC_HW"
                        },
                        {
          "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcStreamingBufferMode",
                          "VALUE": "ADC_STREAM_BUFFER_CIRCULAR"
                        }
                      ],
                      "ECUC-INTEGER-PARAM-DEF": [
                        {
                          "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcGroupId",
                          "VALUE": "0"
                        },
                        {
     "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcGroup/AdcStreamingNumSamples",
                          "VALUE": null
                        }
                      ]
                    },
                    "REFERENCE-VALUES": {
                      "ECUC-REFERENCE-VALUE": {
                        "DEFINITION-REF": "AdcGroupDefinition",
                        "VALUE-REF": "/AUTOSAR/EcucDefs/Adc/AdcConfigSet/AdcHwUnit/AdcChannel"
                      }
                    }
                  }
                ]
              }
            }
          }
        },
        {
          "SHORT-NAME": "AdcPublishedInformation",
          "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcPublishedInformation",
          "PARAMETER-VALUES": {
            "ECUC-BOOLEAN-PARAM-DEF": [
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcPublishedInformation/AdcChannelValueSigned",
                "VALUE": "False"
              },
              {
                "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcPublishedInformation/AdcGroupFirstChannelFixed",
                "VALUE": "False"
              }
            ],
            "ECUC-INTEGER-PARAM-DEF": {
              "DEFINITION-REF": "AUTOSAR/EcucDefs/Adc/AdcPublishedInformation/AdcMaxChannelResolution",
              "VALUE": null
            }
          }
        }
      ]
    }
  }
}
```

4.2.7 Model Inference and Deployment

For model Inference we Have used the beam search to find all the hidden high probability.
Beam search reduces the risk of missing hidden high probability word sequences by keeping the most likely num_beams of hypotheses at each time step and eventually choosing the hypothesis that has the overall highest probability. Let's illustrate with num_beams=2:

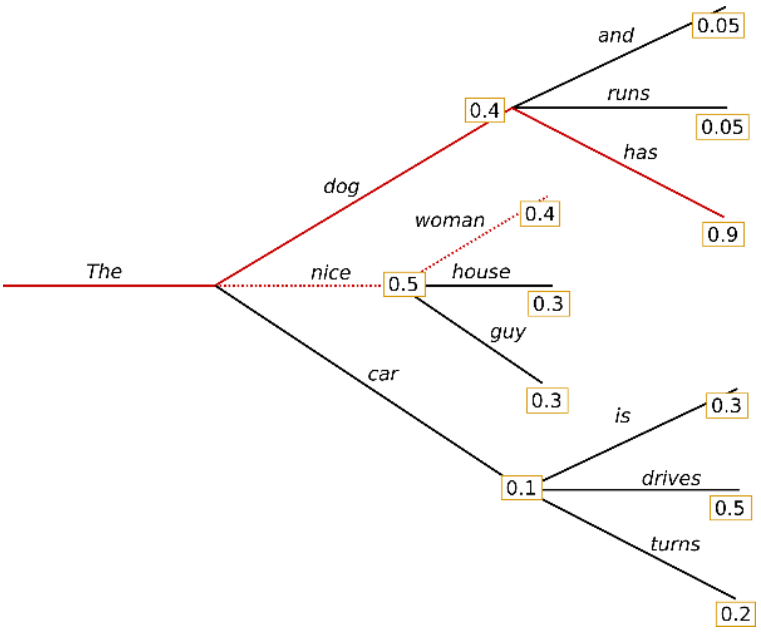


Figure 51: Beam Search.

For Inference there is a trade off between the number of beams and the number of time taken for the inference the more the beam number the largest time for inference but will reach a good result.

We have use an window methods for prediction by apply at first 1024 to generate then generate 256*4 times using the window method.

For the Deployment we have deployed the mode on the hugging face servers which help us to get the model evaluation using tensor board and the api for inference.

Chapter 5 : Testing Module

5.1 Test BSWMD parsing to GUI

The purpose of this test is to verify that the BSWMD parsing functionality correctly parses the BSWMD (Basic Software Module Description) data and populates the GUI (Graphical User Interface) with the relevant information.

Expected Results:

- The GUI should successfully parse the BSWMD file without any errors.
- The GUI should accurately display the parsed BSWMD data, including all relevant information such as module names, descriptions, interfaces, etc.
- All the required fields in the GUI should be populated with the correct information from the BSWMD file.
- In case of errors or invalid BSWMD files, the application should display appropriate error messages guiding the user to rectify the issues.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	Valid BSWD File: Provide a valid BSWD file to the GUI for parsing.	Verify that the GUI successfully reads and parses the file without any errors	Verify that the GUI successfully reads and parses the file without any errors.	pass.
TC_02	Check that the	Displaying data	Displaying	pass.

	extracted data is displayed correctly in the GUI interface.	in GUI without extracting data correctly.	data in GUI without extracting data correctly.	
TC_03	Test the GUI with an empty BSWD file. Ensure that the GUI handles the empty file gracefully	displays an appropriate error message indicating that the file is empty and cannot be parsed.	it doesn't displays an appropriate error message	fail.
TC_04	Test the GUI with a large BSWD file, preferably one that is several megabytes	GUI can handle the large file efficiently without significant performance degradation or memory issues	GUI can handle the large file efficiently without significant performance degradation or memory issues	pass.
TC_05	BSWD File with Dependencies: Provide a BSWD file that contains dependencies between different modules, components, or	Ensure that the GUI correctly resolves and handles these dependencies during parsing, displaying the interrelated data accurately.	Ensure that the GUI correctly resolves and handles these dependencies during parsing, displaying the interrelated data	pass

	parameters.		accurately.	
TC_06	GUI Integration Testing: Test the integration of the BSWD parsing functionality with the rest of the GUI components	The parsed data is properly displayed and utilised within the GUI, allowing users to interact with the parsed BSWD information seamlessly.	The parsed data is properly displayed and utilised within the GUI, allowing users to interact with the parsed BSWD information seamlessly.	pass
TC_07	boolean parameter viewing	boolean parameter appear as checkbox	boolean parameter appear as checkbox	pass
TC_08	enum parameter viewing	enum parameter viewing as drop list to choose one of values	enum parameter viewing as drop list to choose one of values	pass
TC_9	numerical parameter viewing	the view textfield that take input from user	the view textfield that take input from user	pass
TC_10	parameter	the view of this	the view of this	pass

	multiplicity if it's value equal to 0	this parameter is disable to configure and can configure it by apply on checkbox of multicity	this parameter is disable to configure and can configure it by apply on checkbox of multicity	
TC_11	parameter multiplicity if it's value equal to 1	the view of this this parameter is enable to configure and cannot on checkbox of multicity	the view of this this parameter is enable to configure and cannot on checkbox of multicity	pass

5.2 Testing user input parameters

The purpose of this test is to verify the handling of user input parameters in the GUI for generating ARXML module descriptions. This includes validating the user inputs, handling invalid or missing inputs, and ensuring the generation process proceeds correctly based on the provided parameters.

Expected Results:

- The GUI should accept and validate user input parameters without any errors.
- The generation process should start smoothly and progress without any issues.
- The generated ARXML module description should contain all the provided input parameters accurately.

- When invalid or missing inputs are detected, appropriate error messages should be displayed, guiding the user to correct the inputs.
- The application should prevent the generation process from proceeding until all inputs are valid.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	Provide valid input values for all parameters.	the application accepts the input without any errors and processes it correctly.	the application accepts the input without any errors and processes it correctly.	pass.
TC_02	Test the application with default input values	Ensure that the application detects the default input	Ensure that the application detects the default input	pass
TC_03	configure enum values and boolean values	boolean values check box selector and enum choose from drop list	boolean values check box selector and enum choose from drop list	pass
TC_04	Enter input values in an invalid format for specific	Verify that the application detects the invalid format	Verify that the application detects the invalid format	fail

	parameters.	and displays an error message, guiding the user to enter the correct format.	and displays an error message, guiding the user to enter the correct format.	

5.4 ARXML Creation Testing

The purpose of this test is to verify the accurate creation of ARXML files based on the provided module descriptions and configurations.

Expected Results:

- The ARXML creation tool should accurately generate ARXML files based on the provided module descriptions and configurations.
- The generated ARXML files should match the expected structure and contents defined in the module descriptions and configurations.
- The ARXML files should include all the necessary elements and their relationships as specified in the module descriptions.
- Data types, values, and relationships between different elements in the ARXML files should align with the specified configurations.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	Valid Input: Provide valid input data for creating an ARXML file, including all the required elements and attributes.	The ARXML file is generated without any errors and contains the correct structure and data.	The ARXML file is generated without any errors and contains the correct structure and data.	pass.
TC_02	Missing Required Elements: Test the application by excluding one or more required elements from the input data.	that the ARXML file creation process detects the missing elements and displays an appropriate error message in dependency	that the ARXML file creation process detects the missing elements and displays an appropriate error message in dependency	pass
TC_03	Invalid Element Values: Provide invalid values for specific elements in the input data.	The ARXML file creation process detects the invalid values and displays an error message, guiding the user to enter valid values.	an error message doesn’t appear	fail

TC_04	Test the application by providing elements in different orders in the input data.	The ARXML file creation process handles the elements correctly and generates the ARXML file with the expected element order.	The ARXML file creation process sometimes has a different order.	fail
TC_05	Include attributes for elements in the input data and verify that they are correctly duplicate sub container	ensure that the ARXML file creation process handles them appropriately ,by make other one with different name	ensure that the ARXML file creation process handles them appropriately ,by make other one with different name	pass
TC_06	Verify that the generated ARXML file is compatible with other ARXML parsers or tools	Attempt to parse the created ARXML file using different ARXML parsing libraries or tools and ensure that they can correctly read and interpret the file without any errors.	Attempt to parse the created ARXML file using different ARXML parsing libraries or tools and ensure that they can correctly read and interpret the file without any errors.	pass

5.3 ARXML Splitter testing

The purpose of this test is to ensure the proper functionality of the ARXML Splitter tool, which is responsible for splitting a large ARXML file into smaller, manageable parts.

Expected Results:

- The ARXML Splitter tool should successfully split a large ARXML file into smaller files based on the specified splitting criteria.
- Each split file should maintain a valid ARXML structure and contain the necessary elements for consistency.
- The splitting process should preserve the integrity of the data and information within the ARXML file without any loss or corruption.
- The splitter tool should handle boundary cases and different splitting criteria effectively, without performance issues or data loss.
- The splitter tool should handle errors gracefully and provide appropriate error messages or exceptions when encountering invalid or corrupt ARXML files.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	Test Case: Export Single Container	The output file should contain only the container with	The output file should contain only the container with	pass.

		the short name "ContainerA" from the input ARXML file.	the short name "ContainerA" from the input ARXML file.	
TC_02	- Test Case: Export Multiple Containers	The output file should contain all the containers with the short name "ContainerB" from the input ARXML file.	The output file should contain all the containers with the short name "ContainerB" from the input ARXML file.	pass.
TC_03	Export Non-Existent Container	The output file should be empty as there are no containers with the short name "ContainerC" in the input ARXML file.	The output file should be empty as there are no containers with the short name "ContainerC" in the input ARXML file.	pass.
TC_04	Invalid File Path `shortName`: "ContainerD"`fileName`: "nonexistent.ar	An exception or error message indicating that the input file does not exist or cannot be	An exception or error message indicating that the input file does not exist or cannot be	pass.

	xml" (non-existent file)	accessed.	accessed.	

5.4 ARXML Merging testing

The purpose of this test is to ensure the proper functionality of the ARXML Merging tool, which is responsible for merging multiple ARXML files into a single consolidated ARXML file.

Expected Results:

- The ARXML Merging tool should successfully merge multiple ARXML files into a single consolidated ARXML file.
- The consolidated ARXML file should maintain the overall structure and consistency, containing all the necessary elements from the individual files.
- The merging process should preserve the integrity of the data and information within the ARXML files without any loss or corruption.

- The merging tool should handle errors gracefully and provide appropriate error messages or exceptions when encountering invalid or corrupt ARXML files.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	Test Case: Frist Element Path to an existing ARXML file with an `<ECUC-MODULE-CONFIGURATION-VALUES>` element.	The existing ARXML file should be updated with the merged values from the new ARXML file. The merged result should be saved in the specified output file path.	The existing ARXML file should be updated with the merged values from the new ARXML file. The merged result should be saved in the specified output file path.	pass.
TC_02	. Test Case: Second Element Path to a new ARXML file with an `<ECUC-MODULE-CONFIGURATION-VALUES>` element.	The new `<ECUC-MODULE-CONFIGURATION-VALUES>` element from the new ARXML file should be added to the existing ARXML file. The merged result should be saved in the	The new `<ECUC-MODULE-CONFIGURATION-VALUES>` element from the new ARXML file should be added to the existing ARXML file. The merged result should be saved in the specified output	pass.

		specified output file path	file path	
TC_03	Multiple Existing Elements Path to an existing ARXML file with multiple `<ECUC-MODULE-CONFIGURATION-VALUES>` elements	The existing ARXML file should be updated with the merged values from the new ARXML file. The merged result should be saved in the specified output file path.	The existing ARXML file should be updated with the merged values from the new ARXML file. The merged result should be saved in the specified output file path.	pass
TC_04	Non-existent Files(non-existent file)	An error message indicating that the input files cannot be found or accessed.	An error message indicating that the input files cannot be found or accessed.	pass.
TC_05	Invalid Output File Path	An error message indicating that the output file path is invalid or cannot be	An error message indicating that the output file path is invalid or cannot be	pass

		accessed.	accessed.	
TC_06	Empty Files	The merged result should be an empty ARXML file since both input files are empty.	The merged result should be an empty ARXML file since both input files are empty.	pass
TC_07	Missing `<ELEMENTS>` Element Path to an existing ARXML file without an `<ELEMENTS>` element.	: An error message indicating that the `<ELEMENTS>` element is missing in the existing ARXML file.	The file merged file with wrong values	fail
TC_08	Invalid XML Format Path to an existing ARXML file with invalid XML syntax/format	An error message indicating that the existing ARXML file has invalid XML syntax/format.	empty file	fail

5.5 configuration file's parser testing

Function name: initConfigParser()

test objective: make sure that the configuration Arxml file is found successfully and in correct format in order to apply some functionalities.

steps:

- open the tool.
- configure the parameters.
- click save.
- click generate.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	a valid Arxml file.	the input Arxml can be parsed and no problems.	the input Arxml can be parsed and no problems.	pass.
TC_02	missing file path: the test ensures that the code handles exceptions and prints the stack trace.	file not found exception is thrown.	file not found exception is thrown and no actions in the tool.	pass.
TC_03	Empty Arxml	an error	an error	pass.

	file.	appears.	appears.	
TC_04	Arxml file with no <ECUC-CONTAINER-VALUE> tag.	the code will execute without any errors but without any values for the main containers or main file nodes.	the code will execute without any errors but without any values for the main containers or main file nodes.	pass.

Function name: Containers parse_config(NodeList mainNodeList,String conName,Containers container, Subcontainers subcontainer, int type)

test objective: parse the configuration file successfully.

steps:

- open the tool.
- configure the parameters (in some test cases, this step is cancelled to check the code behaviour)
- click save.
- click generate.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	all the containers are	all the containers	all the containers	pass.

	configured.	are parsed successfully.	are parsed successfully.	
TC_02	the user didn't configure any main container.	index out of range exception is thrown when accessing the arraylist **parsedContainers** at any index.	index out of range exception is thrown when accessing the arraylist **parsedContainers** at any index.	pass.
TC_03	only one main container is configured.	we're able to get the parameters' map of the configured container, but an index out of range exception is thrown when accessing other containers' indices in the arraylist **parsedContainers.**	we're able to get the parameters' map of the configured container, but an index out of range exception is thrown when accessing other containers' indices in the arraylist **parsedContainers.**	pass.
TC_04	only two main containers are configured	we're able to get the parameters' map of the configured	we're able to get the parameters' map of the configured	pass.

		containers, but an index out of range exception is thrown when accessing other containers' indices in the arraylist **parsedContainers.**	containers, but an index out of range exception is thrown when accessing other containers' indices in the arraylist **parsedContainers.**	
TC_05	no subcontainer configured.	an index out of range exception is thrown when accessing the arraylist of subcontainers that corresponds to a container.	an index out of range exception is thrown when accessing the arraylist of subcontainers that corresponds to a container.	pass.
TC_06	main node list is empty	the main for loop will not be entered, thus, the **parsedContainers** list will be empty and accessing any index will throw	the main for loop will not be entered, thus, the **parsedContainers** list will be empty and accessing any index will throw	pass.

		the index out of range exception.	the index out of range exception.	
TC_07	The group has no reference values.	the reference values list will be empty, and accessing any index will return an index out of range exception.	the reference values list will be empty, and accessing any index will return an index out of range exception.	pass.

5.4 Generator testing

Function name: void write_in_h()

test objective: make sure that the generated Lcfg.h file can be compiled without any problems.

steps:

- open the tool.
- configure the parameters (in some test cases, this step is cancelled to check the code behaviour)
- click save.
- click generate.

test case id	test case scenario	expected result	actual result	test failure/ success

TC_01	The three main containers are configured and all the parameters required in the template are configured too.	Lcfg.h is generated successfully with no place holder sign ($) appearing.	Lcfg.h is generated successfully with no place holder sign ($) appearing.	pass.
TC_02	the **AdcGeneral** only is configured.	the **no hw units**, **no channels**, **no groups**,**channels in group**, and **Adc published information** parameters will have a placeholder sign ($), compilation fails.	the **no hw units**, **no channels**, **no groups**,**channels in group**, and **Adc published information** parameters will have a placeholder sign ($), compilation fails.	pass.
TC_03	the **AdcPublishedInformation** only is configured.	the **no hw units**, **no channels**, **no groups**,**channels in group**, and **AdcGeneral** parameters will have a placeholder	the **no hw units**, **no channels**, **no groups**,**channels in group**, and **AdcGeneral** parameters will have a placeholder sign ($), compilation	pass.

		sign ($), compilation fails.	fails.	
TC_04	the **AdcConfigSet** only is configured.	the **AdcGeneral** and **Adc published information** parameters will have the sign ($), compilation fails.	the **AdcGeneral** and **Adc published information** parameters will have the sign ($), compilation fails.	pass.
TC_05	no **HW Unit** configured.	placeholder($) will appear in the file, compilation fails.	placeholder ($) will appear in the file, compilation fails.	pass.
TC_06	no **AdcChannel** configured.	placeholder($) will appear in the file, compilation fails.	placeholder($) will appear in the file, compilation fails.	pass.
TC_07	no **Adcroup** configured.	placeholder($) will appear in the file, compilation fails.	placeholder($) will appear in the file, compilation fails.	pass.
TC_08	no reference to a channel in a group.	placeholder($) will appear in the file, compilation fails.	placeholder($) will appear in the file, compilation fails.	pass.

TC_09	no container configured.	placeholders($) will appear in the file, compilation fails.	placeholders($) will appear in the file, compilation fails.	pass.

Function name: void write_in_c()

test objective: make sure that the generated Lcfg.c file can be compiled without any problems.

steps:

- open the tool.
- configure the parameters (in some test cases, this step is cancelled to check the code behaviour)
- click save.
- click generate.

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	The three main containers are configured and all the parameters required in the template are configured	Lcfg.c is generated successfully with no place holder sign ($) appearing.	Lcfg.c is generated successfully with no place holder sign ($) appearing.	pass.

	too.			
TC_02	no **Hwunit** configured.	placeholders ($) will appear in the file, compilation fails.	placeholders ($) will appear in the file, compilation fails.	pass.
TC_03	**AdcHwUnitId** is not configured.	empty string appears in the Hwunit struct, compilation succeeds.	empty string appears in the Hwunit struct, compilation fails.	Fail.
TC_04	**AdcPrescale** is not configured.	empty string appears in the Hwunit struct, compilation succeeds.	empty string appears in the Hwunit struct, compilation fails.	Fail.
TC_05	no **AdcGroup** configured.	placeholders ($) will appear in the file, compilation fails.	placeholders ($) will appear in the file, compilation fails.	pass.
TC_06	**AdcGroupId** is not configured.	empty string appears in the Adc group struct, compilation succeeds.	empty string appears in the Adc group struct, compilation fails.	Fail.

TC_07	no reference channels assigned to the group.	empty string appears in the Adc group struct, compilation succeeds.	empty string appears in the Adc group struct, compilation fails.	Fail.
TC_08	no **AdcChannel** configured.	placeholders ($) will appear in the file, compilation fails.	placeholders ($) will appear in the file, compilation fails.	pass.

5.5 Compiler testing

Function name: void copyFiles ()

test objective: make sure that files are copied successfully.

steps:

- open the tool.
- configure the parameters (in some test cases, this step is cancelled to check the code behaviour)
- click save.
- click generate.
- click compile

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	The current directory doesn't contain the generated Lcfg.h and Lcfg.c	The files are copied successfully.	The files are copied successfully.	pass.
TC_02	The current directory contain the generated Lcfg.h and Lcfg.c	Files are copied successfully.	Files already exist and exceptions are thrown.	Fail.
TC_03	The generate button is clicked but the user didn't make configuration.	No file is generated and No such file found exception is thrown .	The Lcfg.h is generated ,No such file found exception is thrown .	Fail.

Function name: void compileFiles()

test objective: make sure that files are copied successfully.

steps:

- open the tool.
- configure the parameters (in some test cases, this step is cancelled to check the code behaviour)

- click save.
- click generate.
- click compile

test case id	test case scenario	expected result	actual result	test failure/ success
TC_01	It's the first time that the user has clicked on the compile button.	Compilation is done successfully.	Compilation is done successfully.	Pass.
TC_02	The user has changed the configuration and clicked compile again.	Compilation is done successfully	Compilation is done successfully	Pass.
TC_03	The user has misconfigured the Adc and the generated Lcfg.h and Lcfg.c have missing values.	No hex or elf files are generated.	No hex or elf files are generated.	Pass.
TC_04	The files are not copied, but the	No hex or elf files are generated.	The old files still exist.	Fail.

	previous hex and files still exist.			

5.6 dependency testing

Parameters Class

Set_parameters_Dependency_AdcGeneral(Containers container)

TC 1= Containers AdcGeneral

TC 2 = Containers AdcConfigSet

TC 3 = Subcontainers AdcHwUnit

Set_parameters_Dependency_AdcGeneral(Containers container)			
Conditions	TC1	TC2	TC3
IP Type = Containers	T	T	F
IP = "AdcGeneral"	T	F	F
IP contains certain parameters	T	F	F
Actions			
Error Message	F	F	T
Normal flow before setting Initial vals	T	T	F
Set param_dep Initial vals	T	F	F

Set_parameters_Dependency_AdcPublishedInformation(Containers container)

TC 1 = Containers AdcPublishedInformation

TC 2 = Containers AdcConfigSet

TC 3 = Subcontainers AdcHwUnite

Set_parameters_Dependency_AdcGeneral(Containers container)			
Conditions	TC1	TC2	TC3
IP Type = Containers	T	T	F
IP = "AdcPublishedInformation"	T	F	F
IP contains certain parameters	T	F	F
Actions			
Error Message	F	F	T
Normal flow before setting Initial vals	T	T	F
Set param_dep Initial vals	T	F	F

Set_parameters_Dependency_AdcPowerStateConfig(Subcontainers subcontainer)

TC 1 = Subcontainers AdcPowerStateConfig

TC 2 = Subcontainers AdcHwUnite

TC 3 = Containers AdcConfigSet

Set_parameters_Dependency_AdcPowerStateConfig(Subcontainers subcontainer)			
Conditions	TC1	TC2	TC3
IP Type = Subcontainers	T	T	F
IP = "AdcPowerStateConfig"	T	F	F

IP contains certain parameters	T	F	F
Actions			
Error Message	F	F	T
Normal flow before setting Initial vals	T	T	F
Set param_dep Initial vals	T	F	F

Set_parameters_Dependency_AdcHwUnit(Subcontainers subcontainer)

TC 1 = Subcontainers AdcHwUnit

TC 2 = Subcontainers AdcPowerStateConfig

TC 3 = Containers AdcConfigSet

Set_parameters_Dependency_AdcHwUnit(Subcontainers subcontainer)			
Conditions	TC1	TC2	TC3
IP Type = Subcontainers	T	T	F
IP = "AdcHwUnit"	T	F	F
IP contains certain parameters	T	F	F
Actions			
Error Message	F	F	T
Normal flow before setting Initial vals	T	T	F
Set param_dep Initial vals	T	F	F

Set_parameters_Dependency_AdcChannel(Subcontainers subcontainer)

TC 1 = Subcontainers AdcChannel

TC 2 = Subcontainers AdcPowerStateConfig

TC 3 = Containers AdcConfigSet

Set_parameters_Dependency_AdcChannel(Subcontainers subcontainer)			
Conditions	TC1	TC2	TC3
IP Type = Subcontainers	T	T	F
IP = "AdcChannel"	T	F	F
IP contains certain parameters	T	F	F
Actions			
Error Message	F	F	T
Normal flow before setting Initial vals	T	T	F
Set param_dep Initial vals	T	F	F

Set_parameters_Dependency_AdcGroup(Subcontainers subcontainer)

TC 1 = Subcontainers AdcGroup

TC 2 = Subcontainers AdcPowerStateConfig

TC 3 = Containers AdcConfigSet

Set_parameters_Dependency_AdcChannel(Subcontainers subcontainer)			
Conditions	TC1	TC2	TC3
IP Type = Subcontainers	T	T	F
IP = "AdcGroup"	T	F	F
IP contains certain parameters	T	F	F

Actions			
Error Message	F	F	T
Normal flow before setting Initial vals	T	T	F
Set param_dep Initial vals	T	F	F

Model class

DependecyCheck(ArrayList<Containers> containers)

TC 1 = ArrayList<Containers> containers (main containers)

TC 2 = ArrayList<Containers> containers (incorrect containers)

TC 3 = ArrayList<Subcontainers> subcontainers

TC 4 = Containers AdcConfigSet

DependecyCheck(ArrayList<Containers> containers)				
Conditions	TC1	TC2	TC3	TC4
IP Type = ArrayList<Containers>	T	T	F	F
IP.get(i) = "AdcGeneral"	T	F	F	F
IP.get(i) = "AdcConfigSet"	T	F	F	F
IP.get(i) = "AdcPublishedInformation"	T	F	F	F
Actions				
Error Message	F	F	T	T
DependecySUBChceck	T	F	F	F
check_AdcGeneral	T	F	F	F
check_AdcPublishedInformation	T	F	F	F

DependecySUBChceck(ArrayList<Containers> containers, ArrayList<Subcontainers> subcontainers)

TC1 =ArrayList<Containers> containers, ArrayList<Subcontainers> subcontainers

TC2 = ArrayList<String> containers, ArrayList<Subcontainers> subcontainers

TC3 = ArrayList<Containers> containers, ArrayList<Sing> subcontainers

TC4 = ArrayList<String> containers, ArrayList<String> subcontainers

TC5 = ArrayList<Containers>, ArrayList<Subcontainers> (Incorrect contents)

DependecySUBChceck(ArrayList<Containers> containers, ArrayList<Subcontainers> subcontainers)					
Conditions	TC1	TC2	TC3	TC4	TC5
IP_1 Type = ArrayList<Containers> IP_2 Type = ArrayList<Subcontainers>	T	F	F	F	T
IP_1.get(i) = "AdcPowerStateConfig"	T	F	F	F	F
IP_1.get(i) = "AdcHwUnit"	T	F	F	F	F
IP_1.get(i) = "AdcChannel"	T	F	F	F	F
IP_1.get(i) = "AdcGroup"	T	F	F	F	F
Actions					
Error Messages	F	T	T	T	F
check_AdcPowerStateConfig	T	F	F	F	F
check_AdcHwUnit	T	F	F	F	F

check_AdcChannel	T	F	F	F	F
check_AdcGroup	T	F	F	F	F

---

check_AdcGeneral(Containers container)

TC1 = Containers (AdcGeneral)

TC2 = Containers (!AdcGeneral)

TC3 = Subcontainers

check_AdcGeneral(Containers container)			
Conditions	TC1	TC2	TC3
IP Type = Containers	T	T	F
Check for parameters and Dependency val	T	F	F
Actions			
Error Message	F	F	T
container.getParameters()	T	T	F
Set_parameters_Dependency_AdcGeneral(container)	T	T	F
Set configuration values	T	F	F

---

check_AdcPowerStateConfig(Subcontainers subcontainer, ArrayList<Containers> Containers)

TC1 = Subcontainers subcontainer, ArrayList<Containers> (correct contents)

TC2 = ArrayList<Subcontainers> subcontainers, ArrayList<Subcontainers>

TC3 = Subcontainers subcontainer, Subcontainers subcontainer

TC4 = ArrayList<Containers> containers, Subcontainers subcontainer

TC5 = Subcontainers subcontainer, ArrayList<Containers> (Incorrect contents)

check_AdcPowerStateConfig(Subcontainers subcontainer, ArrayList<Containers> Containers)					
Conditions	TC1	TC2	TC3	Tc4	TC5
IP_1 Type = Subcontainers IP_2 Type = ArrayList<Containers>	T	F	F	F	T
Check for parameters and Dependency val	T	F	F	F	T
Actions					
Error Message	F	T	T	T	F
container.getParameters()	T	F	F	F	T
Set_parameters_Dependency_AdcPowerStateConfig	T	F	F	F	T
Set configuration values	T	F	F	F	F

---

check_AdcHwUnit(Subcontainers subcontainer)

TC1 = Subcontainers subcontainer

TC2 = Containers container

check_AdcHwUnit(Subcontainers subcontainer)		
Conditions	TC1	TC2
IP Type = Subcontainers	T	F
Actions		

Error message	F	T
Set_parameters_Dependency_AdcHwUnit	T	F

---

check_AdcChannel(Subcontainers subsubcontainer, ArrayList<Containers> Containers)

TC1 = Subcontainers subcontainer, ArrayList<Containers> (correct contents)

TC2 = ArrayList<Subcontainers> subcontainers, ArrayList<Subcontainers>

TC3 = Subcontainers subcontainer, Subcontainers subcontainer

TC4 = ArrayList<Containers> containers, Subcontainers subcontainer

TC5 = Subcontainers subcontainer, ArrayList<Containers> (Incorrect contents)

check_AdcChannel(Subcontainers subsubcontainer, ArrayList<Containers> Containers)					
Conditions	TC1	TC2	TC3	Tc4	TC5
IP_1 Type = Subcontainers IP_2 Type = ArrayList<Containers>	T	F	F	F	T
Check for parameters and Dependency val	T	F	F	F	T
Actions					
Error Message	F	T	T	T	F
container.getParameters()	T	F	F	F	T
Set_parameters_Dependency_AdcChannel	T	F	F	F	T
Set configuration values	T	F	F	F	F

---

check_AdcGroup(Subcontainers subsubcontainer, ArrayList<Containers> Containers)

TC1 = Subcontainers subcontainer, ArrayList<Containers> (correct contents)

TC2 = ArrayList<Subcontainers> subcontainers, ArrayList<Subcontainers>

TC3 = Subcontainers subcontainer, Subcontainers subcontainer

TC4 = ArrayList<Containers> containers, Subcontainers subcontainer

TC5 = Subcontainers subcontainer, ArrayList<Containers> (Incorrect contents)

check_AdcGroup(Subcontainers subsubcontainer, ArrayList<Containers> Containers)					
Conditions	TC1	TC2	TC3	Tc4	TC5
IP_1 Type = Subcontainers IP_2 Type = ArrayList<Containers>	T	F	F	F	T
Check for parameters and Dependency val	T	F	F	F	T
Actions					
Error Message	F	T	T	T	F
container.getParameters()	T	F	F	F	T
Set_parameters_Dependency_AdcGroup	T	F	F	F	T
Set configuration values	T	F	F	F	F

---

check_AdcPublishedInformation(Containers container)

TC1 = Subcontainers subcontainer

TC2 = Containers container

check_AdcPublishedInformation(Containers container)		
Conditions	TC1	TC2
IP Type = Subcontainers	F	T
Actions		
Error message	T	F
Set_parameters_Dependency_AdcPublishedInformation	F	T

5.7 Gui Testing

File Editor Class

public int highLight(JTextComponent textComp, String pattern)

TC1 = JTextComponent, String

TC2 = JTextComponent, int

TC3 = String, String

TC4 = JTextComponent, JTextComponent

public int highLight(JTextComponent textComp, String pattern)				
Conditions	TC1	TC2	TC3	TC4
IP_1 Type = JTextComponent IP_2 Type = String	T	F	F	F
Actions				
Error Messages	F	T	T	T
List.add(i)	T	F	F	F
HighLight(Start, End)	T	F	F	F
Return value	T	F	F	F

---

public void HighLightValues(JTextComponent textComp, String pattern)

TC1 = JTextComponent, String

TC2 = JTextComponent, int

TC3 = String, String

TC4 = JTextComponent, JTextComponent

public int highLight(JTextComponent textComp, String pattern)				
Conditions	TC1	TC2	TC3	TC4
IP_1 Type = JTextComponent IP_2 Type = String	T	F	F	F
Actions				
Error Messages	F	T	T	T
List.add(i)	T	F	F	F
HighLight(Start, End)	T	F	F	F
ListValues.add(i)	T	F	F	F
HighLightValue(Start, End)	T	F	F	F
Return value	T	F	F	F

---

public void actionPerformed(ActionEvent e)

TC1 = ActionEvent e = e.getActionCommand().equals("New")

TC2 = ActionEvent e = e.getActionCommand().equals("Save")

TC3 = ActionEvent e = e.getActionCommand().equals("Open")

TC4 = ActionEvent e = e.getActionCommand().equals("Print")

TC5 = ActionEvent e = e.getActionCommand().equals("Find")

TC6 = ActionEvent e = e.getActionCommand().equals("UP")

TC7 = ActionEvent e = e.getActionCommand().equals("DN")

TC8 = ActionEvent e = e.getActionCommand().equals("Find Value")

TC9 = ActionEvent e = e.getActionCommand().equals("DN.")

TC10 = ActionEvent e = e.getActionCommand().equals("UP.")

TC11 = String string

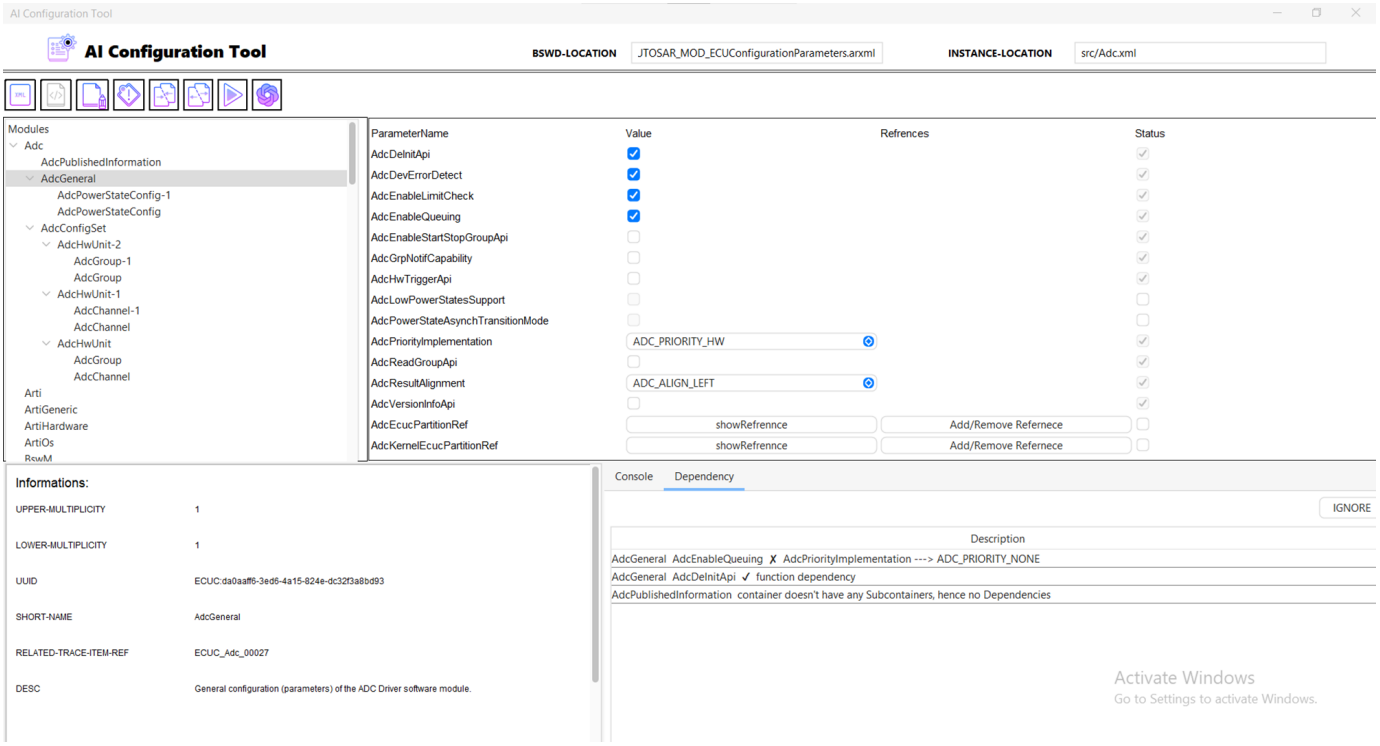
public void actionPerformed(ActionEvent e)					
Conditions	TC1	TC2	TC3	TC4	TC5
IP_1 Type = ActionEvent e	T	T	T	T	T
getActionCommand() = New	T	F	F	F	F
getActionCommand() = Save	F	T	F	F	F
getActionCommand() = Open	F	F	T	F	F
getActionCommand() = Print	F	F	F	T	F
getActionCommand() = Find	F	F	F	F	T
Actions					
HighLight(Start, End)	F	F	F	F	T

public void actionPerformed(ActionEvent e)						
Conditions	TC6	TC7	TC8	TC9	TC10	TC11
IP_1 Type = ActionEvent e	F	T	T	T	T	T
getActionCommand() = UP	F	F	F	F	F	F
getActionCommand() = DN	F	T	F	F	F	F

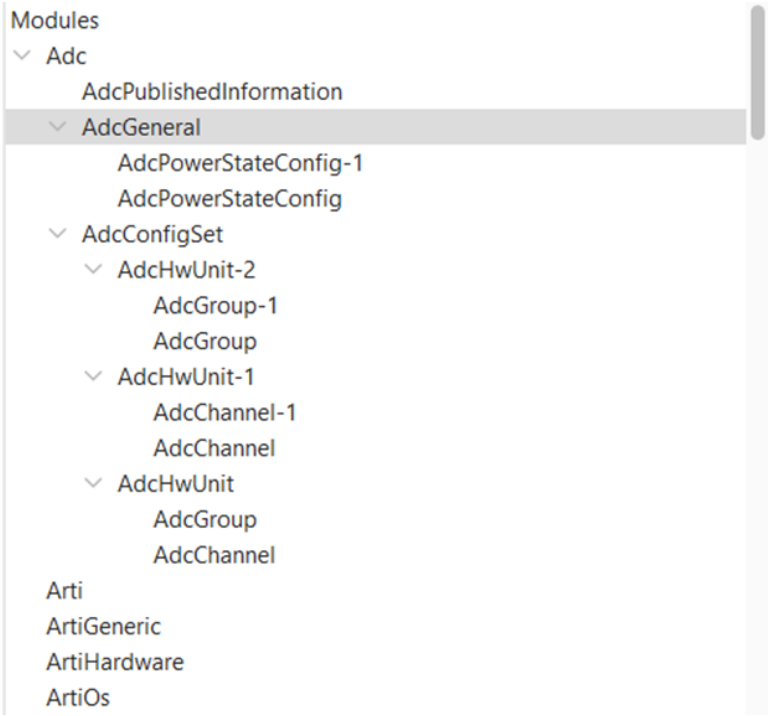
getActionCommand() = Find Value	F	F	T	F	F	F
getActionCommand() = UP.	F	F	F	T	F	F
getActionCommand() = DN.	F	F	F	F	T	F
getActionCommand() = String	F	F	F	F	F	T
Actions						
Error Message	F	F	F	F	F	T
HighLight(Start, End)	T	T	F	F	F	F
HighLightValue(Start, End)	F	F	T	T	T	F

Chapter6:Implementation

Tool GUI



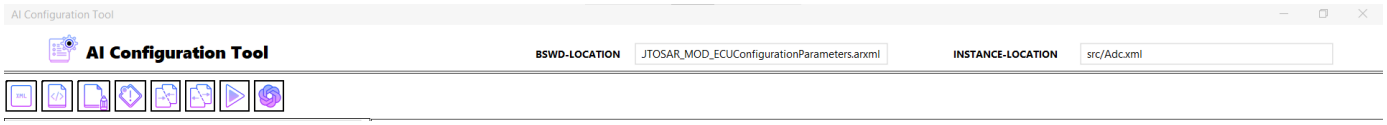
Modules-Tree View



Parameters

ParameterName	Value	Refrences	Status
AdcDeInitApi	☑		☑
AdcDevErrorDetect	☑		☑
AdcEnableLimitCheck	☑		☑
AdcEnableQueuing	☑		☑
AdcEnableStartStopGroupApi	☐		☑
AdcGrpNotifCapability	☐		☑
AdcHwTriggerApi	☐		☑
AdcLowPowerStatesSupport	☐		☐
AdcPowerStateAsynchTransitionMode	☐		☐
AdcPriorityImplementation	ADC_PRIORITY_HW		☑
AdcReadGroupApi	☐		☑
AdcResultAlignment	ADC_ALIGN_LEFT		☑
AdcVersionInfoApi	☐		☑
AdcEcucPartitionRef	showRefrennce	Add/Remove Refernece	☐
AdcKernelEcucPartitionRef	showRefrennce	Add/Remove Refernece	☐

Toolbar



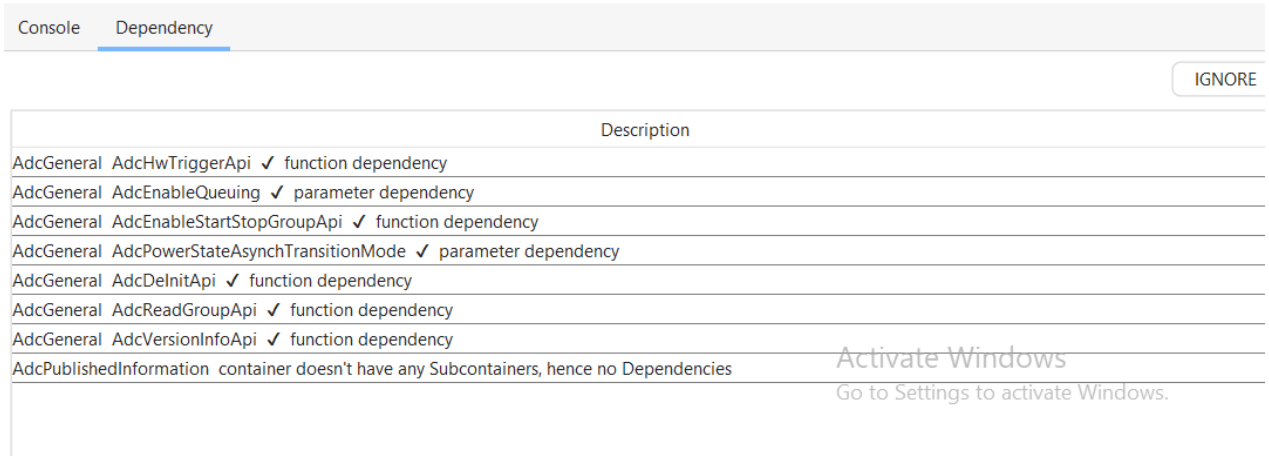
XML File Creation

Code Generation

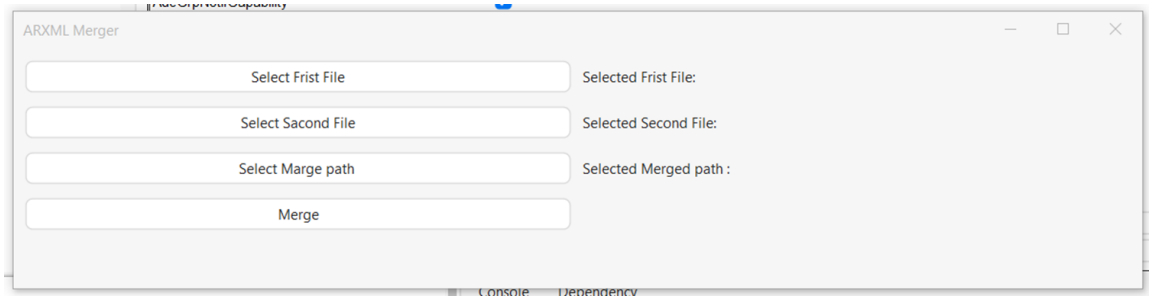
File Editor


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<AR-PACKAGE>
    <SHORT-NAME>AutosarConfigurator</SHORT-NAME>
    <ELEMENTS>
        <ECUC-MODULE-CONFIGURATION-VALUES Dest="ECUC-MODULE-DEF" DEST="BSW-IMPLEMENTATION">
            <SHORT-NAME>Adc</SHORT-NAME>
            <DEFINITION-REF>AUTOSAR/EcucDefs/Adc</DEFINITION-REF>
            <IMPLEMENTATION-CONFIG-VARIANT>VARIANT-POST-BUILD</IMPLEMENTATION-CONFIG-VARIANT>
            <MODULE-DESCRIPTION-REF>AUTOSAR/EcucDefs/Adc</MODULE-DESCRIPTION-REF>
            <CONTAINERS>
                <ECUC-CONTAINER-VALUE>
                    <SHORT-NAME>AdcConfigSet</SHORT-NAME>
                    <DEFINITION-REF>AUTOSAR/EcucDefs/Adc/AdcConfigSet</DEFINITION-REF>
                    <SUB-CONTAINERS>
                        <ECUC-CONTAINER-VALUE>
                            <SHORT-NAME>AdcHwUnit</SHORT-NAME>
```

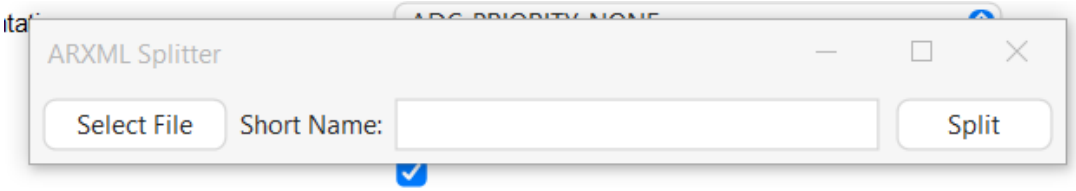
Dependencies



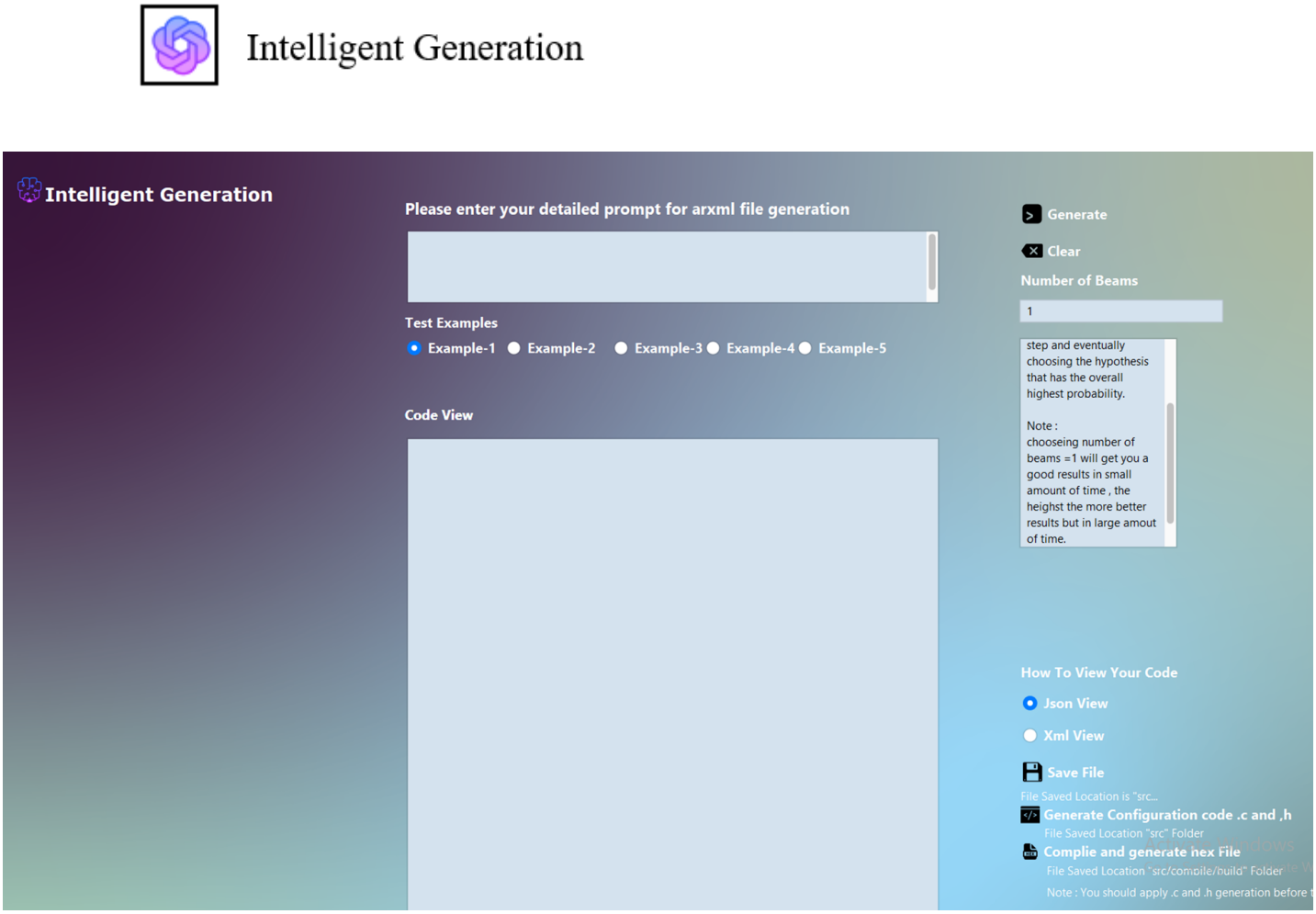
XML Merger



XML Splitter



Compiler


Intelligent Generation
Please enter your detailed prompt for arxml file generation
Generate
Clear
Number of Beams
1
Test Examples
Example-1
Example-2
Example-3
Example-4
Example-5
step and eventually choosing the hypothesis that has the overall highest probability.
Note :
chooseing number of beams =1 will get you a good results in small amount of time , the heighst the more better results but in large amout of time.
Code View
How To View Your Code
Json View
Xml View
Save File
File Saved Location is "src...
Generate Configuration code .c and ,h
File Saved Location "src" Folder
Complie and generate hex File
File Saved Location "src/compile/build" Folder
Note : You should apply .c and .h generation before

References

- Staron, M., Staron, M. and Durisic, D., 2017. Autosar standard. *Automotive Software Architectures: An Introduction*, pp.81-116.
- *ARXML serialization rules - autosar*. Available at: https://www.autosar.org/fileadmin/standards/R20-11/FO/AUTOSAR_TPS_ARXMLSerializationRules.pdf (Accessed: 14 July 2023).
- Autosar (2021a) *Requirements on Runtime Environmen*, *Autosar*. Available at: https://www.autosar.org/fileadmin/standards/R21-11/CP/AUTOSAR_SRS_RTE.pdf.
- Autosar (no date) *General specification of BASIC software modules - autosar*, *General Specification of Basic Software Modules*. Available at: https://www.autosar.org/fileadmin/standards/R21-11/CP/AUTOSAR_SWS_BSWGeneral.pdf (Accessed: 15 July 2023).
- Schulman, John; Wolski, Filip; Dhariwal, Prafulla; Radford, Alec; Klimov, Oleg (2017). "Proximal Policy Optimization Algorithms". arXiv:1707.06347.
- "Proximal Policy Optimization". *OpenAI*. 2017 paper
- GPT-3.5 with Browsing (ALPHA) Now Available for GPT Plus Users". *OPEN AI*. April 27, 2023. Archived from the original on March 20, 2023. Retrieved April 27, 2023.
- "Microsoft Updates Windows, Azure Tools with an Eye on The Future". *PCMag UK*. May 22, 2020.
- "microsoft/DeepSpeed". July 10, 2020 – via GitHub.