

D3.4: Second Report on Integrated Development Environment

Revision: v.0.1

Work package	WP 3
Task	Task 3.3
Due date	2025-02-28
Submission date	2025-02-28
Deliverable lead	Carlos Coutinho (CMS)
Version	1
Authors	Carlos Coutinho (CMS), André Santos (CMS), Carlos Reis (CMS), Afonso Esteves (CMS), Miroslav Popovic (UNS), Pavle Vasiljevic (UNS), Alceste Scalas (DTU), Sotiris Spantideas (NKUA), Luís Pisco (EDP), Manuel Pio Silva (EDP), João Costa Seco (NOVA), Diogo Jesus (NOVA), Cláudia Soares (NOVA), Dimitra Tsigkari (TID), Milos Simić (UNS), Ivan Prokić (UNS)
Reviewers	Ping Hou (UOX), Carla Ferreira (NOVA)
Abstract	This document is the second iteration of the report which provides a comprehensive assessment of the TaRDIS IDE platform, highlighting its suitability for developing the TaRDIS toolbox. Through detailed analysis, it evaluates the requirements, customisation and integration activities to build the most suitable IDE for supporting the development of swarms using TaRDIS.
Keywords	Integrated Development Environment

Document Revision History

Version	Date	Description of change	List of contributors
V0.1	2024-11-15	Document first draft	Carlos Coutinho (CMS), André Santos (CMS)
V0.2	2025-02-20	First version for internal review	Carlos Coutinho (CMS)

V1	2025-02-28	Final report	Carlos Coutinho (CMS)
----	------------	--------------	-----------------------

DISCLAIMER



Funded by the European Union

Funded by the European Union (TARDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

© 2023 - 2025 TaRDIS Consortium

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	DEM + R	
Dissemination Level		
PU	Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)	✓
SEN	Sensitive, limited under the conditions of the Grant Agreement	
Classified R-UE/ EU-R	EU RESTRICTED under the Commission Decision No2015/444	
Classified C-UE/ EU-C	EU CONFIDENTIAL under the Commission Decision No2015/444	
Classified S-UE/ EU-S	EU SECRET under the Commission Decision No2015/444	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

The TaRDIS project aims at building a distributed programming toolbox to simplify the development of decentralised, heterogeneous swarm applications deployed in diverse settings.

The main contribution of this deliverable is the second revision of the TaRDIS IDE platform, the core tool of the proposed development environment to foster the creation of applications that conform to the TaRDIS programming model. The TaRDIS IDE not only provides developers with a source code editor, build automation tools, and debuggers, but also serves as a seamless environment that integrates the TaRDIS tools. It offers centralised access to the TaRDIS toolbox and multiple support features to simplify tool configuration and integration into projects that share a common development environment.

This deliverable builds upon D3.2 [1] (first version of the TaRDIS IDE) and documents the developments and improvements since the submission of D3.2.

This document aims to demonstrate the IDE's suitability for developing the TaRDIS toolbox. It reports on the IDE's development using Visual Studio Code (VS Code) and documents its integration with the tools being developed in the project's WP3, WP4, WP5, and WP6. In some cases, it describes completed integration activities, while in others, it focuses on user interface requirements and integration needs.

The document also intends to perform a first iteration over the concept of “developer stories”, i.e., describing how a developer intending to create a swarm environment would face the challenge of using the TaRDIS toolbox. It includes stories regarding the usage of isolated tools and libraries, and the “use-case” stories of the TaRDIS pilots, describing how the use-cases are being developed considering the usage of the TaRDIS toolbox. This is an essential contribution towards helping new adopters of these paradigms to embrace development strategies, guidelines and other support that may be provided by the project experts on the field.

This deliverable is constituted by a Demonstrator (source code available openly on CodeLab: <https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/vscode/vscode-ide-integration/-/tree/tardis-development> and also available as a standalone package on the Zenodo repository <https://doi.org/10.5281/zenodo.14975047>), and by a related report to describe the activities performed for the development and customisation of the IDE and the integration activities with the TaRDIS toolbox.

TABLE OF CONTENTS

Executive Summary	3
1 IDE Analysis.....	8
2 Integration of VS Code with the TaRDIS Toolbox.....	10
2.1 How the Extension Was Built	10
2.1.1 Development Objectives.....	11
2.1.2 Tools and Frameworks.....	11
2.2 Implementation Steps	12
2.3 Key Features of the Extension.....	14
2.4 Tools Required for Operation.....	16
2.5 Instruction Manual for the TaRDIS Extension	17
2.5.1 Overview.....	17
2.5.2 Installation	17
2.5.3 Getting Started	18
3 TaRDIS Toolbox Integration	23
3.1 T-WP3-01 WorkflowEditor	23
3.2 T-WP3-02 Scribble Editor	24
3.3 T-WP3-03 DCR Choreography Editor	24
3.4 T-WP4-03 JoinActors.....	27
3.5 T-WP4-06 Java Typestate Checker (JaTyC).....	27
3.6 T-WP4-07 Data Centric Concurrency (AtomiS).....	27
3.7 T-WP4-08 Anticipation of Method Execution in Mixed Consistency Systems (Ant).....	28
3.8 T-WP4-09 Correct Replicated Data Types (VeriFx).....	28
3.9 T-WP4-10 IFChannel	28
3.10 T-WP4-11 PSPSP.....	28
3.11 T-WP4-12 CryptoChoreo	28
3.12 T-WP4-13 (Sec)ReGraDa-IFC and DCR Choreographies.....	29
3.13 T-WP5-01 Flower-based FL model training.....	29
3.14 T-WP5-02 Data preparation for Flower-based FL model training.....	29
3.15 T-WP5-03 Flower-based FL model inference and evaluation.....	29
3.16 T-WP5-04 PTB-FLA and MPT-FLA.....	30
3.17 T-WP5-05 Federated AI network orchestrator (FAUNO).....	32
3.18 T-WP5-08 Pruning	32
3.19 T-WP5-09 Decentralised Federated Learning Framework (Fedra)	33
3.20 T-WP5-10 FLaaS	33
3.21 T-WP5-11 Simulator for Peer-to-Peer Networks.....	34

- 3.22 T-WP6-01 A Generic API for Decentralised Overlay and Communication Protocols 34
- 3.23 T-WP6-02 An Epidemic and Scalable Global Membership Service 34
- 3.24 T-WP6-03 Actyx: Reliable event broadcast with configurable durability 35
- 3.25 T-WP6-04 Babel 35
- 3.26 T-WP6-05 Arboreal: Extending Data management from Cloud to Edge leveraging Dynamic Replication..... 38
- 3.27 T-WP6-06 PotionDB: Strong Eventual Consistency under Partial Replication 39
- 3.28 T-WP6-07 Integration of Storage Solutions into the TaRDIS Ecosystem..... 39
- 3.29 T-WP6-08 Distributed Management of Configuration based on Namespaces 39
- 3.30 T-WP6-09/10 Telemetry Acquisition for Decentralised Systems 39
- 4 TaRDIS Developer Stories 41
 - 4.1 Develop a Babel swarm project 41
 - 4.1.1 Initialization..... 42
 - 4.1.2 Protocol Specification..... 42
 - 4.1.3 Practical Examples..... 43
 - 4.1.4 IDE Integration 43
 - 4.2 Develop a PTB-FLA project 43
 - 4.3 Develop a PTB-FLA with integration with Babel..... 45
 - 4.4 Develop a DCR Choreography 45
 - 4.5 Complex Swarm: Common workspace for Multiple projects..... 46
- 5 TaRDIS Pilot (Use Case) Developer Stories..... 48
 - 5.1 EDP NEW Energy: Multi-Level Grid Balancing 48
 - 5.1.1 Using the Fedra tool 48
 - 5.1.2 Pruning tool 51
 - 5.1.3 Membership Service for Energy Markets 52
 - 5.1.4 Community Energy Balancing Application..... 52
 - 5.2 GMV: Distributed navigation concepts for LEO satellites constellations 53
 - 5.2.1 Machine Learning process 53
 - 5.2.2 Decentralized Storage and Communication Between Satellites..... 53
 - 5.3 ACT: Highly resilient factory shop floor digitalisation 54
 - 5.4 TID: Privacy-Preserving Learning Through Decentralized Training in Smart Homes 55
- 6 Conclusions..... 57

LIST OF FIGURES

Figure 1: File package.json defining commands.....	13
Figure 2: Create webview panel	13
Figure 3: TaRDIS Project Creation	14
Figure 4: TaRDIS IDE documentation viewer	15
Figure 5: The TaRDIS VS Code extension.....	18
Figure 6: Babel feature for Create Class	20
Figure 7: Babel importing of a protocol – before the importing.....	21
Figure 8: Babel importing of a protocol – after the importing	21
Figure 9: Screenshot by Caroline Bjørch Fallesen - from the MSc thesis “A Visual Studio Code Extension for Editing Swarm Protocols” (DTU, 2025). Available at: https://findit.dtu.dk/en/catalog/679d746fabd7f915d2a6809d	24
Figure 10: DCR Choreography integrated in the TaRDIS IDE.....	25
Figure 11: IDE buttons for compiling a DCR Choreography project.....	25
Figure 12: Run a DCR Choreography form.....	26
Figure 13: Building the environment for a DCR Choreography	26
Figure 14: Configuration of a PTB-FLA Project.....	31
Figure 15: PTB-FLA project considering less experienced developers’ support	31
Figure 16: TaRDIS PTB-FLA form for selecting Federated Learning algorithms	32
Figure 17: TaRDIS IDE integration with Babel	36
Figure 18: Babel selection of communication protocols.....	37
Figure 19: Babel form for confirmation of inserting a protocol	37
Figure 20: Babel form for creation of a class.....	38
Figure 21: Developer Story over the development of an application using Babel.....	41
Figure 22: Example of Babel definitions for a swarm node.....	42
Figure 23: Representation of a smart home for the EDP NEW pilot.....	48
Figure 24: Configuration of the Fedra tool for the EDP NEW pilot	49
Figure 25: Fedra training process initialisation for the EDP NEW pilot.....	49
Figure 26: Initiation of the first FL round for the EDP NEW pilot.....	50
Figure 27: Fedra metrics for the EDP NEW pilot.....	50
Figure 28: The Pruning tool configuration for the EDP NEW pilot	51
Figure 29: The TaRDIS Pruning tool metrics.....	51
Figure 30: Accuracy of the Pruning activity	51
Figure 31: EDP NEW Community energy balancing application	53
Figure 32: The TaRDIS Nimbus storage model on the GMV pilot.....	54
Figure 33: Administrator Interface for FLaaS in the TID pilot.....	56

ABBREVIATIONS

ACT	Actyx
AI	Artificial Intelligence
API	Application Programming Interfaces
DCR	Dynamic Condition Response
DNN	Deep Neural Network
FAUNO	Federated AI Network Orchestrator
FL	Federated Learning
FLA	FL Algorithm
FLaaS	FL as a Service
IDE	Integrated Development Environment
IFC	Information Flow Control
JAR	Java Archive
LEO	Low Earth Orbit
LSTM	Long-Short-Term-Memory
MARL	Multi-Agent Reinforcement Learning
ML	Machine Learning
MPST	MultiParty Session Type
MPT-FLA	MicroPython Testbed for Federated Learning Algorithms
MSE	Mean-Squared Error
ODTS	Orbit Determination and Time Synchronisation
P2P	Peer-to-Peer
PTB-FLA	Python TestBed for Federated Learning Algorithms
RL	Reinforcement Learning
SL	Split Learning
TID	Telefonica
UI	User Interface

1 IDE ANALYSIS

Integrated Development Environments (IDEs) are critical tools for software development, offering a variety of features that enhance productivity, streamline workflows, and improve the quality of code.

Using an Integrated Development Environment (IDE) for software development offers numerous advantages beyond simply providing a code editor. It includes features such as intelligent code completion, which helps developers write code faster and with fewer errors by suggesting possible completions as they type. IDEs also support project and file management, helping developers organize files, manage dependencies, and streamline the build process. Many IDEs integrate seamlessly with version control systems like Git, making it easier to track changes, collaborate, and maintain project history. Additional features often include syntax highlighting, which improves code readability by visually distinguishing keywords, variables, and other elements, as well as refactoring tools that facilitate restructuring operations such as renaming variables or extracting methods, enhancing code quality and maintainability.

Modern IDEs also feature advanced integration topics such as the possibility to have their functionalities extended by plugins and extensions tailored to the developer's specific needs, such as support for different programming languages or frameworks, or supporting integrated testing frameworks, and often include built-in documentation and help resources, making it easier to learn new APIs and libraries.

The definition of an IDE that fosters and promotes the development of distributed and decentralised SWARM applications needs to be carefully planned. It must take into account the needs of the businesses being served by swarm applications, expressed on the requirements of the existing project pilots, but additionally the foreseen developer experience, and finally the initial vision of the tools being developed in the other WPs of the TaRDIS project that are foreseen to be integrated in the TaRDIS toolbox.

The initial development of the TaRDIS Integrated Development Environment (IDE) was based on the best-of-breed commercial tool Eclipse, due to the large base experience that the project team had with this platform. Over the development of the project, it became more evident that although the Eclipse environment is still highly configurable and suitable for the development purposes, there are other platforms that are becoming more notorious. Namely, the Microsoft Visual Studio Code (VS Code) IDE currently dominates the IDE market with its lightweight design, versatility, and extensive library of extensions. Backed by Microsoft, VS Code became very popular and known for supporting almost every programming language, making it a favourite among full-stack developers. This led the development team already on D3.2 [1] to make a parallel development effort to start working in the VS Code platform for supporting TaRDIS. As the experience became more consistent, the team moved the integration efforts from Eclipse to VS Code, confident that this change will increase the success of the adoption of this IDE with the development community.

Microsoft Visual Studio Code

Visual Studio Code (VS Code) is a powerful and highly customizable code editor developed by Microsoft, quickly becoming one of the most popular tools in the developer community. Known for its speed, flexibility, and user-friendly interface, VS Code is an excellent choice for developers of all skill levels, whether the developers are working on small projects or large-scale software development [2].

One of VS Code's key features is its extensive support for programming languages. It natively supports JavaScript, TypeScript, Python, C++, Java, and many others. Additionally, the editor allows developers to easily install extensions to add language support for virtually any framework or tool in the market. The robust extension marketplace enables developers to enhance their workflows with features like syntax highlighting, code snippets, linting, and formatting, making it a go-to choice for multi-language development [3].

VS Code is built for productivity, offering features that significantly improve the coding experience. The built-in IntelliSense feature provides smart code completion based on variable types, function definitions, and imported modules. This feature speeds up development and helps reduce errors by suggesting code as it's being typed, ensuring developers are always on the right track. For those who prefer a minimalist approach, VS Code can also be heavily customized with themes, icon sets, and editor settings, allowing a full personalization of the interface.

Another standout feature is its integrated version control, which makes it easier to manage code repositories directly within the editor. VS Code supports Git [4] out of the box, enabling developers to stage, commit, and push changes to repositories without leaving the editor. Developers can also view diffs, resolve merge conflicts, and track the status of files in a highly intuitive way, all without opening a separate Git client.

Debugging in VS Code is intuitive and efficient. With built-in support for debugging various languages, developers can set breakpoints, inspect variables, step through code, and analyse the call stack directly within the editor. This saves time by allowing you to debug and test your code in one place, reducing the need to switch between different tools.

VS Code's terminal is another powerful feature, offering an integrated command-line interface that allows developers to run scripts, execute commands, and manage their development environment without leaving the editor. This makes it easy to compile code, run tests, or interact with cloud services, all within the same window.

The editor's workspace functionality allows managing multiple projects in a single instance, keeping the workspace organised, and enabling seamless switching between different files or projects. Developers can also split the editor into multiple panes, to view and edit multiple files side by side.

For developers working in teams, VS Code's Live Share feature is a game-changer. It enables real-time collaboration, allowing multiple developers to share their coding session with others, collaborate on code, debug together, and even run applications remotely—all without leaving the editor.

2 INTEGRATION OF VS CODE WITH THE TARDIS TOOLBOX

VS Code started by being promoted as a text editor, which rapidly evolved to a full-fledged IDE. Its main feature is the support of a popular and rich marketplace of extensions and add-ons that confer to this IDE a very strong and robust functionality. Microsoft's influence led this IDE to be very popular, especially among the newest generations of developers [3].

VS Code is cross-platform, working seamlessly on Windows, macOS, and Linux. This makes it an ideal choice for teams working in diverse environments or for developers who work across different operating systems. Plus, it's free and open-source, which means anyone can contribute to its development or modify it to suit their specific needs.

Languages Support

It can support all languages, either by using an existing extension or by creating new ones.

Customization

Developers can customize the editor with extensions, snippets, themes, language support, keymaps, and notebook renderers, all of which are written in TypeScript or JavaScript.

Prerequisites

- Install Visual Studio Code.
- Install Node.
- Install globally the 'yo' and 'generator-code' packages via Node Package Management.
- Run 'yo code' and choose how to extend the editor.
- Open the created project with the editor and customize it.

Nevertheless, in most recent years, VS Code is becoming increasingly more popular among the newest generations due to its very interesting capabilities of customisation and rapidly growing rich marketplaces of extensions that allow it to be able to work in multiple and heterogeneous environments.

2.1 HOW THE EXTENSION WAS BUILT

The **TaRDIS Extension for Visual Studio Code** was developed to streamline the workflow for creating and managing swarms, whether the purpose is to define a full **TaRDIS swarm application**, as described in the TaRDIS deliverable D3.3 [5], or to define the behaviour of individual **managed** or **free-form swarm elements**. This action is performed by the development of one or multiple projects which span the technologies used in the TaRDIS project. The design process focused on integrating seamlessly with the Visual Studio Code ecosystem, automating complex configurations and offering an intuitive user experience. This section outlines the tools, processes and methodologies employed in this development.

2.1.1 DEVELOPMENT OBJECTIVES

The primary objective behind the development of the TaRDIS extension was to simplify how the developers interact with distributed systems by automating intricate configurations and providing user-friendly workflows. The extension was designed to be versatile and adapt its features to suit their specific project requirements.

To achieve this, the extension supports multiple project types, besides the traditional language development-based (e.g., Java, C++, Python or TypeScript), extended to specific project types related to the TaRDIS project, e.g., Babel-based Java projects, FAUNO and PTB-FLA projects for federated learning, or DCR choreographies for distributed execution. It ensures seamless integration with essential tools such as Maven for Java projects, Python virtual environments for PTB-FLA, and Docker for running DCR-based prosumers. This focus on usability, flexibility and cross-platform compatibility was key to creating an extension that enhances productivity while maintaining robust functionality.

2.1.2 TOOLS AND FRAMEWORKS

Node.js

Node.js was used as the runtime environment to build the extension and execute its operations. It facilitated seamless integration with the Visual Studio Code API while providing an extensive ecosystem for dependency management through npm and helped build the structure of the extension with existing npm packages like 'yo' and 'generator code'.

TypeScript

TypeScript was chosen as the primary development language for its static typing, modern JavaScript features, and strong compatibility with the Visual Studio Code API. This choice ensured the extension's reliability, maintainability and ease of debugging as the tools provided by the yo Node Package Manager facilitated the integration using this language.

Visual Studio Code API

The VS Code API enabled the extension to interact directly with the IDE, allowing for the registration of commands, dynamic file manipulation, and the rendering of interactive webviews. This API served as the backbone for the extension's core functionality. The API documentation is found in the link <https://code.visualstudio.com/api/references/VS-Code-api>.

Webview API

The Webview API played a crucial role in creating visually rich and interactive panels that allowed users to make choices. These panels were used for user inputs, project configuration, protocol documentation and DCR choreography management. This ensured the extension maintained a consistent design while delivering clarity to users.

Maven

For Java-based projects, Maven was employed to manage project dependencies and build configurations. The extension has the ability to automatically update the pom.xml file to include the required Babel libraries and repositories at the press of a button, eliminating any manual configuration steps.

Python Virtual Environment

For PTB-FLA projects, the extension created a dedicated virtual environment (**venv_ptbfla**) with pre-installed dependencies. This environment ensured compatibility with machine learning libraries, providing a ready-to-use workspace for federated learning development.

Docker

Docker was integrated into the extension for running DCR choreographies. The extension automated the creation of Docker containers, networks, and images, ensuring each prosumer instance operated in an isolated yet interconnected environment.

Markdown Renderer:

A custom markdown renderer was implemented to display documentation within webview panels. This allowed users to view formatted protocol specifications, configuration parameters, and API documentation directly within VS Code.

Yeoman:

The initial project structure was scaffolded using Yeoman's generator for VS Code extensions, providing a modular foundation for the codebase and ensuring a consistent development workflow.

2.2 IMPLEMENTATION STEPS

The development of the TaRDIS extension followed a structured approach:

1. Initializing the Extension:

The project began with the initialization of the extension using Yeoman's yo code generator. This tool provided a TypeScript-based scaffold, complete with the necessary configuration files and boilerplate code. This foundation enabled the rapid development of core functionalities while maintaining clean code organization.

2. Registering Commands:

User commands were defined in the **package.json** file, corresponding to each key feature, such as creating a project, importing protocols, compiling DCR choreographies, and visualizing swarm protocols, as depicted on Figure 1. These commands acted as entry points, allowing users to trigger specific actions through the command palette or the TaRDIS sidebar.

```
"contributes": {
  "commands": [
    {
      "command": "tardis.commands.create-project",
      "title": "TaRDIS: Create Project"
    },
    {
      "command": "tardis.commands.create-generic-class",
      "title": "TaRDIS: Create Generic Class"
    },
    {
      "command": "tardis.commands.import",
      "title": "TaRDIS: Import existing project"
    },
    {
      "command": "tardis.commands.visualize-swarm",
      "title": "TaRDIS: Visualize Swarm Protocol"
    },
    {
      "command": "tardis.commands.compileDcr",
      "title": "TaRDIS: Compile tardis DCR"
    }
  ]
},
```

Figure 1: File package.json defining commands

3. Creating Webview Panels:

The Webview API was used to build interactive panels for project creation, protocol management, and DCR choreography execution. These panels displayed user inputs, protocol documentation, and real-time status updates, providing an intuitive interface for complex workflows, as seen on Figure 2.

```
const panel = vscode.window.createWebviewPanel(
  "dcrChoreography",
  "Run DCR Choreography",
  vscode.ViewColumn.Beside,
  { enableScripts: true }
);
```

Figure 2: Create webview panel

4. Automating Project Setup:

To simplify project initialization, the extension automated the creation of folder structures, configuration files, and dependency management. For Babel projects, the pom.xml file was

dynamically updated. PTB-FLA projects were equipped with a virtual environment, while DCR projects included Docker setup scripts.

5. Dynamic Protocol Handling:

The extension implemented dynamic handling for protocol-specific operations, such as fetching dependencies, displaying configuration parameters, and managing API events. This ensured compatibility with a wide range of Babel-based protocols and DCR choreographies.

6. Integrating Docker for DCR Choreographies:

For DCR projects, the extension automated Docker-based execution. It created isolated prosumer instances, each running within its own container, while connecting them via a dedicated Docker network. This approach ensured realistic simulation environments for distributed workflows.

7. Styling and Theming:

Custom HTML and CSS were used to style webview panels, ensuring alignment with Visual Studio Code's dark and light themes. Syntax highlighting for code blocks and a clean layout further contributed to a user-friendly and professional interface.

2.3 KEY FEATURES OF THE EXTENSION

The TaRDIS extension offers a comprehensive set of features designed to enhance the development experience for distributed systems projects.

1. Multi-Project Creation:

With the current integration of tools in the IDE, developers are able to create (for the moment) three types of projects, as depicted in Figure 3:

- **Java Babel Projects:** Fully configured Maven-based projects with Babel dependencies, ideal for developing swarm applications.
- **PTB-FLA Projects:** Python-based projects for federated learning algorithms, complete with a virtual environment and predefined templates.
- **DCR Choreography Projects:** Projects designed for distributed execution, allowing users to compile and run dynamic condition-response choreographies.

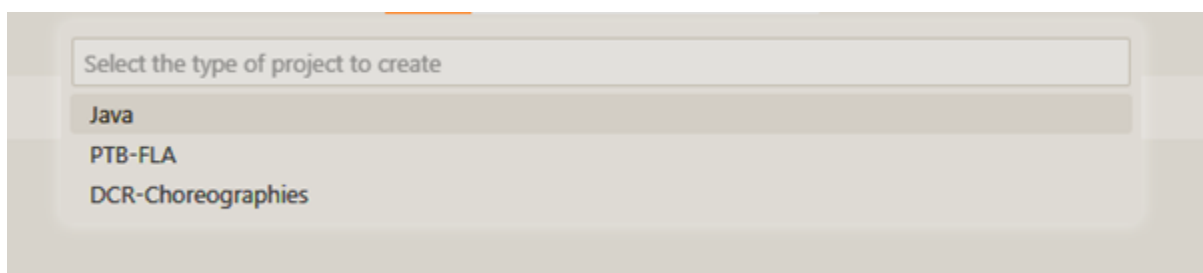


Figure 3: TaRDIS Project Creation

With the evolution of the integration of TaRDIS tools in the IDE, more types of projects are foreseen.

2. Documentation Viewer:

The extension includes a markdown-based documentation viewer, displaying detailed information for each protocol, including usage instructions, configuration parameters, and API events. This ensures users have all necessary resources to import different protocols, as documentation is always present to inform the programmer of the protocol's functionalities.

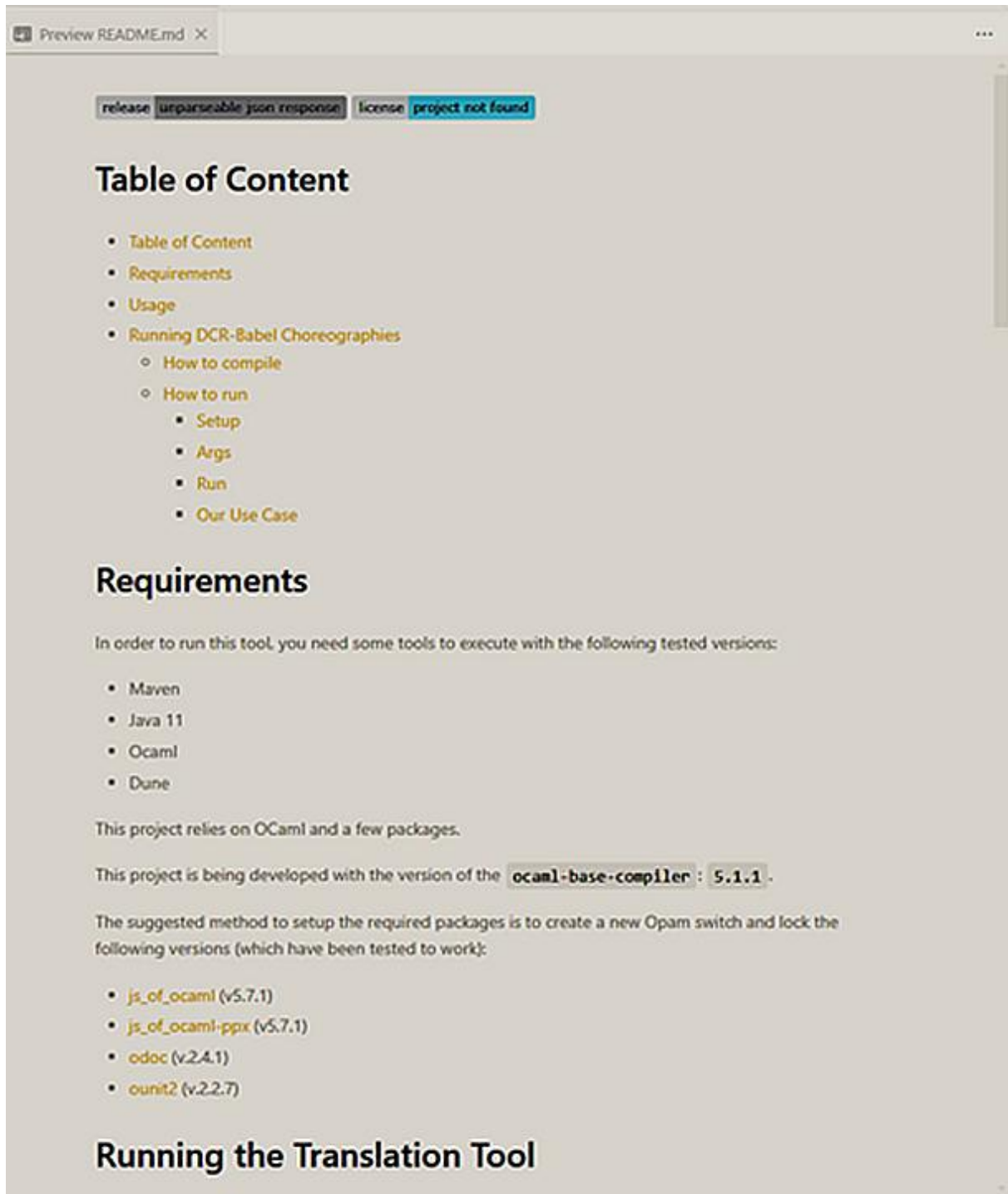


Figure 4: TaRDIS IDE documentation viewer

3. Dependency Management:

Managing dependencies is streamlined through automated functionality. For Java projects, the **pom.xml** file is updated to include both Maven dependencies and those associated with imported protocols. PTB-FLA projects are initialized with all necessary Python packages when a new project is created, while DCR projects ensure Docker and OCaml are properly configured by checking if they are installed before running any commands required for initializing the prosumers.

4. Dynamic Project Setup:

The extension offers guided workflows for project setup, providing pre-configured environments for Java, Python, and DCR projects. Users can easily switch between project types without manual setup by selecting the type of project to be added to the workspace. All projects created are added to one common workspace in order to be managed and viewed all in the same place for better development interaction.

5. Extensibility:

Built with a modular design, the extension allows for easy expansion. Developers can add new protocols, classes, and features as project requirements evolve. The buttons create class, import and visualize swarm and provide the ability to add new predefined classes and protocols to facilitate the process for the programmer, as well as add the ability to visualize the swarm in a workflow editor.

6. Integrated Docker Execution:

For DCR projects, the extension automates Docker-based execution. It creates isolated containers for each prosumer, connects them to a shared network managed by docker, and executes the choreography, providing real-time logs in the terminal.

2.4 TOOLS REQUIRED FOR OPERATION

To ensure seamless operation, the following tools are required:

- **Java Development Kit (JDK):** Java 21 is essential for Babel-based projects, providing the runtime environment for Java applications.
- **Node.js:** Serves as the runtime environment for the extension itself, managing dependencies and executing commands.
- **Maven:** Used for dependency management and build automation in Java projects.
- **Python:** Required for PTB-FLA projects, with virtual environments ensuring package isolation.
- **Docker:** Facilitates containerized execution of DCR choreographies.
- **Visual Studio Code:** Serves as the primary development environment.
- **Git:** Enables version control, code sharing, and collaborative development.

Optional tools, such as ESLint for code quality and the VS Code Debugger for real-time testing, further enhance the development experience. Together, these tools create a robust environment for building and managing distributed systems using the TaRDIS extension.

By combining project management, protocol integration, federated learning support, and distributed execution capabilities, the TaRDIS extension provides a comprehensive solution for developing modern distributed systems within Visual Studio Code. It not only simplifies complex workflows but also ensures users can focus on development without being burdened by intricate configurations.

2.5 INSTRUCTION MANUAL FOR THE TARDIS EXTENSION

2.5.1 OVERVIEW

The TaRDIS Visual Studio Code extension is a comprehensive toolbox designed to streamline the development and management of distributed systems and projects. The extension provides a range of functionalities, including project creation, protocol importing, and class generation, to help developers quickly and efficiently set up and manage their projects. This manual outlines how to use the TaRDIS extension and its current features.

2.5.2 INSTALLATION

Currently the TaRDIS IDE extension is not yet published into the Microsoft marketplace, but it will be made available soon. Nevertheless, it is possible to download it through the TaRDIS internal Git repository – named CodeLab¹ – by following the step 3 down of the installation process described below.

1. From the Visual Studio Code Marketplace:

- Open the Extensions view in VS Code (Ctrl+Shift+X).
- Search for "TaRDIS."
- Click "Install."

2. Manually via VSIX:

- Obtain the VSIX file for the TaRDIS extension.
- Open the Extensions view in VS Code (Ctrl+Shift+X).
- Click on the "... " in the top right corner of the Extensions view.
- Select "Install from VSIX..." and locate the VSIX file.

3. From debugging mode in VS Code:

- Obtain the code present in CodeLab in the link:
https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/VS_Code/VS_Code-ide-integration/-/tree/tardis-development?ref_type=heads
- Open the code on VS Code
- Run the extension by choosing Run->Start Debugging or simply press the F5 button to load the extension in debug mode.

¹ <https://codelab.fct.unl.pt/di/research/tardis>

- A new instance of VS Code will pop up in a new window, with the extension already installed.
- Keep in mind that in order to run the extension, there are prerequisites and technologies mentioned above in order to create and run Java and PTB-FLA projects or open the Workflow editor (install the extension available on link <https://marketplace.visualstudio.com/items?itemName=CarolineFallesen.visua-l-swarm-protocol-editing>), as well as OCamel (<https://ocaml.org>) and Dune (<https://github.com/tarides/dune-release>) for the DCR Choreographies compilation to work.

2.5.3 GETTING STARTED

Once installed, the TaRDIS VS Code extension can be accessed directly from the Activity Bar under the "TaRDIS" view, serving as the central hub for all its features. This interface provides an intuitive way to create new projects, import protocols, generate classes, and manage various development tasks. Users are presented with a set of clearly labelled buttons, each corresponding to a specific functionality, making it easy to navigate and utilize the extension's capabilities, as depicted in Figure 5.

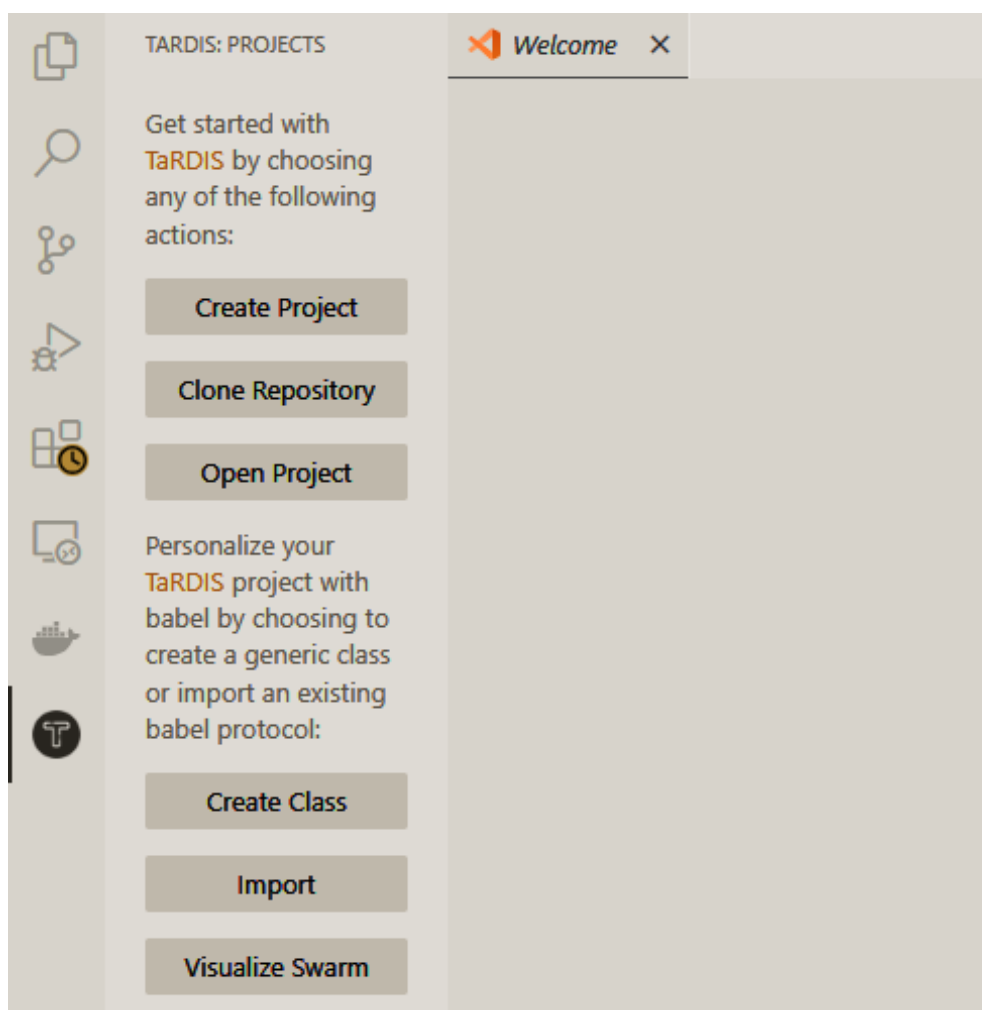


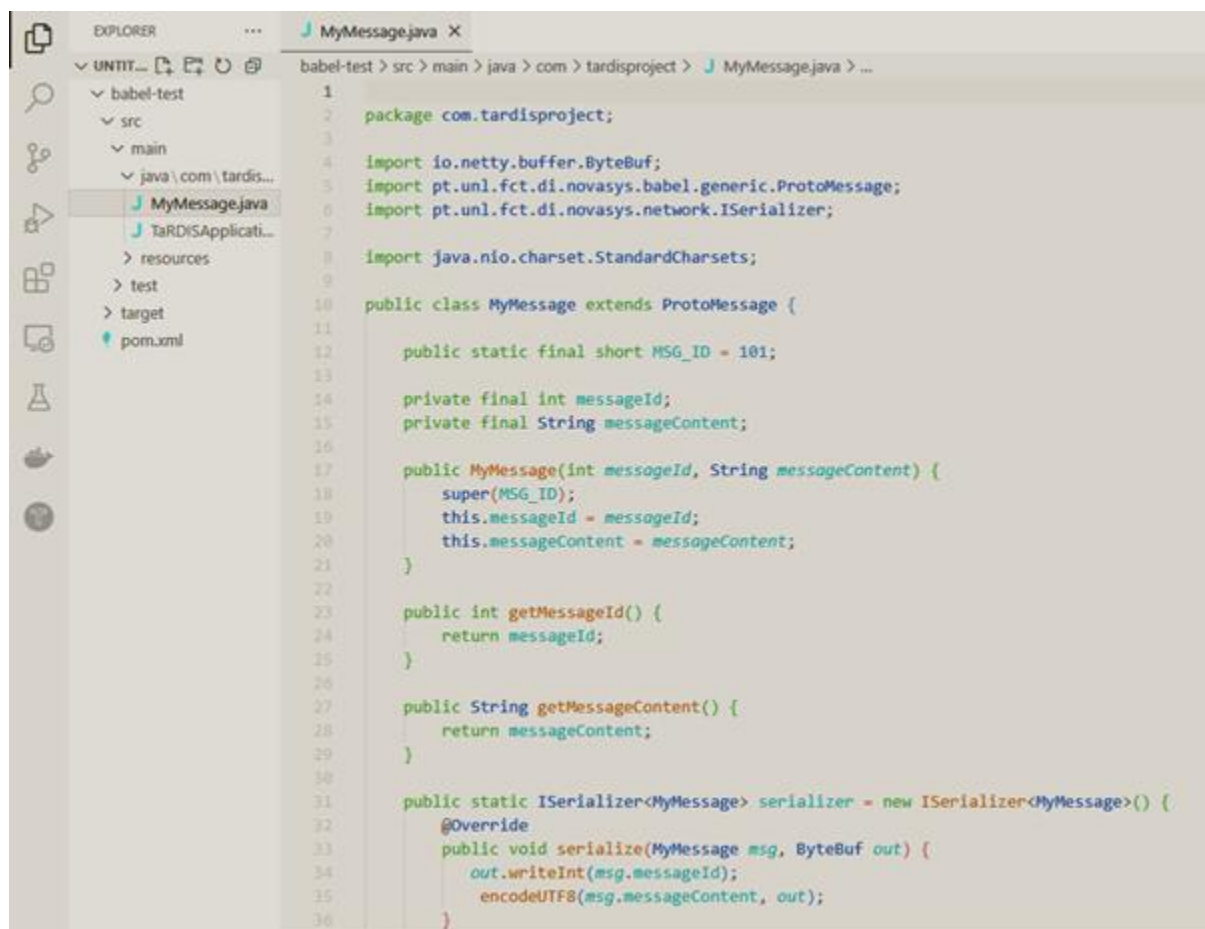
Figure 5: The TaRDIS VS Code extension

The TaRDIS extension allows users to create structured projects with predefined setups and configurations. By selecting the Create Project option, developers can initiate a new project tailored to their needs. During the setup process, they can choose between different project types, including Java-based projects utilizing Babel libraries, PTB-FLA projects designed for federated learning applications in Python, or DCR Choreographies for orchestrating distributed prosumers. After specifying a project name and selecting a destination folder, TaRDIS automatically configures the project with the necessary dependencies and structure. For Java projects, this means integrating Babel and its required configurations. PTB-FLA projects, on the other hand, include the creation of a virtual environment (`venv_ptbfla`), automatic installation of the PTB-FLA package, and generation of a dedicated source folder for implementations. DCR Choreography projects establish an execution-ready structure, enabling users to compile and run choreographies with customizable prosumer counts. Once created, the project workspace seamlessly opens in VS Code, allowing users to start coding immediately.

For developers working with external codebases or collaborating on shared repositories, TaRDIS provides a Clone Repository feature. By selecting this option, users can enter a repository URL and specify a destination folder. The extension then clones the repository, making the project available in the local workspace. This feature simplifies the integration of external projects into the development environment, ensuring that users can quickly access and modify the cloned codebase without additional setup.

TaRDIS also includes functionality for opening existing projects stored locally. Using the “Open Project” option, users can browse their file system, select a project folder, and have it loaded into the workspace. This allows developers to seamlessly switch between projects without needing to manually configure the workspace each time. By automating these processes, TaRDIS enhances productivity and provides a streamlined experience for managing multiple development workflows within VS Code.

In addition to project creation and management, TaRDIS simplifies the development of Babel-based applications by providing built-in support for generating essential components. With the Create Class feature, developers can quickly define fundamental building blocks such as messages (seen in the figure below), protocols, timers, notifications, replies, and requests, as shown in Figure 6. After selecting a class type, the user is prompted to enter a name, and the extension generates a corresponding template file within the appropriate package. This eliminates repetitive boilerplate coding and ensures consistency in the project's structure.



```
1 package com.tardisproject;
2
3
4 import io.netty.buffer.ByteBuf;
5 import pt.unl.fct.di.novasys.babel.generic.ProtoMessage;
6 import pt.unl.fct.di.novasys.network.ISerializer;
7
8 import java.nio.charset.StandardCharsets;
9
10 public class MyMessage extends ProtoMessage {
11
12     public static final short MSG_ID = 101;
13
14     private final int messageId;
15     private final String messageContent;
16
17     public MyMessage(int messageId, String messageContent) {
18         super(MSG_ID);
19         this.messageId = messageId;
20         this.messageContent = messageContent;
21     }
22
23     public int getMessageId() {
24         return messageId;
25     }
26
27     public String getMessageContent() {
28         return messageContent;
29     }
30
31     public static ISerializer<MyMessage> serializer = new ISerializer<MyMessage>() {
32         @Override
33         public void serialize(MyMessage msg, ByteBuf out) {
34             out.writeInt(msg.messageId);
35             encodeUTF8(msg.messageContent, out);
36         }
37     }
38 }
```

Figure 6: Babel feature for Create Class

For users who need to extend their projects with predefined communication patterns, the Import Protocol functionality allows for seamless integration of Babel protocols. By selecting this option, developers can browse and import communication protocols categorized by type, such as Eager Gossip Broadcast.

Once a protocol is chosen, TaRDIS automatically handles dependency management, adding the required entries to the project's configuration files and injecting initialization code into the main application. This streamlined workflow accelerates development while ensuring that imported protocols are correctly integrated and fully functional.

This process can be seen below: Figure 7 shows the `init` function before the import action of a chosen protocol. The image on the right shows us the after, with Eager Gossip protocol imported and properly initiated in the function, ready to be used.

```

J MyMessage.java    J TaRDISApplication.java X
babel-test > src > main > java > com > tardisproject > J TaRDISApplication.java > TaRDISApplication > TaRDISApplication
11  public class TaRDISApplication {
12  }
36  protected TaRDISApplication() {
37      this.babel = Babel.getInstance();
38  }
39
40  private void init(Properties properties) throws Exception {
41      // Initialize Babel
42      this.babel.start();
43      logger.info(message:"TaRDISApplication started successfully!");
44
45      // Initialize protocols
46      // MyProtocol protocol = new MyProtocol();
47      // this.babel.registerProtocol(protocol);
48
49      // Pass the properties object to the protocol for its own initialization
50      // protocol.init(properties);
51  }
52  }

```

Figure 7: Babel importing of a protocol – before the importing

```

J MyMessage.java    J TaRDISApplication.java X
babel-test > src > main > java > com > tardisproject > J TaRDISApplication.java > TaRDISApplication > init(Properties)
14  public class TaRDISApplication {
15  }
38
39  protected TaRDISApplication() {
40      this.babel = Babel.getInstance();
41  }
42
43  private void init(Properties properties) throws Exception {
44      // Initialize Eager Push Gossip Broadcast Protocol
45      EagerPushGossipBroadcast eagerPushGossipBroadcast = new EagerPushGossipBroadcast(null, properties, null);
46      this.babel.registerProtocol(eagerPushGossipBroadcast);
47
48      // Pass the properties object to the protocol
49      eagerPushGossipBroadcast.init(properties);
50
51      // Initialize Babel
52      this.babel.start();
53      logger.info(message:"TaRDISApplication started successfully!");
54
55      // Initialize protocols
56      // MyProtocol protocol = new MyProtocol();
57      // this.babel.registerProtocol(protocol);
58
59      // Pass the properties object to the protocol for its own initialization
60      // protocol.init(properties);
61  }
62  }

```

Figure 8: Babel importing of a protocol – after the importing

Another powerful feature within TaRDIS is the ability to Visualize Swarm Protocols, which integrates with the "Visual Swarm Protocol Editing" extension. If a valid swarm protocol file is detected in the workspace, users can launch a graphical representation of the protocol directly within VS Code. This visualization provides insights into the communication flow and interactions between entities, aiding in debugging and validation of distributed applications.

For projects based on DCR Choreographies, TaRDIS includes a dedicated Compile & Run DCR feature, allowing users to build and execute distributed systems based on dynamic condition response logic. By selecting a `.tardisdcr` file, users can configure the execution environment, specify the number of prosumers, and initiate the build process. The extension ensures that Docker is running, compiles and packages the project, builds a Docker image, and sets up a dedicated network for communication. Once the setup is complete, multiple prosumer instances are launched in separate Docker containers, each executing a part of the choreography. The integrated terminal in VS Code provides real-time logs, allowing users to monitor execution and analyse interactions between components. This feature facilitates the development and testing of distributed applications, making it easier to validate choreography behaviour in a controlled environment.

By combining project management, class generation, protocol importation, visualization tools, and execution capabilities, TaRDIS provides a comprehensive development environment tailored for decentralized and distributed systems. Whether users are working on Java-based Babel projects, federated learning algorithms, or DCR choreographies, the extension ensures a smooth and efficient workflow, integrating seamlessly with VS Code to enhance productivity and streamline development.

3 TARDIS TOOLBOX INTEGRATION

The development of the TaRDIS Toolbox is being progressively accompanied by the integration of its tools into the TaRDIS IDE. This section outlines the ongoing evolution of these integration activities.

3.1 T-WP3-01 WORKFLOWEDITOR

Describe Tool

A graphical editor which allows its users to edit either the textual specification or the graphical representation of a swarm protocol and update the other accordingly. It is used to design, analyse, and implement workflows between actors based on the Actyx tooling.

Details

See Caroline Bjørch Fallesen, “A Visual Studio Code Extension for Editing Swarm Protocols.” MSc thesis, Technical University of Denmark, 2025.

Available at: <https://findit.dtu.dk/en/catalog/679d746fabd7f915d2a6809d>

Integration needs

Visual Studio Code extension, with preliminary release and source code available at: <https://marketplace.visualstudio.com/items?itemName=CarolineFallesen.visual-swarm-protocol-editing>

The WorkflowEditor is a graphical editor for swarm workflows, offering a bidirectional link where the final user can edit either the text or the graphical representation of a swarm protocol, and update the other accordingly. It is used to design, analyse, and implement workflows between actors hosted on swarm devices.

The description of the tool was already stated in the TaRDIS deliverable D3.2 [1], showing that the tool has a self-contained functionality, and listing the foreseen interactions with the IDE.

This tool was already integrated in the IDE, it is offered as an VS Code extension that is invoked by the IDE in case of the creation of an Actyx project, as shown in Figure 9. The IDE instantiates an empty workflow and invokes the editor. All edits of the workflow definition then occur via the graphical editor. The resulting workflow and corresponding declarative persistency statements are then stored in the environment workspace, which can then be used to define the swarm activities, using e.g., the Actyx machine-runner library (T-WP4-01) and the Actyx machine-check library (T-WP4-02).

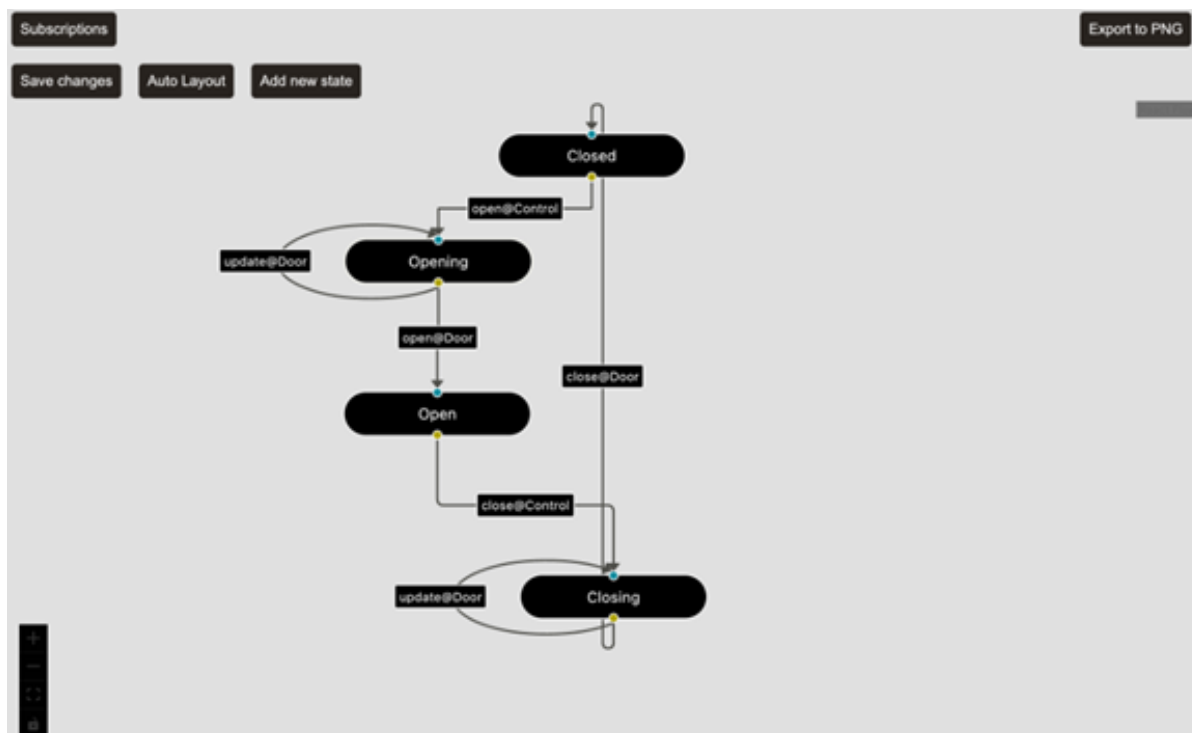


Figure 9: Screenshot by Caroline Bjørch Fallesen - from the MSc thesis “A Visual Studio Code Extension for Editing Swarm Protocols” (DTU, 2025). Available at: <https://findit.dtu.dk/en/catalog/679d746fabd7f915d2a6809d>

3.2 T-WP3-02 SCRIBBLE EDITOR

The Scribble editor (NuScr) serves as an extensible toolchain for MPST-based multiparty protocols. This toolchain converts multiparty protocols into global types within the MPST theory. These global types are then projected into local types and further transformed into corresponding communicating finite state machines (CFSMs). Additionally, NuScr generates APIs from these CFSMs to implement endpoints in the protocol. The design of NuScr supports language-independent code generation, enabling APIs to be generated in various programming languages. This tool is packed closely with the Scribble extensible toolchain for MPST (T-WP4-05).

This tool is under development and will be integrated with the IDE in a near future.

3.3 T-WP3-03 DCR CHOREOGRAPHY EDITOR

TaRDIS provides an editor and compiler for files whose source language is formatted as a DCR choreography. Its output is the projected behaviour in the form of Java code to be integrated in a fully functional communication framework.

A DCR choreography specifies the messages exchanged between participants as well as constraints on control flow that define causality between events and messages in the system's logic. Such choreographies go beyond the traditional sequential choreography specifications

(c.f. sessions). It allows for a more flexible and extendable programming framework for swarms.

The integration of this tool into the TaRDIS IDE streamlines the entire process—from project creation to compilation and execution—by providing an intuitive interface for managing DCR Choreography projects, as shown in Figure 10.

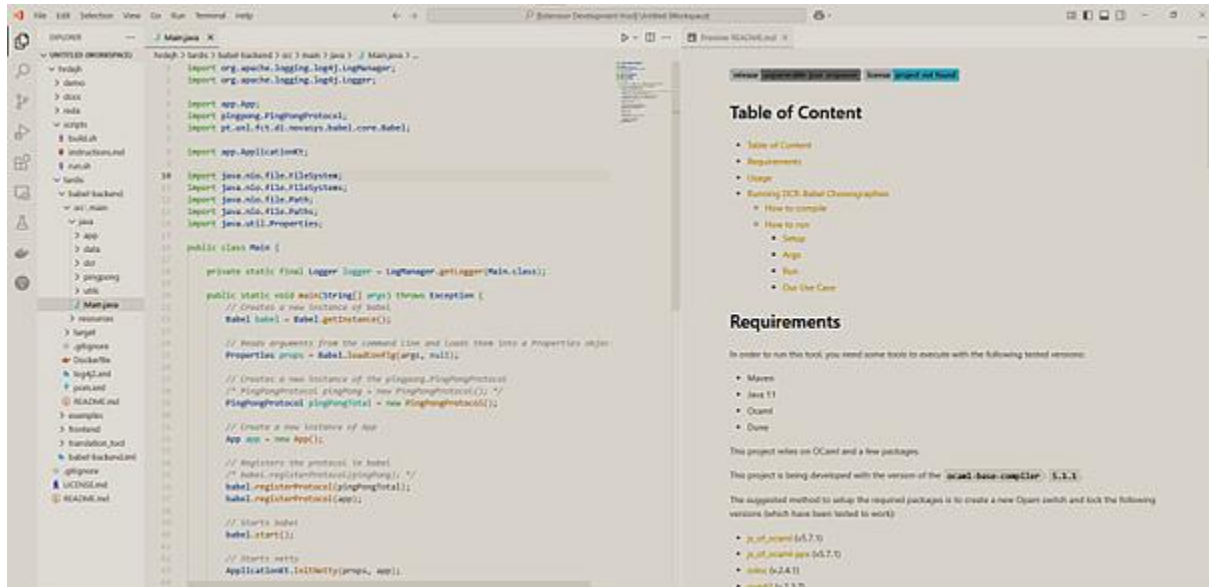


Figure 10: DCR Choreography integrated in the TaRDIS IDE

When a new DCR Choreography project is created within TaRDIS, the extension automatically generates the necessary folder structure and configuration files to support execution. This predefined workspace ensures that all dependencies are properly set up, allowing users to focus on defining their choreography without worrying about manual setup. The project structure includes a dedicated workspace for the DCR logic, configuration files for managing dependencies, and a preconfigured environment that ensures compatibility with the required tools. Once the project is initialized, the user is guided through the next steps of compiling and running the choreography, through a readme preview that pops-up on the right (as seen in Figure 10) in order to provide an explanation on how to operate and set up the project just created. When a DCR project is present in the workspace, a new button appears on the activity bar under the TaRDIS extension tab as shown in Figure 11 on the left, as well as the command “compile TaRDIS DCR” on the same figure on the right, to perform the compilation at the press of a button.

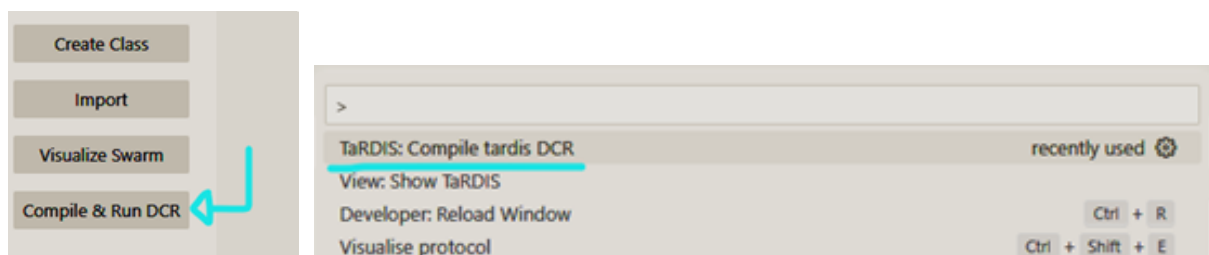


Figure 11: IDE buttons for compiling a DCR Choreography project

The TaRDIS extension simplifies the execution of DCR Choreographies by integrating directly with Docker, Maven, and other required tools. It provides an interactive interface where users can configure and execute their distributed workflows with minimal setup. Users start by pressing the compile button, then a menu pops-up on the right panel (figure below) giving the programmer the option of choosing a .tardisdcr file from their workspace, which defines the communication behaviour of multiple participants (prosumers) in the distributed system. They can then specify the number of prosumers that will participate in the choreography execution. By default, the extension sets up three prosumers, but this value can be adjusted, as shown in Figure 12.

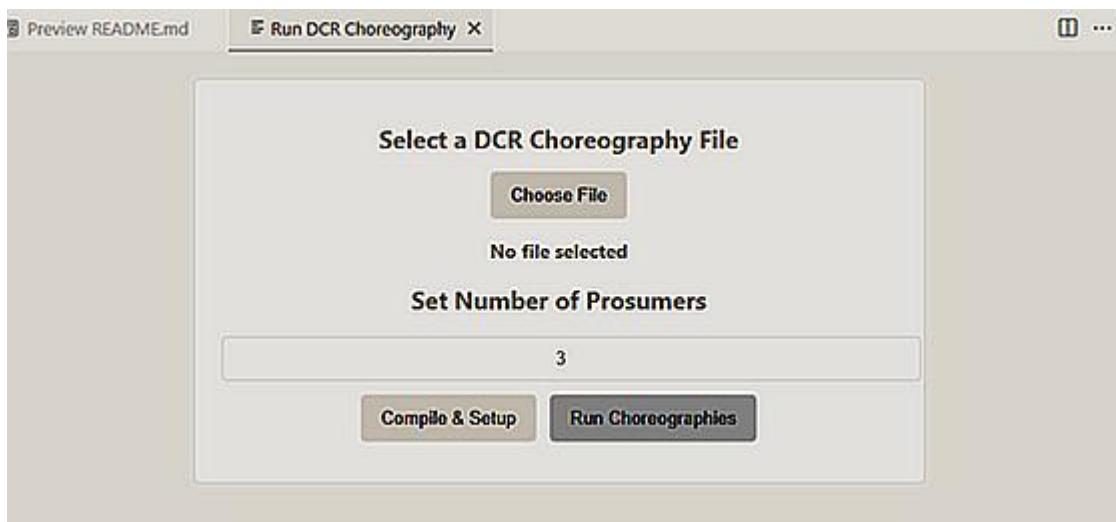


Figure 12: Run a DCR Choreography form

Once the file and configuration are selected, the compilation and setup process begin. By clicking Compile & Setup, the extension verifies that Docker is running, executes necessary build scripts, compiles and packages the project using Maven, builds a Docker image for running the DCR Choreography, and creates a dedicated Docker network to allow communication between prosumers, as shown in Figure 13. These steps ensure that the execution environment is correctly set up before launching the choreography.



Figure 13: Building the environment for a DCR Choreography

With the environment ready, users can execute the choreography by clicking Run Choreographies, which starts multiple terminals, each running a separate prosumer instance inside an isolated Docker container. Each prosumer is assigned a unique identifier (e.g., P_0, P_1, P_2), and the DCR logic is executed according to the provided specifications, simulating a distributed environment. Throughout the execution, users can observe real-time logs in VS Code's integrated terminal, allowing for direct monitoring of message exchanges, event triggering, and overall choreography behaviour.

By following this structured workflow, developers can easily simulate distributed systems using DCR choreographies, ensure correctness in message exchange and event handling through automated validation, seamlessly manage execution environments without manual configuration, and quickly debug and refine their workflows with real-time feedback in VS Code. To enhance usability, the TaRDIS extension also provides a graphical interface for managing DCR execution. This interface allows users to select a `.tardisdcr` file, set the number of prosumers, and monitor execution progress while displaying any errors that may occur. This integration ensures that users can focus on developing their DCR-based communication logic without needing to manually manage complex execution environments.

3.4 T-WP4-03 JOINACTORS

JoinActors is a Scala 3 library (developed by DTU) for performing pattern matching on complex combinations of messages/events and conditions. The underlying matching algorithm implements a formal specification of “fair matching” ensuring that, if some incoming message can be matched by a pattern, then that message will be eventually matched and processed.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.5 T-WP4-06 JAVA TYPESTATE CHECKER (JATYC)

A tool that verifies Java source code with respect to tpestates. A tpestate is associated with a Java class by the `@Typestate` annotation and defines the object's states, the methods that can be safely called in each state, and the states resulting from the calls. The tool statically verifies that when a Java program runs sequences of method calls obey to object's protocols, objects' protocols are completed, null-pointer exceptions are not raised, and a subclass' instances respect the protocol of their superclasses.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.6 T-WP4-07 DATA CENTRIC CONCURRENCY (ATOMIS)

This extended Java compiler is used to mark resources which need to be accessed in mutual exclusion; a type-checking and inference system ensures race freedom and produces deadlock free code.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.7 T-WP4-08 ANTICIPATION OF METHOD EXECUTION IN MIXED CONSISTENCY SYSTEMS (ANT)

A tool to statically determine operations that can safely commute with other operations and use this information to allow the run-time to anticipate calls to commutable operations. The analysis takes into consideration the consistency policy of each operation.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.8 T-WP4-09 CORRECT REPLICATED DATA TYPES (VERIFx)

A language to design provably correct replicated data types (RDTs), supported by a library of verified conflict-free RDTs.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.9 T-WP4-10 IFCHANNEL

Verify that the use of channels (generating and reacting to events) respects the secure information flow policy, e.g., confidential information of some group of participants is not accidentally transmitted on a channel where non-members of the group can read. This includes implicit flows, e.g., we assume the attacker can see that communication is occurring.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.10 T-WP4-11 PSPSP

Tool for verifying security protocols that setup and implement the channel (e.g., TLS) or change group memberships and the associated key infrastructure. This is internally used by TaRDIS for verifying security of the communication infrastructure that libraries provide.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.11 T-WP4-12 CRYPTOCHOREO

An implementation of a verified choreography will not be secure if the actors do implement the behaviour expected by the top-down model. Cryptographic Protocols formulated as a choreography can be translated into local behaviours. Local behaviours can be checked with PSPSP and also an implementation can be derived from them.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.12 T-WP4-13 (SEC)REGRA DA-IFC AND DCR CHOREOGRAPHIES

A compiler and type checker with Dependent Information Flow Control for ReGraDa graphs, mapped onto a centralised graph database, currently being extended to a decentralised version using Actyx as backend runtime support.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.13 T-WP5-01 FLOWER-BASED FL MODEL TRAINING

The Flower-based FL model training tool provides ML model training by utilizing federated learning solutions. The aim of the tool is to offer an AI/ML library with a set of different decentralized solutions, as well as to provide use case specific solutions. The implementations of FL algorithms are relying on the Flower framework. The tool provides a simple user interface that does not necessarily require expert knowledge to start the training process. The user can select the task that needs to be solved, and the tool provides a list of applicable models and algorithms. The finished training process produces a trained ML model and a training status overview. This provides a customizable and reliable approach that supports the developers' decisions with ease of use.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.14 T-WP5-02 DATA PREPARATION FOR FLOWER-BASED FL MODEL TRAINING

The Data preparation for Flower-based FL model training tool provides data preparation and preprocessing approaches for the ML model training, with the aim to overcome potential irregularities in the target data set. This includes common approaches, such as dealing with outliers, duplicates, missing values etc., but also some custom data preparation techniques, for example pseudo-labelling, that enables adding labels to an unlabelled data set, when a model requires them. The tool offers the facilitation of the ML training process (it provides input for the T-WP5-01 tool), by supporting a more efficient process of the preparation of the data.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.15 T-WP5-03 FLOWER-BASED FL MODEL INFERENCE AND EVALUATION

The Flower-based FL model inference and evaluation tool provides the possibility of getting output for the relevant data on a model trained by tool T-WP5-01. The output can be of different forms, depending on the needs regarding the relevant data, for example, predictions, forecasting, anomaly detections, and metrics. It offers a straightforward approach for gaining valuable insights into the quality of the trained model and for obtaining important conclusions.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.16 T-WP5-04 PTB-FLA AND MPT-FLA

As reported in D5.1 [6] and D5.2 [7], PTB-FLA stands for Python Testbed for Federated Learning Algorithms. It consists of a framework for developing and testing distributed federated learning algorithms. PTB-FLA execution environment provides SPMD (single program multiple data) applications' launching facilities and the simple API (amenable both to AI & ML developers who do not need to be professionals and generative AI tools), which offers the generic centralized/decentralized federated learning algorithms that may be specialized by specifying client and server callback functions.

PTB-FLA is a completely independent solution based on pure Python, without additional dependencies, that is available as open source. It is directly compatible with the main AI & ML libraries and supports the development of both centralised and decentralised federated learning algorithms. This tool was designed to target small IoTs in edge systems, such as Raspberry Pi Pico W boards, ROS2 robots, etc. and support the FL algorithm development on a single computer. With its simple API, the tool is easy-to-use by non-professional programmers, and it is amenable to LLMs such as ChatGPT. To facilitate easier development of the federated learning algorithms from sequential algorithms, a development paradigm was proposed to guide developers in transforming a referent sequential code into the target PTB-FLA code. The target PTB-FLA code is also easy to transform into the CSP formal model, which can then be used to formally verify the system properties, such as deadlock freedom, termination, etc.

MPT-FLA, which stands for MicroPython Testbed for Federated Learning Algorithms, was developed as a successor of PTB-FLA to support very small IoT edge devices with very limited computational capabilities. It is based on MicroPython, a lightweight version of Python. It supports decentralised applications whose instances can run on Raspberry Pi Pico W boards, robots, and PCs, connected to a Wi-Fi network.

The PTB-FLA and MPT-FLA are already being integrated with the TaRDIS IDE, namely with the inclusion of the creation of a PTB-FLA project, which includes several options, as seen in Figure 14.

The TaRDIS IDE now provides seamless support for the creation and management of PTB-FLA projects, offering a user-friendly environment for the development of federated learning algorithms. When creating a PTB-FLA project, users have access to various options designed to streamline the setup and development process.

One of the key features is the automatic setup of a dedicated Python virtual environment (venv_ptbfla). This ensures that all necessary dependencies are pre-installed, guaranteeing compatibility with major AI & ML libraries and eliminating the need for manual configuration.

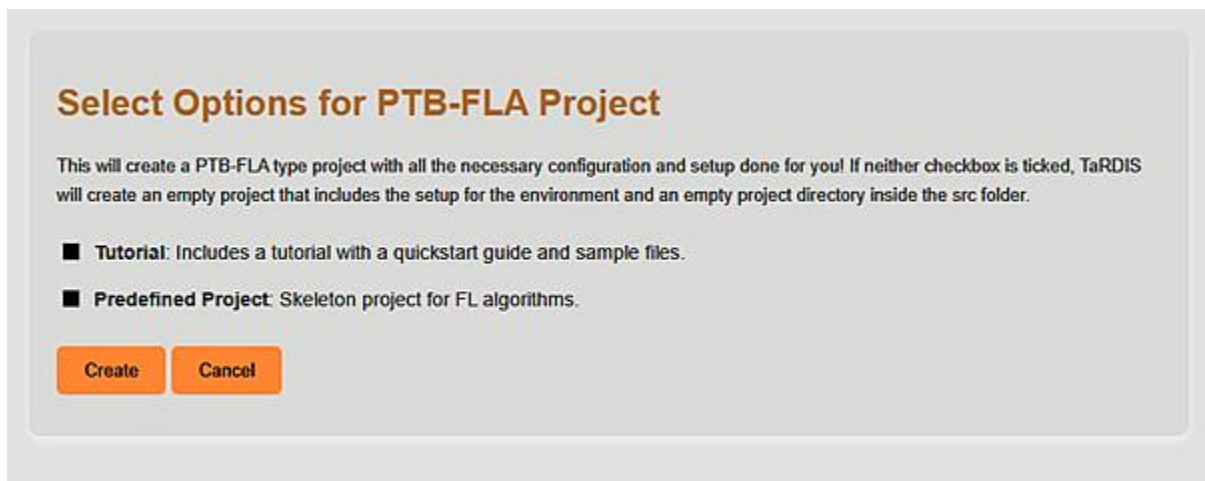


Figure 14: Configuration of a PTB-FLA Project

To accommodate different levels of expertise and project requirements, users can choose between several project structure options, the menu can be seen in the figure above. The most basic option is the Empty Project, which provides a clean and minimal PTB-FLA project structure, allowing developers to configure it according to their needs. Alternatively, users may opt for the Guided Tutorial, which assists newcomers in navigating PTB-FLA development. In this mode, the left panel of the IDE displays a script for novice PTB-FLA programmers, while the right panel contains a README file with step-by-step instructions, ensuring a smooth learning experience (see Figure 15).

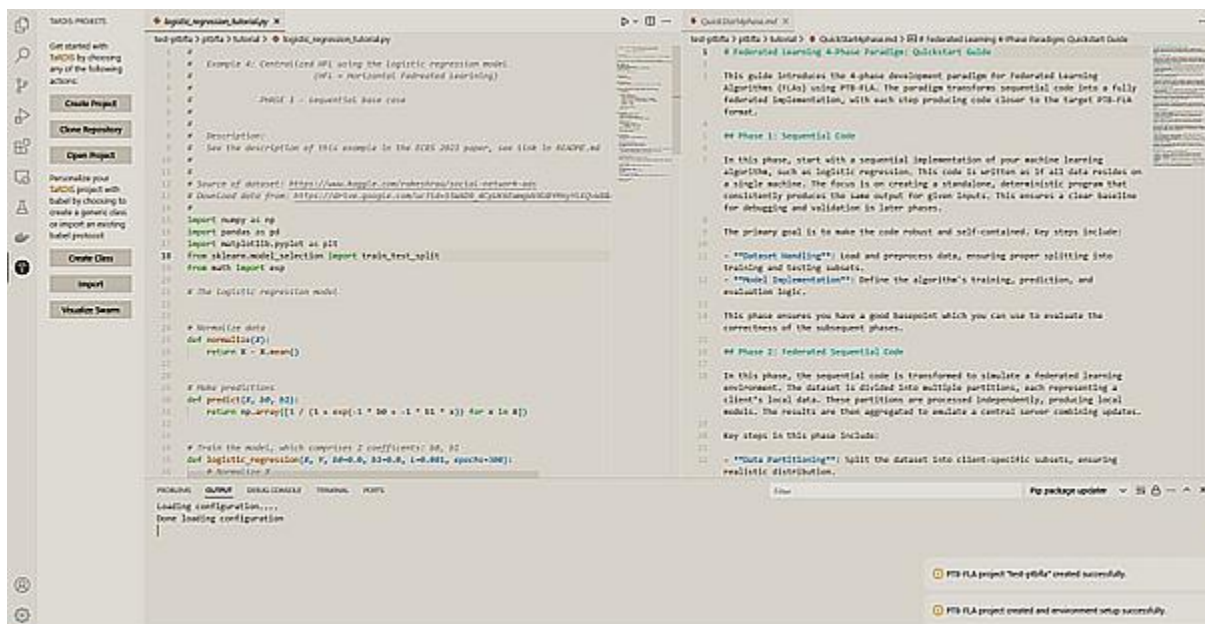


Figure 15: PTB-FLA project considering less experienced developers' support

For those who prefer a predefined starting point, the TaRDIS IDE offers three structured templates for federated learning algorithms, as depicted in Figure 16:

1. Centralized FLA, which follows a centralized server-client model.
2. Decentralized FLA, enabling fully distributed learning without a central coordinator.

3. TDM Peer Data Exchange, which allows time-division multiplexing for peer-to-peer data exchange.

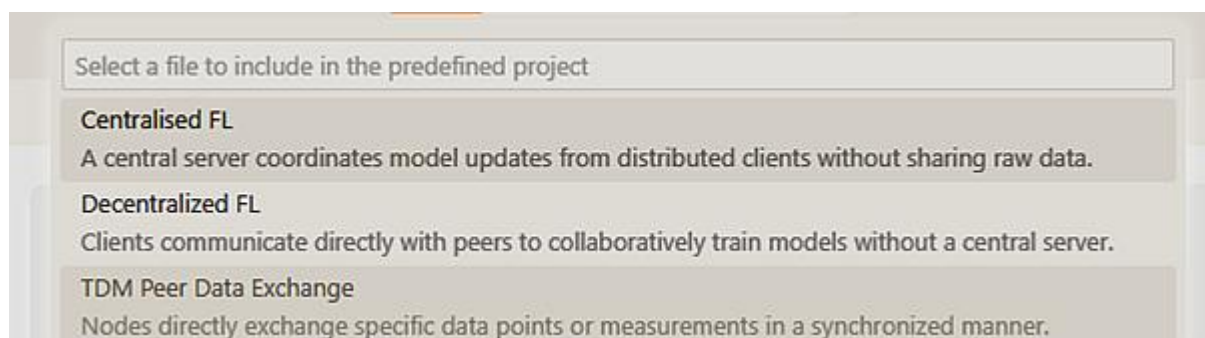


Figure 16: TaRDIS PTB-FLA form for selecting Federated Learning algorithms

Additionally, every PTB-FLA project includes a custom implementation folder. Upon creation, the TaRDIS IDE automatically generates a `<project_name>_src` folder where users can develop their Python-based Federated Learning (FL) implementations. If one of the predefined FLA skeletons is selected, the corresponding boilerplate code is placed in this folder, significantly reducing the effort required to get started.

These enhancements allow developers to quickly create, customize, and experiment with PTB-FLA projects while ensuring seamless integration with AI & ML workflows. Whether starting from scratch, following a guided tutorial, or leveraging predefined federated learning templates, the TaRDIS IDE provides an efficient and structured environment for federated learning development.

3.17 T-WP5-05 FEDERATED AI NETWORK ORCHESTRATOR (FAUNO)

The Federated AI Network Orchestrator (FAUNO) is a tool providing state-of-the-art agents for Multi-Agent Reinforcement Learning (MARL) working in the collaborative, federated setting. FAUNO is compliant with the PettingZoo API and exploits and specialises the MARL framework for the planning, deployment, and orchestration of the complete TaRDIS framework through federated reinforcement learning and other relevant methodologies.

This framework is trainable with the TaRDIS tool PeersimGym, a MARL environment for training MARL agents, already reported in D5.1 [6].

The FAUNO tool is already being integrated with the TaRDIS IDE, namely with the inclusion of the creation of a FAUNO project, which includes several options.

3.18 T-WP5-08 PRUNING

Lightweight Functionalities and Energy-Efficient ML

The lightweight inference functionalities provided by these tools are: (i) The early-exit tool transforms a pre-trained deep neural network (DNN) model in a more lightweight version that includes multiple exits during the model feed-forward for purposes of providing quick inference

at the cost of reduced accuracy. In addition, a distributed form of the early-exit tool was developed in order to implement the deployment of the early-exit in a distributed architecture, i.e., model segment among several edge nodes; (ii) The knowledge distillation tool transforms a pre-trained ML model in a more lightweight version in terms of network complexity, number of neurons and ultimately in terms of computational intensity. In specific, the original model is utilized to train a student, more lightweight model, essentially reducing the computational resources required during the inference process at the cost of decreased model accuracy; (iii) The pruning tool again transforms a DNN in a more lightweight version by nullifying the neuron connections that have a negligible impact on the DNN performance. To this end, the pruning functionality streamlines the inference process, in terms of latency and conservation of energy and computational resources.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.19 T-WP5-09 DECENTRALISED FEDERATED LEARNING FRAMEWORK (FEDRA)

This tool provides a decentralised federated learning framework integrated with p2p communications between the participating nodes, specifically designed for swarm systems. Fedra leverages libp2p for peer-to-peer communications between the swarm members, enabling the secure model weight exchange for aggregating the federated global model, guaranteeing privacy and data ownership. Moreover, Fedra is model-agnostic, in the sense that different ML algorithms can be seamlessly integrated and utilized to train ML federated models, including models for forecasting, resource allocation, anomaly detection, among others.

It should be noted that the frameworks that have been developed for FL training cover, in principle, different requirements. The Flower-based framework is more standardised, being acknowledged to the open-source community, while revisions and updates are frequent. On the other hand, Fedra deals with completely decentralised learning using p2p communication, without the requirement of a centralized aggregator in the framework. Finally, PTB-FLA framework deals with a more lightweight version of FL, to be deployed in resource-constrained devices.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.20 T-WP5-10 FLAAS

FLaaS (FL-as-a-Service), developed by Telefonica, is a practical federated learning framework for mobile environments that allows app developers to perform cross-device and cross-app (i.e., on-device cross-silo) FL. There are four core components of FLaaS: 1) App Developer Interface 2) FLaaS Server 3) Notification Service and 4) Client Devices. The FLaaS developer orchestrates the FL operations through the Admin Developer Interface and the clients' devices should have installed FLaaS local, which is a standalone service/app for Android devices. FLaaS may be used by a swarm developer in order to initiate and orchestrate a federated learning instance with Android devices and a cloud-based FL aggregator. Lastly, we stress

here that, while the initial version of FLaaS was built outside of TaRDIS, the development and subsequent integration of the TaRDIS tools into FLaaS will result in an improved or more modular version of FLaaS.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.21 T-WP5-11 SIMULATOR FOR PEER-TO-PEER NETWORKS

A simulation for training a Reinforcement Learning (RL) agent has been built as tool ML-Gym. It is divided into discrete time steps and can handle various network configurations, i.e., peer-to-peer networks but also networks where there is a hierarchy among the nodes. The simulator runs in Java and must be wrapped into a Python environment following the interface defined as a Markov Decision Process or a Markov Game, in the case of decentralised decision-making. Such RL training environments are embedded in an RL training framework, in our case, we chose PettingZoo. The developed environment models task offloading-based orchestration. It is an open platform that the team plans to extend to broader action spaces.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.22 T-WP6-01 A GENERIC API FOR DECENTRALISED OVERLAY AND COMMUNICATION PROTOCOLS

This tool consists of a collection of protocols (i.e., pre-made interactions between swarm participants for various shapes of communication, to be performed over network connections), a runtime component for managing instantiated protocols, and programming language bindings for interacting with the protocol manager as well as each protocol instance. This allows a TaRDIS application to use higher-level communication primitives than manually opening connections and sending or receiving bytes on them. Examples include efficient routing of messages between peers that are not directly connected or broadcasting from one peer to a group of peers, each offering a range of message delivery guarantees to select from.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.23 T-WP6-02 AN EPIDEMIC AND SCALABLE GLOBAL MEMBERSHIP SERVICE

This membership abstraction (provided in the form of a library) allows a TaRDIS application to obtain information on the size of the surrounding swarm and the identities and health of its participants. In contrast to earlier work that only gave a partial view (focusing on a peer's neighbours), this tool aims to provide a global view, albeit with eventual consistency—i.e. after a membership change there is a delay before this change is reflected in the swarm view presented on all participating peers.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.24 T-WP6-03 ACTYX: RELIABLE EVENT BROADCAST WITH CONFIGURABLE DURABILITY

Actyx is a middleware tool that will internally use the above tools for swarm communication and membership to provide even higher-level services to TaRDIS applications, namely the reliable and durable dissemination of event streams within the swarm. This is significant because individual events are quite small and typically don't warrant the overhead of being individually treated (e.g. for being identifiable or localisable) in a swarm system. Therefore, Actyx partitions the emitted events into streams that are then the unit of dissemination, leading to significant benefits in compressed event storage size. Storage resource usage can be controlled via configurable per-stream data retention policies. Actyx also introduces an eventually consistent global order between events that allows the resolution of conflicts arising from concurrent swarm behaviour in a fashion that does not compromise on system availability or resilience.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.25 T-WP6-04 BABEL

This internal tool allows a more formal expression of low-level communication behaviour of an application or algorithm and its validation in a variety of environmental scenarios (i.e., network availability and performance). It will be used to develop and test several of the TaRDIS tools, and it may also be of use to external developers, for example when they create their own communication protocols to be used via the generic communication protocol API. The Babel framework existed before the start of TaRDIS. In the context of TaRDIS Babel has evolved into a full ecosystem for supporting the development of highly decentralised swarm applications compatible with a variety of different devices. This ecosystem is composed of Babel-Swarm which enriched the framework with support for security, self-configuration, and self-management; and Babel-Android which transfers and expands the capabilities of Babel to the Android environment, including mobile phones and tablets.

This tool was integrated into the IDE, consisting of an option to create a Babel project, with multiple options available which may take into account the expertise of the developer. When creating a new Babel project, the developer is guided through a structured process to ensure a smooth workflow. The process begins with the selection of a project type, where the user can choose to create a Java-based project utilizing Babel libraries. The IDE then prompts the developer to enter a project name and select a directory where the project should be created, as you can see in the example below, depicted in Figure 17.

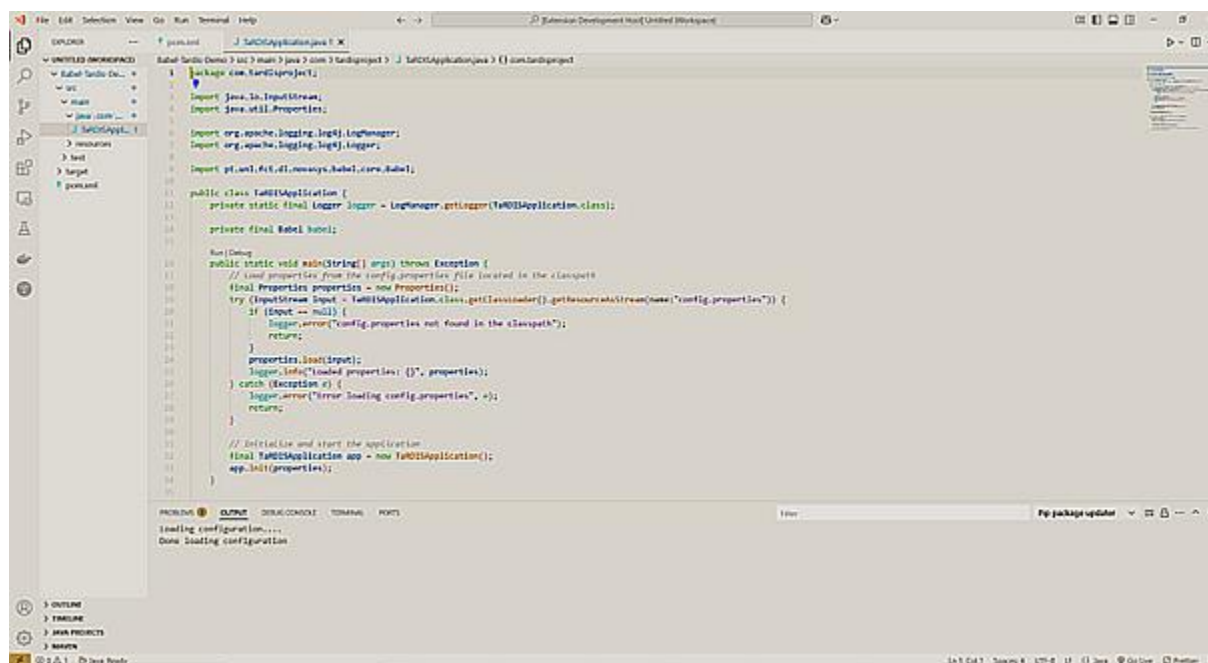


Figure 17: TaRDIS IDE integration with Babel

Once the project details are set, the extension automatically generates the required folder structure, including the necessary Maven configuration files. The setup includes predefined dependencies for Babel, ensuring that the project is ready to be developed without requiring manual dependency management. The generated structure provides a dedicated src folder, where Java classes and the main application logic can be implemented. This automated setup eliminates the need for manual environment configuration, allowing developers to focus on building their applications.

After setting up a new Babel project, the TaRDIS extension offers additional tools that enhance development productivity. Within the TaRDIS project view in the sidebar, users have access to buttons that facilitate common tasks, including the creation of essential Babel components and the importation of predefined communication protocols developed by NOVA. These options allow users to extend functionality of their project without any manual configuration of dependencies onto the pom.xml file or writing boilerplate code.

To assist developers in managing their project's communication structure, the extension provides a streamline process for importing existing Babel protocols. When clicking on the **"Import Protocol"** button, the user is presented with a categorized dropdown list of communication protocols available for integration. The protocol selection menu, as shown in the screenshot, allows the user to browse different protocol types, including options such as AntiEntropy, Eager Gossip Broadcast, Flood Broadcast and One Hop Broadcast. This categorization ensures that developers can quickly choose and locate the protocol that best suits their application needs, as seen in Figure 18.



Figure 18: Babel selection of communication protocols

Once a protocol is selected, a detailed information panel appears within the right side panel, providing an overview of the protocol's functionality, implementation details and configuration parameters. This information is presented in a formatted documentation view, making it easier for developers to understand how the protocol works before making a decision to integrate it into their project. The protocol documentation outlines key aspects such as membership services, broadcast mechanisms and required configuration parameters. For example, in the case of the Eager Gossip Broadcast Protocol, the documentation specifies how it handles message dissemination and highlights relevant parameters like the fanout setting.

The information panel also includes two interactive buttons: **Import Protocol** and **Cancel**, as shown in Figure 19. The Import Protocol button confirms the user's selection and proceeds with integrating the protocol into the project. The extension automatically updates the pom.xml file to include the necessary Maven dependencies and repository links. Additionally, it injects the required initialization code into the main application, ensuring that the protocol is properly registered and ready for use. This automated integration significantly reduces the risk of misconfiguration and allows developers to start using the protocol without additional setup.

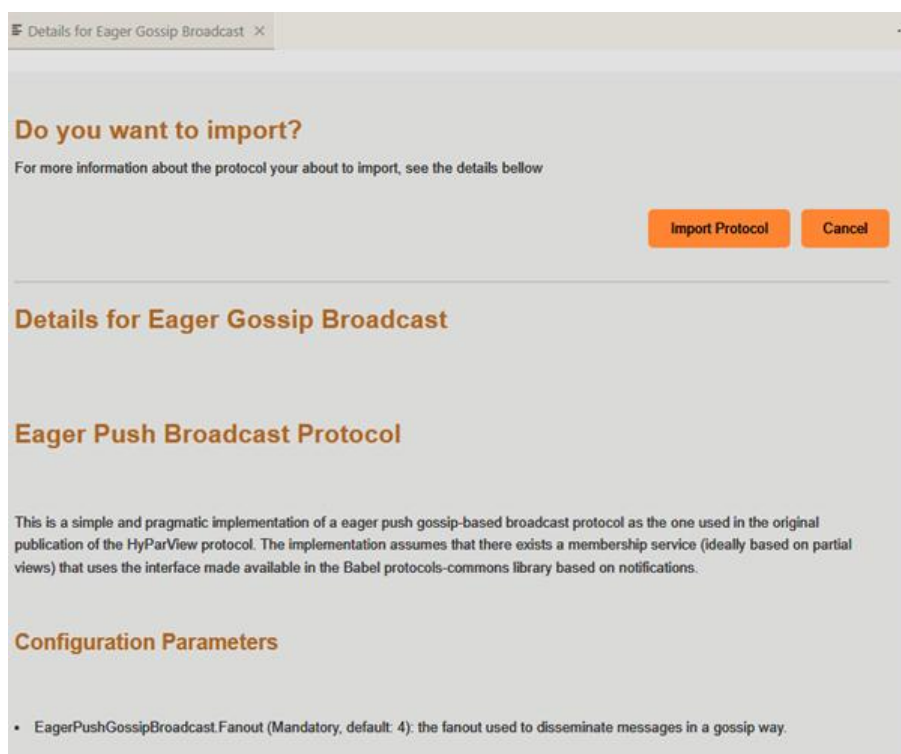


Figure 19: Babel form for confirmation of inserting a protocol

If the developer decides that the selected protocol is not suitable, they can click the **Cancel** button, which closes the window without making any modifications to the project. This provides an intuitive and non-intrusive way to explore available protocols before committing to an import.

Beyond protocol integration, the TaRDIS extension further accelerates Babel projects development by simplifying the process of creating commonly used Babel components and classes. The **Create Class** button, as seen in Figure 20, enables users to generate structured Babel elements such as Messages, Protocols, Timers, Requests, and Replies. When selecting this option, the user is prompted to choose the type of class they wish to create and provide a name for it. Once confirmed, the extension generates the necessary Java class file with predefined templates, ensuring that it is structured correctly and follows best practices for Babel-based development.

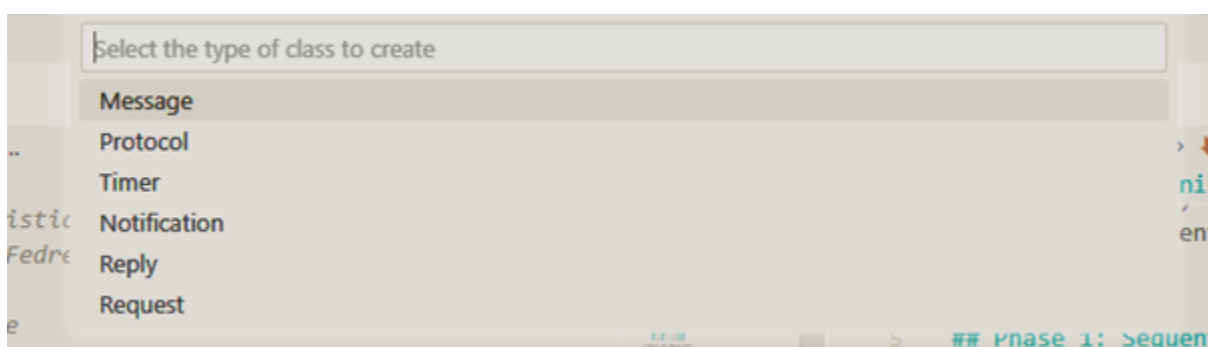


Figure 20: Babel form for creation of a class

By incorporating these functionalities, the TaRDIS extension provides a comprehensive toolkit for developing distributed applications in Java using the Babel framework. The seamless workflow, from project creation to protocol integration and component generation, allows developers to build robust communication-driven applications efficiently. This combination of automation, predefined templates, and interactive documentation makes TaRDIS an invaluable extension for managing Babel-based projects within Visual Studio Code.

3.26 T-WP6-05 ARBOREAL: EXTENDING DATA MANAGEMENT FROM CLOUD TO EDGE LEVERAGING DYNAMIC REPLICATION

Arboreal is a data management tool that replicates key-value bindings across data centres and dynamically distributes updates to these bindings within each data centre according to the declared interests of each edge node. Due to the way updates are propagated and using the included metadata, the system ensures so-called causal+ consistency which means that updates become visible at any swarm participant in an order where causality is preserved (i.e. you only see an effect once you have already seen the corresponding cause, and you will eventually see all changes) while ensuring that all replicas of a data objects eventually converge to the same value. This makes Arboreal a fully available and high-performance NoSQL database a.k.a. key-value store.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.27 T-WP6-06 POTIONDB: STRONG EVENTUAL CONSISTENCY UNDER PARTIAL REPLICATION

PotionDB is a data management system designed to be deployed in a small number of nodes, potentially geo-distributed, and with support for partial replication. Unlike Arboreal, PotionDB supports a transactional API, thus providing a more powerful API for the application. Still under development, it is the support for materialised views over geo-partitioned data, providing a mechanism for supporting recurrent queries that are common in applications.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.28 T-WP6-07 INTEGRATION OF STORAGE SOLUTIONS INTO THE TARDIS ECOSYSTEM

Similar in spirit to the generic API for communication, this library facilitates the use of a range of storage solutions by a TaRDIS application through a common API. Current solutions other than the above include the industry standard Cassandra (also with C3 enhancements for causal+ consistency), Engage, and Hyperledger Fabric. The latter will allow to easily leverage on blockchains in the design of TaRDIS use cases, which will provide a tamper-proof and publicly verifiable ledger where, for instance, exchanges between members of the swarm can be reliably registered, for instance, energy exchanges among elements of the renewable energy community.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.29 T-WP6-08 DISTRIBUTED MANAGEMENT OF CONFIGURATION BASED ON NAMESPACES

This tool allows swarm administrators to inject the desired configuration for any participant or application running on it. Parameters are selected based on namespaces to isolate parts of the system from each other, which allows a large measure of heterogeneity within the swarm: applications do not need to be co-designed to guard against interference, and even different versions can be clearly and cleanly separated. In addition, labels are used to allow fine-grained grouping of any kind of resources, allowing homogenous configuration where required.

This tool is being developed and will be considered for integration with the IDE in a near future.

3.30 T-WP6-09/10 TELEMETRY ACQUISITION FOR DECENTRALISED SYSTEMS

These tools (one for containers and one for Babel protocols) consist of a manager, registries, and exporters for a wide variety of measurements performed within a running swarm. Due to the dynamic nature and usually complex communication topology of such systems, dedicated tooling is necessary to transport this data from the device where it originates to the person or ML algorithm that monitors the swarm. The purpose is to enable the smooth operation of the system, which includes quick incident response as well as proactive management of resources to avoid incidents. This is currently being evolved to take advantage of in-network processing

by swarm elements to ensure that telemetry can be effectively and efficiently processed and disseminated to elements of the swarm that make decisions related to the current configuration.

This tool is being developed and will be considered for integration with the IDE in a near future.

4 TARDIS DEVELOPER STORIES

This section aims to state as many as possible different storyboards on how to develop a swarm project, in order to create an easy identification of the challenges that businesses will face when developing a swarm environment.

4.1 DEVELOP A BABEL SWARM PROJECT

Building swarms is a complex and challenging task due to the inherent unpredictability and scale of such systems. These systems often consist of multiple nodes that may be located in different geographic regions and need to collaborate seamlessly to provide services or process data. The difficulty arises in managing issues like network latency, node failures, variable load distribution, or possibly node displacement in particular environments.

Babel is a framework that aims to simplify the development of distributed protocols, by offering a set of tools, abstractions, and best practices to allow developers to design, deploy, and manage dynamic, scalable, and robust distributed applications.

The main unit of interaction with the framework is the protocol. Protocols embed the logic implemented by the developer and use the abstractions provided by the framework to interact with other protocols being executed locally, as well as handling communication with other nodes. When using Babel, developers specify a set of protocols for each node, which will dictate the different behaviours and interactions with the system. For instance, if a user intends to create an application to transmit messages, a possible protocol stack would be:

- 1) A protocol for reading the input from the user
- 2) A protocol to disseminate the messages through the network to other nodes, and
- 3) A protocol in charge of membership, i.e., a protocol responsible for finding other nodes/neighbours in the swarm, leading to this interaction, as depicted in Figure 21.

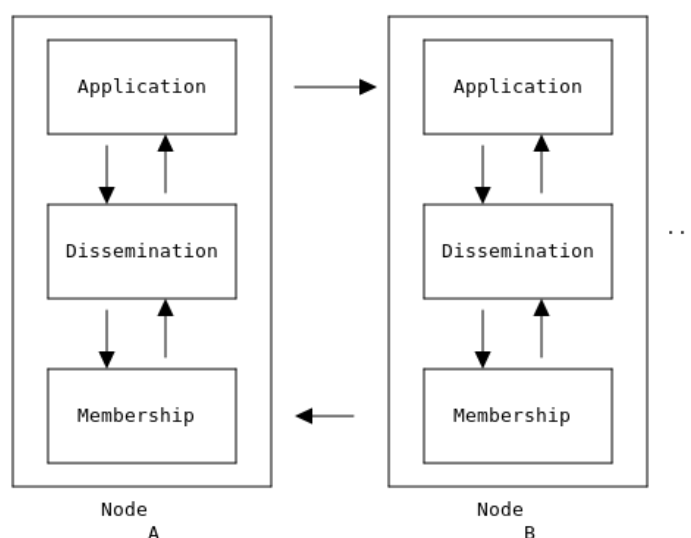


Figure 21: Developer Story over the development of an application using Babel

In this context, we will describe the developer story for creating applications with Babel.

4.1.1 INITIALIZATION

First and foremost, to use Babel, each node in the system should specify the list of protocols it wishes to use, as depicted in Figure 22:

```
main(config):
    framework = Babel.newInstance()
    props = framework.loadConfig(config)

    app = MessageApplication()
    dissemination = EagerGossipProtocol()
    membership = FullMembershipProtocol()

    framework.registerProtocol(app)
    framework.registerProtocol(dissemination)
    framework.registerProtocol(membership)

    app.init(props)
    dissemination.init(props)
    membership.init(props)

    framework.start()
```

Figure 22: Example of Babel definitions for a swarm node

Each Babel instance receives a configuration file, which contains the different parameters that are supposed to be loaded into the framework (i.e., method for discovering other nodes, the network interface being used in the device, etc.).

After this, the developer should specify which protocols are wished to be used for its deployment. Finally, when each protocol is initialised, the framework starts executing.

4.1.2 PROTOCOL SPECIFICATION

One of the most important aspects of Babel lies in choosing the right protocols for the application being built. Protocols dictate how communication is done, what guarantees are provided in the system, etc. With this in mind, while integrating Babel with TaRDIS, we targeted two different audiences: developers with high expertise in the area, and newcomers.

Regarding the first, Babel provides extensive ways of building their own protocols and their custom APIs. With a meaningful variety of functionalities, Babel has all of the necessary tools to write protocols based on previously set specifications. This way, project owners who wish to build their protocol stack from scratch are able to do so, by writing or exporting their protocols from other sources and deploying their own swarm.

On the other hand, considering the developers with less experience, Babel allows importing protocols already written and tested by developers, from the already established TaRDIS

protocols repository, available on link <https://codelab.fct.unl.pt/di/research/tardis/wp6/Babel-swarm/protocols>. For instance, looking at the previous example, both the *EagerGossipProtocol* and *FullMembership* protocols are available in the repository with their respective specifications.

Each protocol has relevant information about communication mechanisms, implementation logic, guarantees and behaviour. When choosing already written protocols, developers are able to take an informed decision and use the protocols that align with their needs.

4.1.3 PRACTICAL EXAMPLES

Babel Examples² provides a tutorial with the goal of guiding the process of developing protocols and applications using the Babel framework. The repository contains five different examples with increasing complexity.

The tutorial describes the different abstractions of Babel and a boilerplate to create protocols from scratch. Moreover, the tutorial explains how to deploy such protocols. While this example presents the full-fledged functionalities of Babel, not all developers, especially beginners, have the required knowledge to implement every component in a swarm system.

Thus, to cover these scenarios, Babel Applications³ provides sample applications that import and use protocols already written by experts on swarm systems. In this context developers primarily focus on developing their applications and leverage protocols that handle the complexity of swarm systems, such as handling communication, ensuring fault tolerance in the presence of fails, etc. to create a highly dynamic application.

4.1.4 IDE INTEGRATION

As stated in [T-WP6-04 Babel](#), the IDE integration with Babel aims to facilitate the process of building swarm systems. The IDE focuses on giving an intuitive UI to create a project from scratch and adding the different building blocks that compose a node.

Users are able to write protocols (and the necessary events) through a boilerplate or simply import already written protocols with a simple press of a button. Regarding the last option, while importing a protocol, the IDE shows different details concerning its functionality.

4.2 DEVELOP A PTB-FLA PROJECT

PTB-FLA projects aid the development of federated learning applications and algorithms using the PTB-FLA and MPT-FLA frameworks. Based on the degree of familiarity a would-be developer has with both these frameworks and FL in general, three options present themselves, each catered towards a certain level of expertise.

² <https://codelab.fct.unl.pt/di/research/tardis/wp6/Babel/Babel-examples>

³ <https://codelab.fct.unl.pt/di/research/tardis/wp6/Babel-swarm/applications>

The first option would be to use the tutorial, aimed at newcomers, guiding them through the process of turning a sequential ML algorithm into a fully federated code ready for distributed execution.

The tutorial guides the user through the PTB-FLA four-phase federated learning algorithms development paradigm:

- Phase one refers to the development of the sequential machine learning algorithm.
- Phase two aims to partition (i.e., split) the data and train client models sequentially before aggregating them.
- Phase three introduces the use of client callback functions for training and server callback functions for aggregation and is the final step before introducing the PTB-FLA framework.
- The fourth and the final step in the paradigm consists of using the callback functions, developed in the third phase, as callback functions passed to PTB-FLA generic algorithms.

A more detailed description of the four-phase paradigm is given in TaRDIS deliverable D5.2 [7]. Going through these steps gives users a solid foundation, leading to a more formal and correct by construction FL application development workflow.

For intermediate developers, familiar with the framework, the predefined project option offers a choice of three generic algorithms while providing a preset application structure (also called a skeleton or a boilerplate). The developer is offered a choice of: Centralized FLA, Decentralized FLA and TDM Peer Data Exchange. These projects are ready to be filled with outputs of the third phase of the previously mentioned FL development paradigm or can be used for direct implementation by those confident in their experience. The generated project files outline the FL application, enhancing development efficiency while ensuring the structure aligns with the PTB-FLA restricted programming model and API.

The final and default option is an empty project, providing experienced users with complete control, enabling them to immediately start building FL applications as they like.

The common thread that connects all the project creation options is the automatic setup of the virtual environment and the pre-installed dependencies, saving users the trouble of doing it manually. This accounts for a more seamless and hassle-free experience across projects.

Testing the newly developed FL applications could be as easy as invoking the launch command from the terminal along with the main script path and required parameters (e.g., `launch ./ptbfla/examples/example1_fedd_mean.py 3 id 0`).

As previously mentioned, the PTB-FLA projects can also aid the creation of MPT-FLA based federated learning applications. This could be done by manually changing: (i) the imports from `ptbfla_pkg.ptbfla` to `ptbfla_pkg.mp_async_ptbfla`, (ii) the function `main` to the `main` coroutine and, (iii) the PTB-FLA API function calls to MPT-FLA API coroutine `awaits`, thus transforming the previously developed PTB-FLA applications into the corresponding MPT-FLA applications that may be used inside tiny IoTs running MicroPython.

4.3 DEVELOP A PTB-FLA WITH INTEGRATION WITH BABEL

This storyboard focuses on how a PTB-FLA project should be developed, with its option to be integrated with Babel.

The PTB-FLA Babel adapter may be used by PTB-FLA applications to leverage Babel's network communication capabilities in a seamless way that does not require any modifications to the applications. In order to do so, users should do the following:

1. Clone the PTB-FLA Babel repository.
2. In the file **src/main/resources/adapter.conf**, change `doppelganger.ports` to reflect those of PTB-FLA application instances running on other devices (IP is formed as 6000 + instance ID).
3. Build the app by running: `mvn clean package -U *`
4. Run it with the command:
`java -jar target/PTBFLA-Babel-adapter-0.0.6.jar Babel.address=<your_device_ip>`
5. Proceed to the startup of PTB-FLA applications manually on the devices.

Babel uses maven as a build tool, so it should be installed before using the adapter. If the remote ports change after the initial build of the adapter, the script `portsChangeUtil.py` can be used for changing the `doppelganger` port without repeating the build. If everything is set up properly, the PTB-FLA applications can be used for federated learning on multiple devices on the same network. The PTB-FLA Babel adapter code is publicly available on the repository PTB-FLA_BabelAdapter, available on https://github.com/LinguineP/PTB-FLA_BabelAdapter.

4.4 DEVELOP A DCR CHOREOGRAPHY

Storyboard on how a DCR Choreography project should be developed.

Developing a swarm application using DCR choreographies is possible using the Babel-based communication stack. The steps that need to be taken are the following:

- 1) Use the TaRDIS plugin for DCR choreographies to start a new application. This step creates a new application from an existing template in a git repository.
- 2) The developer then opens the application textual code in the TaRDIS editor and programs the choreography using DCR notation. The specification includes security levels for the information exchanged between participants.
- 3) The choreography is checked to eliminate security errors (confidentiality) whenever the developer compiles the code. The verification is based on the verification of the language SecReGraDa (IFC) and its runtime assumptions are guaranteed on the verification by IFChannel of the label communication layers.
- 4) The compilation process then projects the global behaviour of the choreography onto Java code that implements the local behaviour of each kind of swarm element.
- 5) The application code is fully integrated in a Babel application that can then be deployed following the standard procedures for Babel applications.

Ongoing and future developments should allow the TaRDIS development environment to edit DCR choreographies visually, deploy them to container instances, monitor swarms running, and change them in real-time.

4.5 COMPLEX SWARM: COMMON WORKSPACE FOR MULTIPLE PROJECTS

The development of a sophisticated workspace for intelligent swarms' software development requires a robust, flexible environment capable of handling multiple heterogeneous projects, each with unique requirements and configurations. Intelligent swarms are inherently complex, requiring a workspace that supports both the simulation and deployment of swarm algorithms, real-time data processing, Artificial Intelligence algorithms for Machine Learning, and integration with various hardware platforms.

A key challenge in developing such a workspace for intelligent swarms is managing the heterogeneity of the projects involved. Each project may have distinct dependencies, technologies, and runtime environments. As such, the workspace must support a range of programming languages (e.g., Python, C++, Java), simulation environments and external APIs to enable seamless development across different aspects of the swarm system.

A successful workspace for swarm development should adopt a modular architecture, allowing developers to work independently on different components or subsystems. This includes support for containerized environments (e.g., Docker) for each project, ensuring that the necessary dependencies and configurations are isolated and independent of other projects. Additionally, the workspace should support easy management of external resources, such as hardware for real-world deployment or cloud-based infrastructure for larger simulations or integration with ML activities.

The simple scenario, mostly followed by the TaRDIS use-case pilots, typically involves:

- A project for the development of some kind of Machine Learning (Federated Learning, Reinforcement Learning, etc.) algorithm training to create the backbone knowledge of the environment, e.g., using Fedra ([T-WP5-09](#)), PTB-FLA ([T-WP5-04](#)), FLaaS ([T-WP5-10](#)), FAUNO ([T-WP5-05](#)), or other.
- Then, on the same workspace, a new project can be developed, designing the swarm workflow(s) using tools like the WorkflowEditor ([T-WP3-01](#)), DCR Choreographies ([T-WP3-03](#)) or similar to model the swarm elements' behaviours and interactions. The workflows can use the knowledge extracted or inferred by the ML algorithms in the previous projects, as there can be APIs to make this knowledge available to the other projects of the workspace.
- Additional Java, C++, TypeScript or other projects may be created and developed on the same workspace to support the implementation of the behaviours defined in the previous workflows, and/or other tools to help on the coordination of the swarm. They can make use of the TaRDIS tools such as Babel ([T-WP6-04](#)) for defining the protocols, behaviours and patterns of elements of the swarm in tasks such as registering, membership, and specific protocol patterns.
- Following, the elements of the swarm can be launched locally or remotely using TaRDIS tools to launch, monitor and control local and remote nodes such as the

Distributed Management of Configuration based on Namespaces ([T-WP6-08](#)), and such configurations may be validated by the Scribble Editor tool ([T-WP3-02](#)).

More sophisticated functionality can also be foreseen and provided in the common environment workspace:

- The TaRDIS Toolbox provides services that should do reconfiguration and monetization of the swarm.
- The developers are allowed to create multiple logically isolated environments using concepts of namespaces and run different applications in them without the fear of name and/or resource conflicts.
- Using the provided APIs, the developers can disseminate information and configuration elements throughout the swarm using:
 - (i) direct message propagation to every affected node
 - (ii) peer-to-peer using already present gossip protocols
- Besides that, the Toolbox can collect metrics from the swarm hardware, kernel and applications. All metrics are stored in a centralized place which allows:
 - (i) dashboard visualisation of the metrics over time
 - (ii) specific APIs that other sides in and out of the toolbox can utilize for different purposes.
- The described elements are developed as multiple services communicating over the network, and to ensure that they do not end up in situations not favourable to the swarm, every major communication protocol that ensure both reconfiguration and monitoring are first modelled and verified using multiparty session types.
- Also, the reconfigurations of the system could be formally modelled with graph transformations, easing the development and creation of models and swarm elements infrastructures.

The development of a multi-project workspace for intelligent swarms software development involves creating an environment that can efficiently manage the complexity of heterogeneous projects while ensuring seamless integration and collaboration across various subsystems. By leveraging modern tools such as containerization, simulation environments, version control systems, and real-time data pipelines, developers can work within a flexible, scalable environment that supports both experimentation and production deployment. As intelligent swarm systems continue to grow in complexity and scope, the workspace must evolve to address emerging challenges, ultimately enabling the efficient design, testing, and deployment of decentralized, intelligent systems.

5 TARDIS PILOT (USE CASE) DEVELOPER STORIES

This section describes the more complex developer stories that are taking place in order to produce the complex interactions and flows that are required to provide a solution to the TaRDIS use-cases (pilots).

5.1 EDP NEW ENERGY: MULTI-LEVEL GRID BALANCING

This developer story considers the utilization in a single environment, of two TaRDIS tools: **Fedra**, as a framework for Federated Learning (FL) training, and **Pruning**, for transforming the trained ML model into a more lightweight version. This developer story uses a dataset regarding time-varying power generated by solar panels that are included in smart homes (each smart home has its individual local dataset). The ML model will enable the prediction/forecasting of some of the smart homes' parameters in the future based on historical data.

In this context, the initial objective is to train a Long-Short-Term-Memory (LSTM) model for predicting the power generated by the solar panels of smart homes in a decentralized federated framework (Fedra). The FL-trained model is then pruned to be more lightweight and used for inference purposes.

Regarding the federated learning process, each Client/Node (smart home) in the FL framework represents a smart home with a solar panel with its local dataset, as depicted in Figure 23. The LSTM model of each Client/Node is trained based on the local dataset and the model weights are aggregated in a federated model after several local rounds.

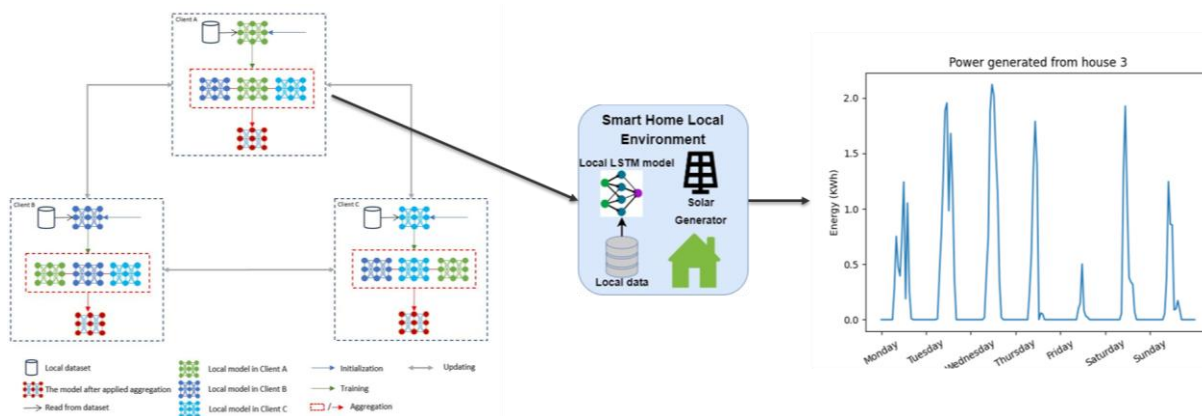


Figure 23: Representation of a smart home for the EDP NEW pilot

5.1.1 USING THE FEDRA TOOL

The configuration for **Fedra** is straightforward, requiring only the setup of node-specific parameters and network configurations. Configuration can be achieved through a simple configuration file (**node.conf**), which specifies the model parameters, training rounds, and P2P network settings. Figure 24 illustrates the configuration parameters, i.e., the minimum number of peers required to join in the framework and the P2P network configuration, the trained model

(dimensionality including hidden layers and number of neurons), as well as the learning parameters (number of federated rounds, epochs, etc.).

Figure 24: Configuration of the Fedra tool for the EDP NEW pilot

The developer can then initialize the FL training process by running the fedra.py, as depicted in Figure 25. Note that the process is waiting for the minimum number of peers to connect before starting the ML model training, printing also the status of the participating nodes.

Figure 25: Fedra training process initialisation for the EDP NEW pilot

When the required number of nodes enter the Fedra framework, the FL process starts. Figure 26 depicts the initiation of the first federated round, i.e., the participating nodes are performing local training for a configurable number of epochs.

```

PeerStatus updated for 1203KooNJC9UF3FK9j7z2n6J892sbvo3Ufpg5iEogFIEANuLYkEC: Status.JOINED
2024-07-18 10:08:33,593 - DEBUG - Deserializing data of size: 150 bytes
2024-07-18 10:08:33,594 - DEBUG - Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooNJC9UF3FK9j7z2n6J892sbvo3Ufpg5iEogFIEANuLYkEC: Status.JOINED
Starting training for LSTM_RES3 model
LSTM_RES3 DataLoader initialized with DE_KN_residential3_grid_export data.
LSTM_RES3 model initialized.
Round 1/5
2024-07-18 10:08:41,618 - DEBUG - Serialized data size: 150 bytes
2024-07-18 10:08:41,618 - DEBUG - Data chunked into 1 parts
2024-07-18 10:08:41,719 - DEBUG - Serialized data size: 150 bytes
2024-07-18 10:08:41,720 - DEBUG - Data chunked into 1 parts
2024-07-18 10:08:41,821 - DEBUG - Serialized data size: 150 bytes
2024-07-18 10:08:41,821 - DEBUG - Data chunked into 1 parts
2024-07-18 10:08:43,863 - DEBUG - Deserializing data of size: 150 bytes
2024-07-18 10:08:43,863 - DEBUG - Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooNJC9UF3FK9j7z2n6J892sbvo3Ufpg5iEogFIEANuLYkEC: Status.TRAINING
2024-07-18 10:08:43,964 - DEBUG - Deserializing data of size: 150 bytes
2024-07-18 10:08:43,965 - DEBUG - Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooNJC9UF3FK9j7z2n6J892sbvo3Ufpg5iEogFIEANuLYkEC: Status.TRAINING
2024-07-18 10:08:44,066 - DEBUG - Deserializing data of size: 150 bytes
2024-07-18 10:08:44,066 - DEBUG - Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooNJC9UF3FK9j7z2n6J892sbvo3Ufpg5iEogFIEANuLYkEC: Status.TRAINING
Serialized data size: 150 bytes
Data chunked into 1 parts
Serialized data size: 150 bytes
Data chunked into 1 parts
Deserializing data of size: 150 bytes
Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooND0vh3shwPmaa27J378Mlyx968ZBue8DdJxnD8jdxGyXF: Status.TRAINING
Deserializing data of size: 150 bytes
Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooND0vh3shwPmaa27J378Mlyx968ZBue8DdJxnD8jdxGyXF: Status.TRAINING
Deserializing data of size: 150 bytes
Deserialized data type: <class 'fedra.utils.state.PeerStatus'>
PeerStatus updated for 1203KooND0vh3shwPmaa27J378Mlyx968ZBue8DdJxnD8jdxGyXF: Status.TRAINING
Starting training for LSTM_RES4 model
LSTM_RES4 DataLoader initialized with DE_KN_residential4_grid_export data.
LSTM_RES4 model initialized.
Round 1/5
Serialized data size: 150 bytes
Data chunked into 1 parts
Serialized data size: 150 bytes
Data chunked into 1 parts
Serialized data size: 150 bytes
Data chunked into 1 parts
Serialized data size: 150 bytes
Data chunked into 1 parts
    
```

Figure 26: Initiation of the first FL round for the EDP NEW pilot

Once the FL process is finalized, the developer can visualize the performance results of the training, since Fedra automatically produces the metrics for the training and testing loss, as demonstrated in Figure 27.

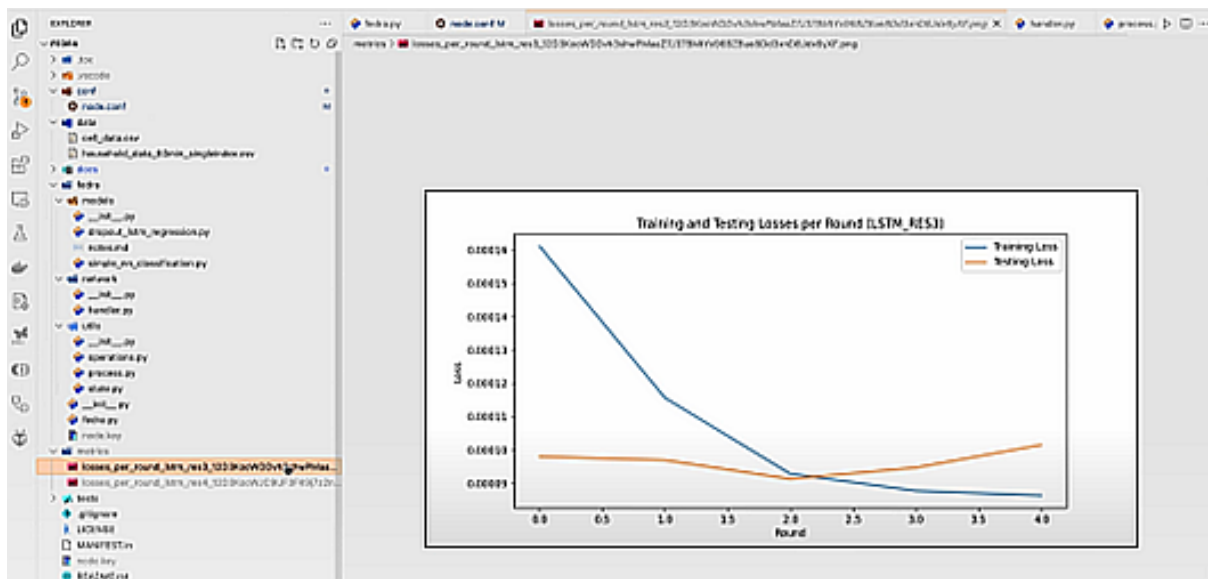


Figure 27: Fedra metrics for the EDP NEW pilot

5.1.2 PRUNING TOOL

The global FL model that has been trained via the Fedra tool can be then pruned to make it more lightweight. For this purpose, the pruning tools can be utilized, requiring from the user to configure at least the following parameters: (i) the ratio of sparsity to be applied to the model; (ii) the shape of the input tensor that the model expects. An example of using the pruning function is depicted in Figure 28.

```
SPARSE_RATIO = 0.5
INPUT_SHAPE = (1, 3, 32, 32) # including batch
model = myVGG().to(device)
new_model = prune_model(model, SPARSE_RATIO, INPUT_SHAPE)
```

Figure 28: The Pruning tool configuration for the EDP NEW pilot

By utilizing the pruning functionality several times, the following metrics can be obtained: (i) the required memory for the model inference; (ii) the inference latency; (iii) the Mean-squared error (MSE) of the pruned model. These metrics are showcased in Figure 29 for variable pruning rate in the LSTM forecasting scenario.

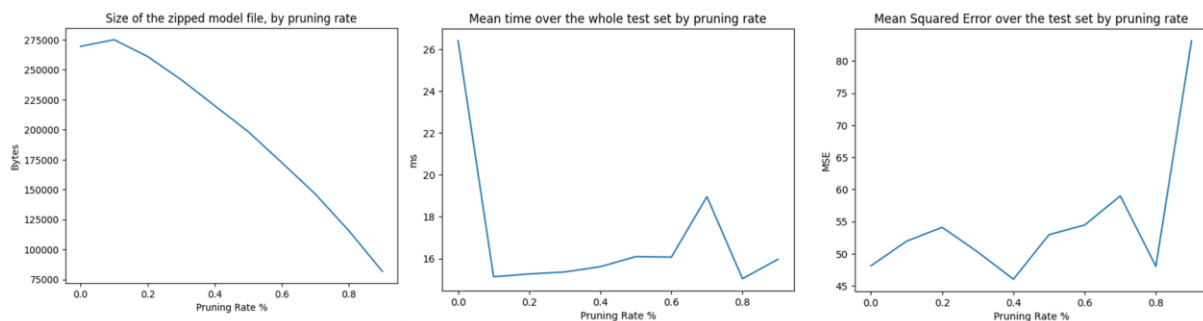


Figure 29: The TaRDIS Pruning tool metrics

The pruned ML models can be then used to estimate and visualize the accuracy of the prediction as depicted in Figure 30, i.e., the model that was pruned by 40% follows the trends of the original model, inheriting the abnormalities at 0 and 250 hours, whereas the model that was pruned by 90%, being the worst model steadily underpredicts the value of energy prediction.

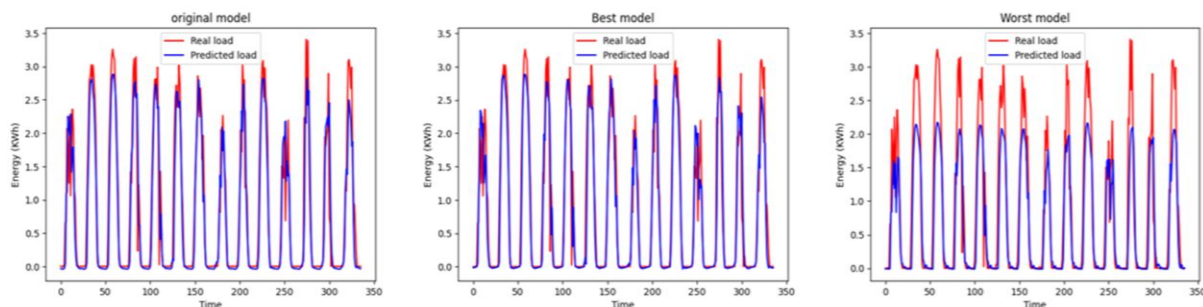


Figure 30: Accuracy of the Pruning activity

5.1.3 MEMBERSHIP SERVICE FOR ENERGY MARKETS

Membership abstractions allow applications to obtain information on the size of the surrounding swarm and the identities and health of its participants.

The TaRDIS WP6 is focused on devising a novel peer-to-peer overlay network distributed protocol that can generate and manage a specific membership management protocol and adapt to different conditions in the system. This membership abstraction can benefit from an additional hierarchical organization that will simplify some management activities in this open and decentralized renewable energy market.

The membership service will be used to model the connections among the different participants in the energy market and their respective communication, as well as allow layering of other tools on top of the established connections (i.e., ML applications).

The hierarchical structure of such a network tightly models the different roles in the energy market and their different interactions, while offering resilience in the presence of failures.

The service is being actively written in Babel, following the APIs described in the TaRDIS deliverable D3.3 [5], acting as a protocol which can be instantiated and added to the stack of protocols handling communication during the swarm deployment.

5.1.4 COMMUNITY ENERGY BALANCING APPLICATION

This developer story consists in creating a “bridge” application between the information that is gathered from the Fedra tool and the Pruning tool and the DCR choreographies. This application’s objective is to coordinate the different components: consumers, producers and the Community Orchestrator that are within an energy community. The coordination itself is based on trying to match everyone’s needs with the lowest possible cost for each one.

The developer’s approach is expected to be applicable to all energy management components. In this sense, the best reliable approach is to have a decision tree algorithm where a group of conditions are defined. Depending on the type of component, the corresponding DCR choreography is used within the component’s application. If it is a consumer, it will do a particular DCR choreography; if it is a producer, it will do other DCR choreography, and so on. By doing this, we can apply all choreographies to every single component. Based on either is a producer or consumer, hence the corresponding forecasted data that is available through the Fedra and Pruning Tools is ingested. This strategy even allows a component to assume a completely different role if needed, e.g., a producer transitioning into a Community Orchestrator.

The process is schedule driven. This means, the time scheduler triggers each single tool to be used, in this use case. After the schedule starts, the Fedra and Pruning tools will give the forecast data, for the next hour, for production and consumption for respectively each producer and consumer that exists in the energy community. Then, the community energy balancing application starts (bear in mind that this app is running on every single component/local machine, which in this case are Raspberry Pi’s). Through a decision tree algorithm, the

applicable choreography will run on that component, which, by consequence, will define the set of actions available for itself, as shown in Figure 31.

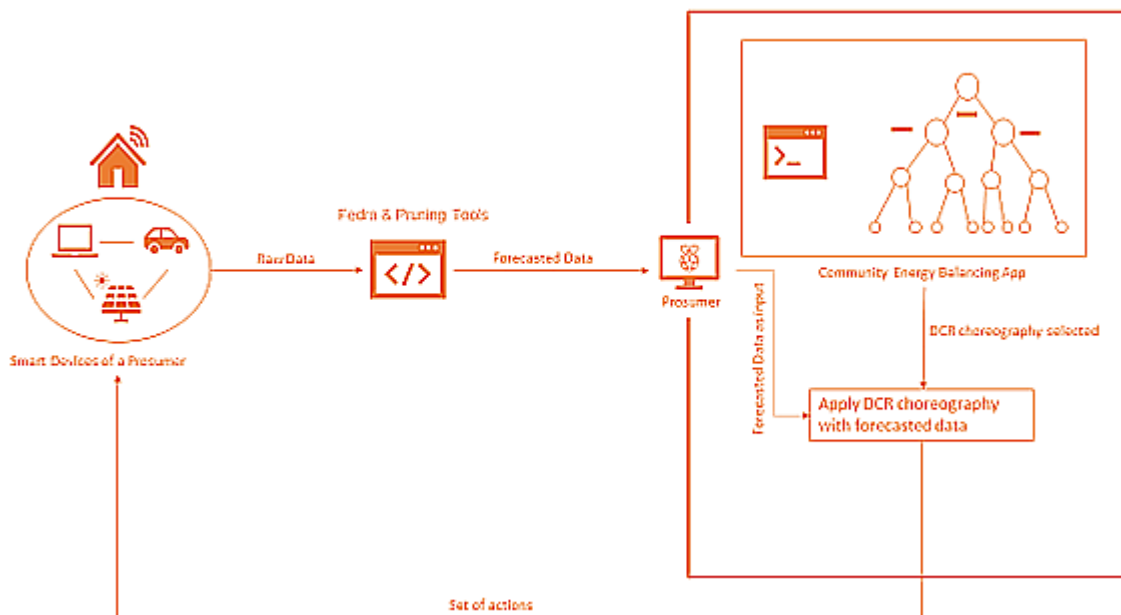


Figure 31: EDP NEW Community energy balancing application

5.2 GMV: DISTRIBUTED NAVIGATION CONCEPTS FOR LEO SATELLITES CONSTELLATIONS

5.2.1 MACHINE LEARNING PROCESS

This pilot project includes, as the other use cases, a Machine Learning process, composed by Federated Learning or Reinforcement Learning, which will be described in the last version of this deliverable, D3.6.

5.2.2 DECENTRALIZED STORAGE AND COMMUNICATION BETWEEN SATELLITES

The Babel framework is designed to execute swarm algorithms while emulating a distributed system using specialized protocols. Its primary function is to simulate the communication layer in detail, ensuring realistic testing scenarios for swarm algorithms. The baseline implementation assumes seamless information access for each node, without considering communication constraints. To enhance this model, developing a satellite simulation that accurately replicates key satellite characteristics is crucial. Babel's emulation layer focuses on simulating the connection between nodes and ensuring that each node can access the data (i.e., the values being calculated for ODTS) seamlessly.

Regarding the first, WP6 is actively developing a membership service for modelling visibility between nodes (i.e., satellites). This abstraction mimics the conditions of satellite connectivity

by calculating the visibility taking into account different factors that affect satellite positioning, such as altitude, distance to the ground station, etc. This aims to create a simulation as close as possible to a realistic deployment scenario.

With the proper membership service, nodes need to store and propagate locally calculated information with their respective “neighbours”. To achieve this, the team plans to use the TaRDIS tool **Nimbus**, a fully decentralized storage system providing scalable and efficient data storage without relying on a central authority, to synchronize data between all nodes, as depicted in the following illustration:

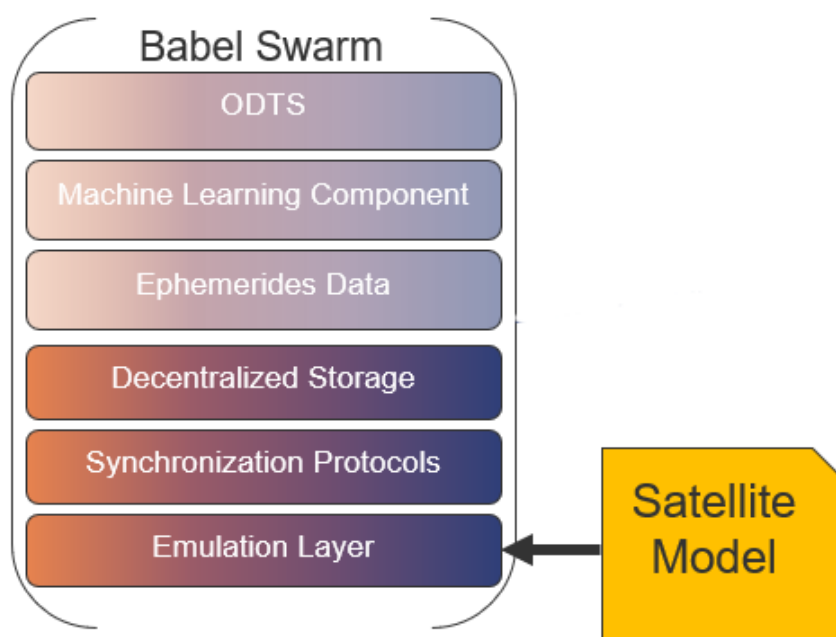


Figure 32: The TaRDIS Nimbus storage model on the GMV pilot

With the proper membership service modelling satellite communication and decentralized storage in place, each node will use the proper ODS algorithm and Machine Learning component especially handmade for this use case.

5.3 ACT: HIGHLY RESILIENT FACTORY SHOP FLOOR DIGITALISATION

The ACT use case implementation with the TaRDIS toolbox is expected to utilize a recent version of the Actyx machine-runner library (T-WP4-01) and machine-check tool (T-WP4-02), incorporating extensions and enhancements developed during the TaRDIS project. One such enhancement is the WorkflowEditor project ([T-WP3-01](#)), a Visual Studio Code extension for the graphical design and development of swarm protocols.

A developer will be able to open, under the TaRDIS IDE, a project based on the Actyx libraries (or create a new project, with the assistance of the TaRDIS IDE extensions). Such libraries require the developer to define a swarm protocol that describes the intended behaviour of swarm elements (the specifics of the protocol will be based on the use case requirements). This definition of the protocol can be performed manually (by editing TypeScript files and

defining suitable data structures) or graphically, using the WorkflowEditor extension for VS Code.

The WorkflowEditor extension helps the programmer's work by reading a (possibly empty) swarm protocol definition from the swarm application source code and visualising it; the user/programmer can then interact with the visualised protocol, e.g. by adding or removing protocol states, modifying transitions, etc.; the user/programmer can also use the WorkflowEditor to check whether the protocol is well formed. Finally, the programmer can save the protocol definition back into the application source code. The swarm protocol definition can be further modified again using the WorkflowEditor, or manually.

The user/programmer then writes the code that controls each element ("machine") in the swarm, using the Actyx machine-checker tool (possibly through the WorkflowEditor) to ensure that each swarm element behaves as required by the overall swarm protocol definition. Then, the code can be deployed and executed using the machine-runner tooling.

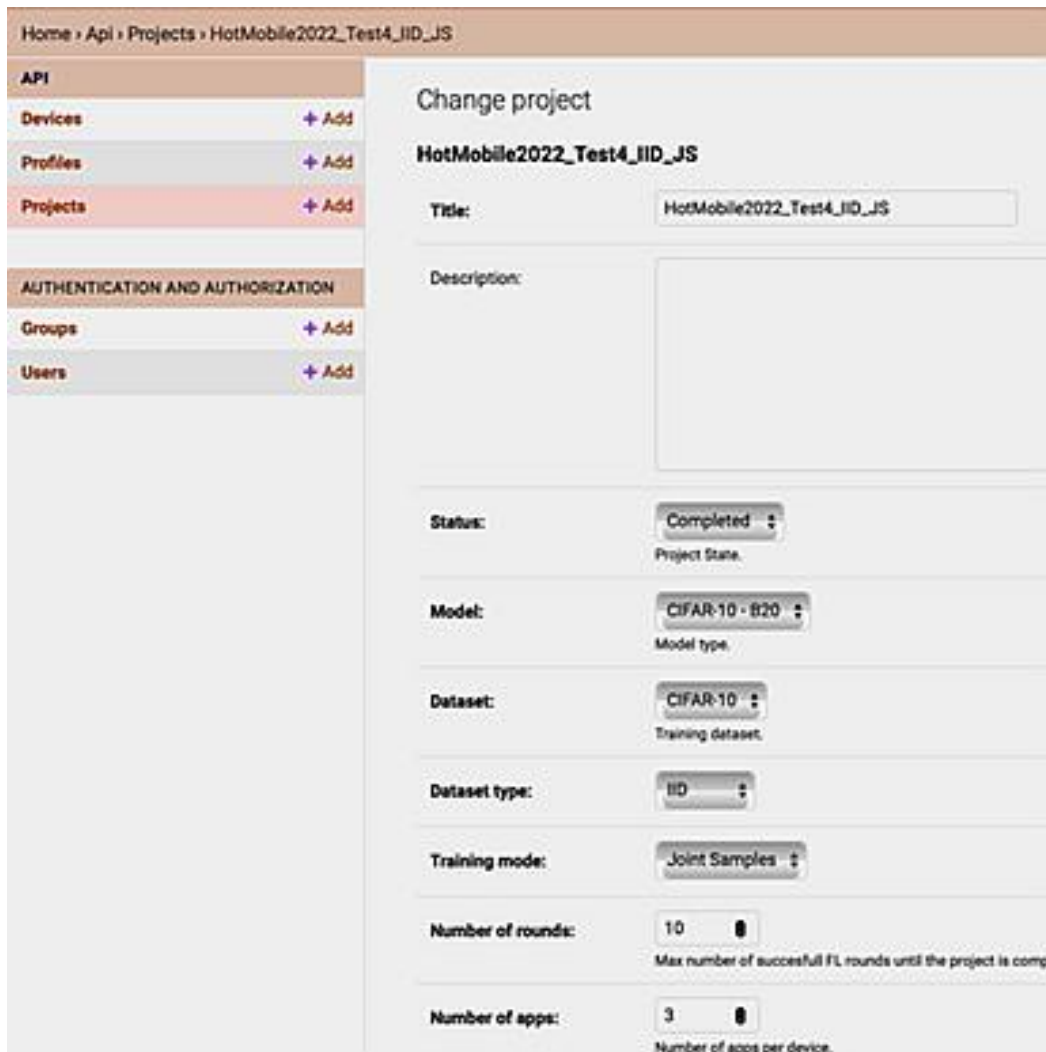
If the swarm protocol definition needs to be further changed later (e.g. to reflect new requirements from the factory), then the WorkflowEditor can be used to apply the changes, and the machine-checker tool can be used to verify whether the code of each swarm element aligns with the updated swarm protocol definition.

5.4 TID: PRIVACY-PRESERVING LEARNING THROUGH DECENTRALIZED TRAINING IN SMART HOMES

This section focuses on the developer story for the TID use case on intelligent homes. The use case centres on ML model training using Federated Learning (FL) or Split Learning (SL). Developers can leverage the Federated Learning as a Service (FLaaS) tool developed by TID. TaRDIS integration with FLaaS will introduce a privacy-preserving module and extend its learning capabilities to support SL. Additionally, plans include integrating an energy-efficient module from WP5 (e.g., the knowledge distillation API) and partially integrating Babel through a joint effort with WP6.

Figure 33 shows the administrator interface of FLaaS. The idea is that the developer can initiate an instance of FL through this interface in a way that hides the complexity of the task and the lines of codes necessary to be modified to capture the characteristics of every instance. In the baseline implementation of FLaaS, the developer can choose the characteristics of an FL instance through drop-down menus. For example, she/he can choose whether the "joint samples" or the "joint models" mode will be used during the training. Further details on these modes were provided in the TaRDIS Deliverable D7.1 [8].

Upon integration with a privacy preserving module, new drop-down menus could be added to tune this functionality, e.g., to choose between local or global privacy. Moreover, in the case of addition of differential privacy, there are additional parameters that should be tuned such as the privacy budget. This could be also the case for the SL functionality, where the developer could choose the learning method (FL or SL) through a drop-down menu, eliminating the need to navigate through blocks of code. Further details on this will be provided upon advancement of the use case implementation in the next deliverables.



Home » Api » Projects » HotMobile2022_Test4_IID_JS

API

- Devices + Add
- Profiles + Add
- Projects + Add

AUTHENTICATION AND AUTHORIZATION

- Groups + Add
- Users + Add

Change project

HotMobile2022_Test4_IID_JS

Title: HotMobile2022_Test4_IID_JS

Description:

Status: Completed
Project State.

Model: CIFAR-10 - 820
Model type.

Dataset: CIFAR-10
Training dataset.

Dataset type: IID

Training mode: Joint Samples

Number of rounds: 10
Max number of successful FL rounds until the project is comp

Number of apps: 3
Number of apps per device.

Figure 33: Administrator Interface for FLaaS in the TID pilot

Eventual Integration with Babel

As of this writing, TID and WP6 are working on integrating machine learning (ML) training in a partially decentralized setting, more precisely, Federated Learning (FL) and Split Learning (SL). This approach enables helpers to perform partial aggregation through peer-to-peer communication. The implementation will be conducted on Babel, through the implementation of different protocols for handling communication between nodes and the proper synchronisation of data.

6 CONCLUSIONS

This document presents the updated approach towards the definition of an Integrated Development Environment tool suited for the TaRDIS toolbox integration efforts.

In conclusion, the integration of Visual Studio Code (VS Code) with the toolset explored in the TaRDIS research project offers a highly effective and streamlined environment for software development, particularly in modern and dynamic swarm project contexts. By combining the rich set of features native to VS Code, such as its lightweight, customizable interface, with an array of specialized tools, this document has demonstrated a significant improvement in both developer productivity and workflow efficiency.

Throughout the study, it became evident that the combination of VS Code and the TaRDIS toolbox tools not only enhances code editing and debugging experiences but also fosters a smoother collaboration across various stages of development. The use of version control systems, integrated testing frameworks, and advanced extensions—combined with VS Code's seamless support for languages, frameworks, and debugging tools—has created an ecosystem that can adapt to a diverse range of project requirements and developer preferences.

Furthermore, the extension architecture in VS Code proved to be a key advantage, allowing for highly specific customization to match the unique demands of different project types. The ability to incorporate third-party tools, such as continuous integration/deployment pipelines, Docker containers, and cloud platforms, extends VS Code's capabilities, making it an ideal choice for developers working on projects of varying scope and complexity.

From a research perspective, the integration of these tools has highlighted areas of potential improvement in the development cycle, including streamlining the process for tool configuration, ensuring robust documentation for new users, availability of templates, examples and other support, and refining workflows for collaborative development. The overall results confirm that the synergies created by leveraging VS Code as the central hub for development significantly enhance the overall software development lifecycle.

Ultimately, this research reinforces the importance of using a flexible, well-supported, and extensible Integrated Development Environment (IDE) like Visual Studio Code as a highly added value in conjunction with the TaRDIS toolbox, as it fosters an environment conducive to innovation, rapid prototyping, and efficient code delivery in modern software projects. The outcomes suggest that further research into refining these integrations and expanding the set of supported tools will continue to improve both individual and team-based software development efforts in the future.

This document not only presented the customisation of the VS Code IDE to cope with the TaRDIS development needs, but also showed how the customisation and integration of each of the selected tools was performed, from the point of view of the integration process and from the point of view of the operational instructions.

As future work for the subsequent and final deliverable of this work-package, the team will continue to integrate the other TaRDIS tools as they are becoming finalised and available and will report the integration status and activities on the last deliverable D3.6.

This deliverable also introduces a developer's vision, where the team leverages their experience to present multiple developer stories from various perspectives. These stories aim to help readers relate to specific scenarios and provide guidance for tackling intelligent swarm challenges.

REFERENCES

- [1] D3.2 – First release of TaRDIS development environment, 2024, TaRDIS project, <https://www.project-tardis.eu/download/d3-2-integrated-development-environment>
- [2] Microsoft (n.d.). Code editing. Redefined. Visual Studio Code. Retrieved February 14, 2024, from <https://code.visualstudio.com>
- [3] Microsoft (2024, January 2). *Extension API*. Visual Studio Code. Retrieved February 21, 2024, from <https://code.visualstudio.com/api>
- [4] Git (n.d.). Local-branching-on-the-cheap. Retrieved November 17, 2024, from <https://git-scm.com>
- [5] D3.3 – Report on the 2nd iteration of the application model and APIs, 2024, TaRDIS project, <https://project-tardis.eu/download/d3-3-second-report-on-programming-model-and-apis>
- [6] D5.1 – Initial report on Distributed AI and AI-based orchestration, 2024, TaRDIS project, <https://www.project-tardis.eu/download/d5-1-initial-report-on-distributed-ai-and-ai-based-orchestration>
- [7] D5.2 – Second report on Distributed AI and AI-based orchestration, 2024, TaRDIS project, <https://www.project-tardis.eu/download/d5-2-second-report-on-distributed-ai-and-ai-based-orchestration>
- [8] D7.1 – Report on the expected improvements and quantification procedures, 2024, TaRDIS project, <https://www.project-tardis.eu/download/d7-1-report-on-the-expected-improvements-and-quantification-procedures>