GALICIA

February 2025

Lorenzo Vandoni (HAL Service)

KPI4 – Platform logical architecture and requirements

Funded by the European Union

NGI SARGASSO

Funded by the **European Union**. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them. Funded within the framework of the **NGI Sargasso** project under grant agreement No **101092887**

Project Name: GALICIA
Report Title: KPI4 – Platform logical architecture and requirements
Authors: Lorenzo Vandoni (HAL Service)
Revised by: Alberto Stefanini (Novareckon)
Date: 26/02/2025
Version: 1
Distribution: Public Report

TABLE OF CONTENTS

1 APPLICATION OVERVIEW

The key components of the application will be:

- A user-friendly User Interface (UI) (Windows or web), to let users input functional requirements in their natural language. This interface would also display verification results, offering both graphical and textual feedback to users.
- A server application acting as an intermediary between the user interface and the LLM. It will process user inputs and will forward them to the LLM for code generation. It will also perform initial checks, handle data exchange, and store generated code and related verification results.
- An integration with an LLM that generates source code from natural language inputs provided by users. The server sends user requirements to LLM and receives the corresponding code, which is then presented back to the user. The LLM also interacts with a formal verification tool (e.g., NuSMV) to ensure that the generated code meets security and functionality standards.

After code generation, a compliance verification step ensures the code's alignment with specified security standards. This step uses tools such as NuSMV or PyModel, verifying that the code complies with a formal model of user requirements.

This is a description of the application workflow:

1. The user enters natural language functional requirements through the Windows or web interface.
2. The server forwards these inputs to the LLM, which generates the appropriate source code and a formal model of the generated code. Distinct LLMs could be used for the two different tasks
3. The server applies formal verification tools to assess the generated code's correctness and security compliance.
4. In case the generated code does not pass the formal verification, the server will use again the LLM to generate a new version of the code, after having informed it of the formal errors to be corrected
5. After the code has been verified to be correct, i.e. after one or more iterations of steps 2 to 4, the system provides feedback to the user, including whether the code passes or fails verification tests, and may offer suggested corrections.

2 APPLICATION FEATURES

This is the list of features provided by the application:

- Users can input functional requirements in natural language through a user-friendly interface.

- The application communicates with a backend LLM to generate source code based on user inputs
- The application communicates with a backend (possibly different) LLM to generate a formal model of the source code using formal verification tools like NuSMV or PyModel
- The generated code is automatically checked against the formal model.
- In cases where the generated code fails verification, the system iteratively refines the code based on the formal errors detected, using the LLM to generate a corrected version.
- Users receive detailed feedback, both graphically and in text, on the correctness and compliance of the generated code. This includes information on the source code and formal model generated at each iteration, the error notifications and the final result.
- Users can opt not to receive any feedback and just have the final code in return.
- Users can customize the maximum number of iterations. In case the source code does not match the requirements of the formal model after this number of iterations, the users receive information about this, and can decide to use the code anyway, or refine their initial requirements

3 DELIVERY

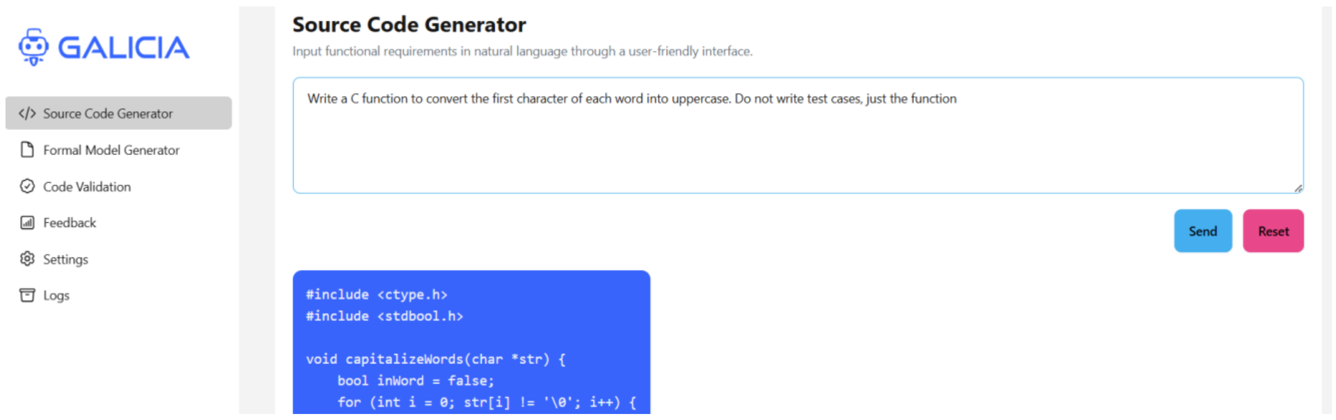
The software could be released in three different ways:

1. integrated within a development environment, i.e. for example as a Visual Studio Code Plugin
2. as a specialized GPT, i.e. for example made available among the custom GPTs available in OpenAI ChatGPT when you press the “Explore GPTs” button.
3. as a custom application, to be installed as a Windows Application

In this implementation, option nr. 3 is considered. The implementation, however, will be made by creating a layered software architecture that will ease the creation of the other types of applications, by reusing the common parts.

4 COMPONENT DESCRIPTION

The platform front-end is a user-friendly web User Interface (UI) that lets users input functional requirements in natural language. After writing the request, the UI displays a first version of the requested source code:



Source Code Generator

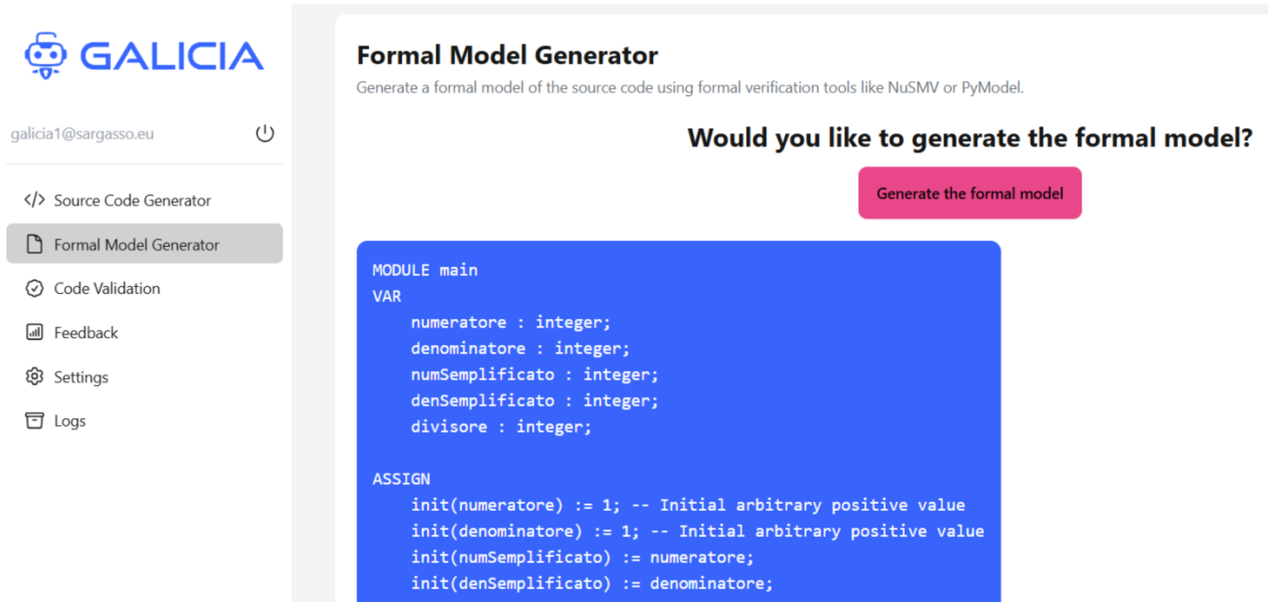
Input functional requirements in natural language through a user-friendly interface.

Write a C function to convert the first character of each word into uppercase. Do not write test cases, just the function

```
#include <ctype.h>
#include <stdbool.h>

void capitalizeWords(char *str) {
    bool inWord = false;
    for (int i = 0; str[i] != '\0'; i++) {
```

The UI also allows the user to generate the corresponding formal model. The order in which these two outputs, i.e. source code and formal model, are created can be customized by the user.



Formal Model Generator

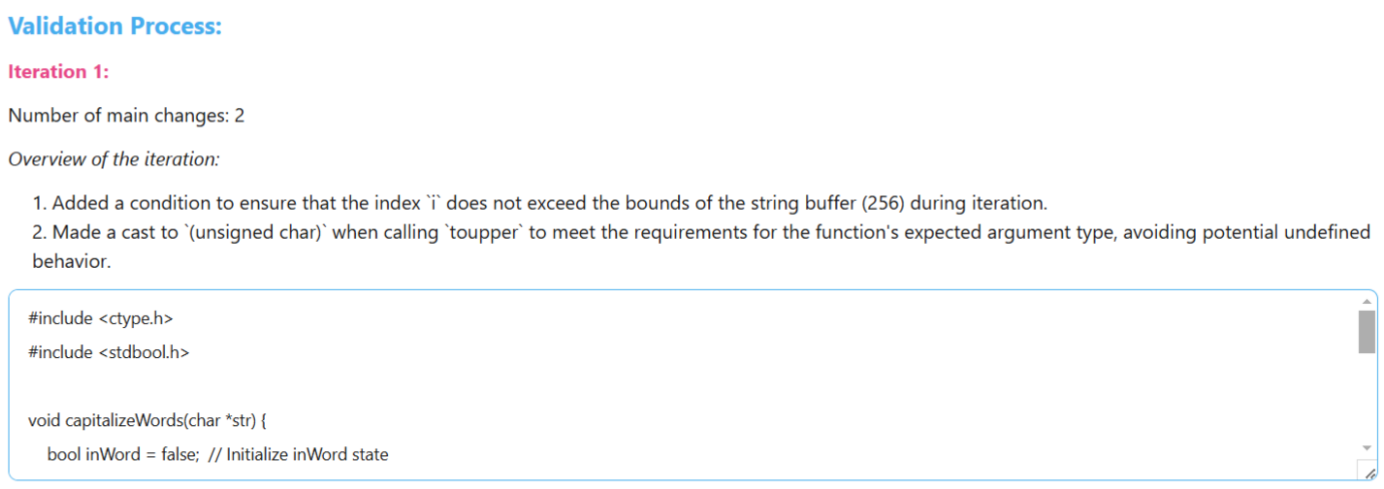
Generate a formal model of the source code using formal verification tools like NuSMV or PyModel.

Would you like to generate the formal model?

```
MODULE main
VAR
    numeratore : integer;
    denominatore : integer;
    numSemplificato : integer;
    denSemplificato : integer;
    divisore : integer;

ASSIGN
    init(numeratore) := 1; -- Initial arbitrary positive value
    init(denominatore) := 1; -- Initial arbitrary positive value
    init(numSemplificato) := numeratore;
    init(denSemplificato) := denominatore;
```

This interface also displays verification results, offering both graphical and textual feedback to users. This feedback shows how many iterations have been needed before obtaining a correct result, and which type of modifications have been made in each iteration:



Validation Process:

Iteration 1:

Number of main changes: 2

Overview of the iteration:

1. Added a condition to ensure that the index `i` does not exceed the bounds of the string buffer (256) during iteration.
2. Made a cast to `(unsigned char)` when calling `toupper` to meet the requirements for the function's expected argument type, avoiding potential undefined behavior.

```
#include <ctype.h>
#include <stdbool.h>


void capitalizeWords(char *str) {
    bool inWord = false;  // Initialize inWord state
```

A server application acts as an intermediary between the user interface and the LLM. This server application manages user authentication, processes user inputs, stores all information about the whole process and forwards individual requeststo the LLM for code generation, formal model generation and source code refinement in successive iterations.

The server applications invokes a LLM that generates source code and formal models from natural language inputs provided by users. Currently Galicia supports two different LLMs, that are LLama and OpenAI's GPT4, and allows the user to use both at the same time, e.g. one for generating source code and a different one for verifying it against the formal model.

The Galicia platform architecture has been built after performing a selection of the necessary software components, that have been identified as follows;

- User interface

  The Frontend User Interface (UI) developed using a HTML/JavaScript Framework delivers a responsive, user-friendly interface for both desktop and web environments. It facilitates user input (e.g., natural language descriptions, file uploads) and displays results, including source code, verification reports, and iteration statistics. It also provides customization options, such as modifying security requirements or adjusting verification parameters.

- Mind In A Box AI+

  Mind in a Box AI+ Integrated Ecosystem acts as the primary platform interfacing with open source language models such as LLaMA. It provides a standard API for integrating AI capabilities, ensuring flexibility in model selection and updates. It connects to one or more large language models (LLMs) for generating source code based on user-provided descriptions, translating descriptions and generated code into formal models, and iteratively refining code to address verification errors.

- LLama and GPT 4o

  Besides the open source LLM LLama, integrated via Mind in a Box AI+, Galicia implements a direct access to OpenAI's GPT-4o model, to let users test Galicia with different generative artificial intelligence models, and let them choose the best option for their specific tasks

- Python Interpreter

  Even if not part of the prototype release of the Galicia platform, Python has been extensively used to perform prompt engineering tasks, thanks to its direct interface with both the LLMs mentioned. The prompts, once tuned, have been then integrated into the PHP/Laravel back end

- PHP/Laravel back end

  A Backend Server, developed with a PHP/laravel codebase, handles core application logic, including routing user inputs, managing iterations during code generation and verification, and storing interaction history. It integrates with tools like NuSMV or PyModel for validating generated code against predefined formal models.

PROJECT CONSORTIUM:

NOVARECKON
knowledge brokers

MIND IN A BOX™

HAL
service

Funded by
the European Union

NGI SARGASSO

Funded by the **European Union**. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them. Funded within the framework of the **NGI Sargasso** project under grant agreement No **101092887**