



# D3.2: Integrated Development Environment

Revision: v.1

<b>Work package</b>	WP 3
<b>Task</b>	Task 3.3
<b>Due date</b>	2024-03-31
<b>Submission date</b>	2024-03-31
<b>Deliverable lead</b>	Carlos Coutinho (CMS)
<b>Version</b>	1
<b>Authors</b>	Carlos Coutinho (CMS), Carlos Reis (CMS) Roland Kuhn (ACT) Alceste Scalas (DTU) João Costa Seco (NOVA), António Ravara (NOVA) Miroslav Popovic (UNS)
<b>Reviewers</b>	Miodrag Djukic (UNS) Rafael Oliveira Rodrigues (EDP)
<b>Abstract</b>	This document presents an in-depth evaluation of the TaRDIS IDE platform, demonstrating its suitability for the development of the TaRDIS toolbox. Through a thorough examination, the document analyses the requirements and alternatives to select an IDE as the optimal choice for empowering the TaRDIS swarm development endeavours.
<b>Keywords</b>	Integrated Development Environment



## Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	2023-11-15	Document first draft	Carlos Coutinho (CMS), Carlos Reis (CMS)
V0.2	2024-03-20	First version for internal review	Carlos Coutinho (CMS), Carlos Reis (CMS)
V1	2024-03-31	Final Version to be submitted	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:	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 <a href="#">No2015/ 444</a>	
Classified C-UE/ EU-C	EU CONFIDENTIAL under the Commission Decision <a href="#">No2015/ 444</a>	
Classified S-UE/ EU-S	EU SECRET under the Commission Decision <a href="#">No2015/ 444</a>	

\* 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 integrating a set of tools on an Integrated Development Environment (IDE) to simplify the development of decentralized applications deployed in a diverse setting.

This document presents an in-depth evaluation 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. This document aims to demonstrate its suitability for the development of the TaRDIS toolbox. It includes an alternatives analysis on the most suitable candidates and the development of a customisation layer to ease the development of projects related to TaRDIS. Moreover, this document also includes a brief description of the main features and integration needs (still at an early stage) of the tools that are being developed in the project's WP4, WP5 and WP6, focusing essentially on their foreseen needs of user interface and integration.

## TABLE OF CONTENTS

Executive Summary .....	3
1 IDE Analysis.....	8
1.1 TaRDIS Requirements.....	8
1.2 IDE Alternatives Analysis.....	8
1.3 Discussion of Results.....	28
2 Introduction to the Eclipse IDE .....	30
2.1 Workspaces.....	30
2.2 Plug-in Development Environment.....	31
3 Eclipse Plug-in Integration for TaRDIS .....	32
3.1 Installation.....	32
3.2 Extending the TaRDIS Plug-in .....	32
3.3 How to use the TaRDIS Plug-in .....	33
3.4 Examples .....	35
4 TaRDIS Candidate Applications .....	37
4.1 WorkflowEditor (WP4).....	37
4.2 Ant (WP4).....	39
4.3 Data preparation for Flower-based FL model training (WP5).....	40
4.4 Flower-based FL model training (WP5).....	42
4.5 Flower-based FL model inference and evaluation (WP5).....	44
4.6 Early-Exit (Lightweight Functionality) (WP5) .....	45
4.7 Federated AI Network Orchestrator (WP5).....	47
4.8 IFChannel (WP4) .....	47
4.9 Java Typestate Checker (JaTyC) (WP4).....	48
4.10 Knowledge Distillation (Lightweight Functionality) (WP5).....	49
4.11 P4R-Type (WP4).....	50
4.12 Pruning (Lightweight Functionality) (WP5) .....	52
4.13 PSPSP (WP4).....	53
4.14 PTB-FLA-based FL or ODTs model training (WP5) .....	54
4.15 ReGraDaIFC (WP4).....	56
5 Conclusions.....	57

## LIST OF FIGURES

Figure 1: Customisation of Apache NetBeans	11
Figure 2: Apache NetBeans ProjectFactory	12
Figure 3: Customising Project Display on NetBeans	13
Figure 4: Generating a new project type in NetBeans	13
Figure 5: Creating a menu entry in IntelliJ IDEA	16
Figure 6: Registering a new class in the plugin.xml file	16
Figure 7: Creating a menu item in IntelliJ IDEA	16
Figure 8: Registering a menu item in IntelliJ IDEA	16
Figure 9: Creation of a ModuleBuilder class in IntelliJ IDEA	17
Figure 10: Creation of a ModuleType class in IntelliJ IDEA	17
Figure 11: Registering the classes in plugin.xml	18
Figure 12: Adding an image to the Eclipse project nature	20
Figure 13: Defining properties for creating a new nature in Eclipse	21
Figure 14: Extension setup in Eclipse	21
Figure 15: Definition of nature class in Eclipse	21
Figure 16: Defining the TaRDIS wizard in Eclipse	22
Figure 17: Defining the code for the wizard behaviour in Eclipse	22
Figure 18: Adding a project Wizard in Eclipse	23
Figure 19: Definition of the nature of the current Eclipse project.	23
Figure 20: Define a perspective in Eclipse	23
Figure 21: Code to use the defined perspective in Eclipse	24
Figure 22: Creating a separator for TaRDIS in VSCode	26
Figure 23: Add commands to the VSCode welcome views	26
Figure 24: VSCode configuration	26
Figure 25: Adding an action to the create button in VSCode	27
Figure 26: Registering configuration options in VSCode	27
Figure 27: Clean installation of Eclipse IDE with the TaRDIS plug-in	33
Figure 28: Wizard for creating a TaRDIS project	34
Figure 29: Menu for creating specific Babel Java classes	35
Figure 30: TaRDIS WorkFlowEditor	39
Figure 31: Data Preparation for Flower-Based FL Model Training	41
Figure 32: Data Preparation for Flower-Based FL Model Training Mock-up	42
Figure 33: Flower-Based FL Model Training	43
Figure 34: Flower-based FL model training Mock-ups	44
Figure 35: Flower-Based FL Model Inference and Evaluation	45
Figure 36: Flower-Based FL Model Inference and Evaluation Mock-ups	45
Figure 37: Example of a program using an API generated by P4R-Type	52
Figure 38: ODTs algorithm execution	56

**LIST OF TABLES**

Table 1: Comparison between selected IDEs .....28

## ABBREVIATIONS

<b>AI</b>	Artificial Intelligence
<b>API</b>	Application Programming Interfaces
<b>DNN</b>	Deep Neural Network
<b>FL</b>	Federated Learning
<b>GUI</b>	Graphical User Interface
<b>HMI</b>	Human-Machine Interface
<b>IDE</b>	Integrated Development Environment
<b>JAR</b>	Java Archive
<b>JaTyC</b>	Java Typestate Checker
<b>ML</b>	Machine Learning
<b>ODTS</b>	Orbit Determination and Time Synchronization
<b>OS</b>	Operating System
<b>OSGi</b>	Open Services Gateway Initiative
<b>PDE</b>	Plug-in Development Environment
<b>PTB-FLA</b>	Python TestBed for Federated Learning Algorithms
<b>SDK</b>	Software Development Kit
<b>SPMD</b>	Single Program Multiple Data
<b>TDM</b>	Time Division Multiplexing
<b>UI</b>	User Interface

## 1 IDE ANALYSIS

The definition of an Integrated Development 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.

### 1.1 TARDIS REQUIREMENTS

The TaRDIS project entails distinctive requirements, elicited in the project's deliverable D2.2, that entail a highly resilient and adaptable IDE. Key project specifications encompass communication, membership and storage abstractions, telemetry acquisition, configuration, and state management. The proposed IDE should exhibit broad OS support, efficiently handle numerous projects and files, offer comprehensive programming language support, and allow graphical customization to facilitate the creation of intuitive GUIs for developer assistance.

### 1.2 IDE ALTERNATIVES ANALYSIS

The choice of an IDE for the development of projects is essentially a personal choice, similar to the choice of a web browser or the choice of a text editor. Nevertheless, it was important to capture the state of the art in development to ensure the choice would be consistent with the opinion of a large group of developers. To make an educated analysis, the team selected a set of research articles which are not only relevant to the development discussion but are recent to ensure it follows the current trends in development.

Hence, [1] performs an interesting analysis over the choice of cloud-based IDEs versus the traditional “desktop” IDEs, stating that cloud-based IDEs may be helpful in supporting the integration of Artificial Intelligence (AI) and Machine Learning (ML). From the team's experience, many IDEs now support cloud-based or web-based IDEs, e.g., Eclipse CHE or Visual Studio Code are or have web-based interfaces. On the other hand, [2] analyses the migration from education towards development, focusing on the JetBrains platforms for supporting students to learn how to develop properly. In [3], the authors used the Jupyter platform integration with Visual Studio Code to teach programming to students, describing the benefits of this integration. Going on a more popular approach, numerous technical portals perform comparisons on these platforms. The KeyCDN blog [4] lists their top IDEs with an inclusion of Visual Studio Code, NetBeans, IntelliJ IDEA and Eclipse, among several other choices depending on the developed language. [5] also enumerates multiple IDEs according to the developed language, but for Java\*-like languages, the choice also falls on Eclipse, NetBeans, Visual Studio or IntelliJ IDEA. And while Stackify [6] made a large and comprehensive list of the 51 most popular IDEs, they did not rank the list or determine the best ones, just listed them, and showed some of their advantages and cons. Same with CTO [7], which lists a shorter list, including again tools such as Xcode, IntelliJ IDEA, Visual Studio Code, RubyMine, NetBeans or WebStorm. [8] also lists Eclipse, IntelliJ IDEA, Visual Studio Code, Xcode and Atom among their best choices for IDEs. GoodFirms [9] has an interesting approach



to defining common requirements and features that are desired on IDEs, then proceeding with the comparison of a list of IDEs including NetBeans, Eclipse, Xcode or IntelliJ IDEA among others. Many other sites such as GeeksforGeeks [10], TatvaSoft [11], Syndell [12], StudySmarter[13], TechBeamers [14], HackIO [15] or even Sourceforge [16] make similar lists, which helped a lot to determine a common understanding of which tools should be passed to a short list for better analysis and understanding.

Among all the open-source IDEs considered best-of-breed in the market, some prominent choices stand out, such as NetBeans, IntelliJ IDEA, Eclipse, and Visual Studio Code. Each of these IDEs caters to different needs and preferences, offering a diverse set of features, plugins, and customization options.

The following subsections present a short list of the most popular and prominent platforms, and the best candidates to be selected as the TaRDIS IDE. This analysis will comprehensively compare these IDEs, shedding light on their strengths and weaknesses.

### 1.2.1 Apache NetBeans

NetBeans is a free and open-source IDE maintained by The Apache Software Foundation [17].

#### Languages Support

NetBeans supports Java by default, but it can additionally support a wide variety of programming languages.

#### Customization

The IDE may be customised by creating NetBeans Modules.

#### Prerequisites

- Install NetBeans IDE

#### Pros

- Project management
- Static analysis tools
- GUI-Design Tool
- Stability
- Plugins
- Documentation
- Highly customizable

#### Cons

- Resource-intensive
- Community

#### Example: create a custom project type and a menu item

This example will create a new project type, with icons and a wizard.

Over this explanation, there is the need to add dependencies, something that can be done by selecting ALT+ENTER over the classes that are missing and selecting “Search Module Dependency” to import them.

#### Setup

After installing the NetBeans IDE, create a new project and select “Java with Ant” → “NetBeans Modules” → “Module”.

This creates a new module, which can be used to customize the IDE to support a different type of project.

## Define a project type

Defining a project type requires implementing the class `org.netbeans.api.project.Project`.

The example described in Figure 1 creates a class called “MyProject” which implements the class `Project` mentioned above:

```
public class MyProject implements Project {
    private final FileObject fo;
    private final ProjectState ps;
    private Lookup lkp;

    public MyProject(FileObject fo, ProjectState ps) {
        this.fo = fo;
        this.ps = ps;
    }

    @Override
    public FileObject getProjectDirectory() {
        return fo;
    }

    @Override
    public Lookup getLookup() {
        if (lkp == null) {
            lkp = Lookups.fixed(new Object[]{
                ps,
                new MyProjectInformation(),
                new MyProjectLogicalView(project: this)});
        }
        return lkp;
    }

    private final class MyProjectInformation implements ProjectInformation {

        public static final String ICON = "eu/tardis/module/tardis-logo.png";

        @Override
        public Icon getIcon() {
            return new ImageIcon(image: ImageUtilities.loadImage(resourceID: ICON));
        }

        @Override
        public String getName() {
            return getProjectDirectory().getName();
        }

        @Override
        public String getDisplayName() {
            return getName();
        }

        @Override
        public void addPropertyChangeListener(PropertyChangeListener pcl) {
        }

        @Override
        public void removePropertyChangeListener(PropertyChangeListener pcl) {
        }

        @Override
        public Project getProject() {
            return MyProject.this;
        }
    }
}
}
```

Figure 1: Customisation of Apache NetBeans

After this definition, the IDE must be instructed to recognize this project type. This is done by creating a `MyProjectFactory` class that implements `org.netbeans.spi.project.ProjectFactory`, as described in Figure 2:

```
@ServiceProvider(service = ProjectFactory.class)
public class MyProjectFactory implements ProjectFactory {

    public static final String PROJECT_FILE = ".tardis";

    @Override
    public boolean isProject(FileObject fo) {
        return fo.getFileObject(relativePath: PROJECT_FILE) != null;
    }

    @Override
    public Project loadProject(FileObject fo, ProjectState ps) throws IOException {
        return isProject(fo) ? new MyProject(fo, ps) : null;
    }

    @Override
    public void saveProject(Project prjct) throws IOException, ClassCastException {
    }
}
```

*Figure 2: Apache NetBeans ProjectFactory*

By doing this, every folder that has a “\*.tardis” file inside will be recognized as a TaRDIS project by NetBeans.

### Customize the project display on the IDE

Since the IDE recognizes the new project and can open it, the next step is to easily identify this project type in the projects list. This is done by adding a custom icon instead of the default folder icon.

In order to do that, it is necessary to create a class called “`MyProjectLogicalView`” that implements `org.netbeans.spi.project.ui.LogicalViewProvider`, as described in Figure 3:

```

public class MyProjectLogicalView implements LogicalViewProvider {
    private final MyProject project;

    public MyProjectLogicalView(MyProject project) {
        this.project = project;
    }

    @Override
    public Node createLogicalView() {
        try {
            final FileObject projectDirectory = project.getProjectDirectory();
            final DataFolder projectFolder = DataFolder.findFolder(fo: projectDirectory);
            final Node nodeOfProjectFolder = projectFolder.getNodeDelegate();
            return new MyProjectNode(node: nodeOfProjectFolder, project);
        } catch (DataObjectNotFoundException ex) {
            Logger.getLogger(name: MyProjectLogicalView.class.getName()).log(level: Level.SEVERE, msg: null, thrown: ex);
            return new AbstractNode(children: Children.LEAF);
        }
    }

    @Override
    public Node findPath(Node node, Object o) {
        return null;
    }

    private final class MyProjectNode extends FilterNode {

        public static final String ICON = "eu/tardis/module/tardis-logo.png";

        final MyProject project;

        public MyProjectNode(Node node, MyProject project)
            throws DataObjectNotFoundException {
            super(original: node,
                new FilterNode.Children(or: node),
                new ProxyLookup(
                    new Lookup[]{
                        Lookups.singleton(objectToLookup: project),
                        node.getLookup()
                    }));
            this.project = project;
        }

        @Override
        public Image getIcon(int type) {
            return ImageUtilities.loadImage(resourceID: ICON);
        }

        @Override
        public Image getOpenedIcon(int type) {
            return getIcon(type);
        }
    }
}

```

Figure 3: Customising Project Display on NetBeans

## Create a wizard to generate the new project type

To add a custom entry to the new project categories, it is required to define that new category. To do that, create an XML layer and add the code in Figure 4 inside the filesystem tag:

```

<folder name="Templates">
  <folder name="Project">
    <folder name="TaRDIS"/>
  </folder>
</folder>

```

Figure 4: Generating a new project type in NetBeans

To create the wizard, NetBeans has an easy way of doing it:

1. Create a sample project including the “.tardis” file and other files/folders.
2. Run the defined module.
3. Open the sample project.
4. On the module project, add a new “Project Template”.
5. Select the sample project and indicate the category and the name of the template.
6. Customize the wizard, if needed.
7. Close the running instance of the module.
8. Run it again and create a new project with the sample and wizard just created.

## 1.2.2 JetBrains IntelliJ IDEA Community Edition

While IntelliJ IDEA is a very complete development environment it requires a paid licence. However, the project includes a Community Edition which is built on open-source code, providing essential features for Java and Kotlin enthusiasts [18]. It is also the basis for the popular IDE Android Studio for the development of mobile applications.

### Languages Support

It supports Java, Groovy, and Kotlin out of the box, but also Scala, Python, Rust, and Dart via plugins. It also supports XML, JSON, YAML, XSLT, XPath, and Markdown.

### Customization

We can create custom plugins for this IDE using the Java programming language alongside IntelliJ Platform SDK.

### Prerequisites

- Install IntelliJ Idea Community Edition.
- Create an IDE Plugin Project.
- Update dependencies and install missing ones (Java, Gradle, etc.).
- Create the necessary classes, extend and register them to customize the IDE.

### Pros

- Intelligent Code Assistant
- Build Tools
- Refactoring Tools
- Debugging Tools
- Documentation
- It is straightforward to create a plugin

### Cons

- Resource-intensive
- Lots of features and support are only available in the paid version
- Not very customizable

### Example: create a plug-in

This example creates a plugin to add a new menu entry and a menu item and create a new project type:

## Create a menu entry

To create a menu entry on the main menu of the IDE, it is required to create a class and extend it like described in Figure 5:

```
public class TardisDefaultActionGroup extends DefaultActionGroup {
    @Override
    public void update(AnActionEvent event) {
        // Set the availability based on whether a project is open
        Project project = event.getProject();
        event.getPresentation().setEnabledAndVisible(project != null);
    }

    @Override
    public @NotNull ActionUpdateThread getActionUpdateThread() { return ActionUpdateThread.BGT; }
}
```

Figure 5: Creating a menu entry in IntelliJ IDEA

After the class is created, it needs to be registered in the plugin.xml file inside the actions tag, as depicted in Figure 6:

```
<group id="eu.tardis.plugin.TardisDefaultActionGroup" class="eu.tardis.plugin.TardisDefaultActionGroup"
    text="TaRDIS" popup="false">
    <add-to-group group-id="MainMenu" anchor="last"/>
</group>
```

Figure 6: Registering a new class in the plugin.xml file

## Create a menu item

This action will create a menu item inside the menu group created previously, creating a class like described in Figure 7.

```
public class PopupDialogAction extends AnAction {
    @Override
    public void actionPerformed(@NotNull AnActionEvent e) {
        Messages.showMessageDialog("Hello, TaRDIS!", "TaRDIS", null);
    }
}
```

Figure 7: Creating a menu item in IntelliJ IDEA

The next action is to register the menu item inside the group previously registered, in the **plugin.xml** file, as seen in Figure 8:

```
<group id="eu.tardis.plugin.TardisDefaultActionGroup" class="eu.tardis.plugin.TardisDefaultActionGroup"
    text="TaRDIS" popup="false">
    <add-to-group group-id="MainMenu" anchor="last"/>
    <action id="eu.tardis.plugin.PopupDialogAction" class="eu.tardis.plugin.PopupDialogAction"
        text="Hello"/>
</group>
```

Figure 8: Registering a menu item in IntelliJ IDEA



## Create a project type

Creating a project type requires creating a module builder and a module type.

The custom module will be displayed on the new project popup, and its creation needs the definition of a `ModuleBuilder` class (see Figure 9) and of a `ModuleType` class (see Figure 10).

```
public class TardisModuleBuilder extends ModuleBuilder {
    @Override
    public ModuleType<TardisModuleBuilder> getModuleType() { return TardisModuleType.getInstance(); }
}
```

Figure 9: Creation of a `ModuleBuilder` class in IntelliJ IDEA

```
public class TardisModuleType extends ModuleType<TardisModuleBuilder> {
    private static final String ID = "eu.tardis.plugin.TardisModuleType";

    public TardisModuleType() {
        super(ID);
    }

    public static TardisModuleType getInstance() {
        return (TardisModuleType) ModuleTypeManager.getInstance().findById(ID);
    }

    @Override
    public @NotNull TardisModuleBuilder createModuleBuilder() {
        return new TardisModuleBuilder();
    }

    @Override
    public @NotNull @Nls(capitalization = Nls.Capitalization.Title) String getName() {
        return "TaRDIS";
    }

    @Override
    public @NotNull @Nls(capitalization = Nls.Capitalization.Sentence) String getDescription() {
        return "TaRDIS Project";
    }

    @Override
    public @NotNull Icon getNodeIcon(boolean isOpened) {
        final URL url = TardisModuleType.class.getResource("/assets/tardis-logo.png");
        assert url != null;
        return IconLoader.getIcon(new File(url.getFile()).getAbsolutePath(), TardisModuleType.class);
    }

    @Override
    public ModuleWizardStep @NotNull [] createWizardSteps(@NotNull WizardContext wizardContext, @NotNull TardisModuleBuilder moduleBuilder, @NotNull ModulesProvider modulesProvider) {
        return super.createWizardSteps(wizardContext, moduleBuilder, modulesProvider);
    }
}
```

Figure 10: Creation of a `ModuleType` class in IntelliJ IDEA

To conclude the process, it is needed to register both these classes inside the extensions on the **plugin.xml** file, as described in Figure 11:

```
<!-- Extension points defined by the plugin.
|   Read more: https://plugins.jetbrains.com/docs/intellij/plugin-extension-points.html -->
<extensions defaultExtensionNs="com.intellij">
|   <moduleType id="eu.tardis.plugin.TardisModuleType" implementationClass="eu.tardis.plugin.TardisModuleType"/>
|   <moduleBuilder id="eu.tardis.plugin.TardisModuleBuilder" builderClass="eu.tardis.plugin.TardisModuleBuilder"
|       order="first"/>
</extensions>
```

*Figure 11: Registering the classes in plugin.xml*

### 1.2.3 Eclipse

Eclipse is a free and open-source IDE (Eclipse Licensed) maintained by the Eclipse Foundation [19]. For years it has been considered one of the most complete IDEs in the market and it has numerous customisations and plugins developed for various purposes.

#### Languages Support

Eclipse supports Java by default but can be extended by creating custom plugins to support a different language or by using already developed plugins.

#### Customization

We can use Plug-in Development Environment (PDE) which provides the proper tools to extend the IDE.

The Eclipse 4 developed plugins are incompatible with project Eclipse Che because Che has a new code base.

#### Prerequisites

- Install Eclipse IDE with PDE integrated
- Check for updates
- Create a Plugin-Project
- Add the desired dependencies
- Add the desired extensions
- Customize the IDE by adding extension points

#### Reading and Saving Data

There are several options for saving and reading data. The recommended way is using the Eclipse runtime preferences, although we can use files and databases.

Regarding the runtime preferences, there are scopes. Scopes enable us developers to indicate if the data should be stored related to the workspace, project, etc.

We have the Configuration Scope which is used to save data across multiple workspaces, Instance Scope to save data only for a workspace, Default Scope (not saved on disk) is used to indicate default values and when the data is not found in other scopes it uses the default ones and finally the Project Scope that is used to save data in a single project.

#### Pros

- Language support
- Code editing and code refactoring
- Performance
- Community
- Integrations

- Highly customizable

### Cons

- Resource-intensive
- Somewhat steep learning curve

### Example: create a plug-in

This example creates a plugin that adds a menu entry for TaRDIS, a view for settings, a new project type, a project wizard creator, and a new perspective.

### Setup

After following the steps on Prerequisites to create a Plugin Project make sure to add the following dependencies:

1. **Packages:** javax.annotation, org.eclipse.ui, org.eclipse.ui.actions
2. **Plug-ins:** org.eclipse.core.runtime, org.eclipse.ui, org.eclipse.ui.resources, org.eclipse.ui.forms

These packages and plug-ins will enable us to extend the IDE interface and access the SDK tools to add logic to it.

### Create a Project Nature

The project nature indicates the type of project that will be created (e.g., Java, TypeScript). As the target here is to create a new type, it requires creating a project nature. This action requires defining two extensions, named **org.eclipse.core.resources.natures** and **org.eclipse.ui.ide.projectNatureImages**.

To add an image for the project nature just right-click on the extension and select New → image, and apply these configurations (see Figure 12), ensuring that the icon image is stored inside the plugin project:

**Extension Element Details**

Set the properties of 'image' Required fields are denoted by '\*'.

id*:	<input type="text" value="eu.tardis.natures.images.tardis"/>
natureId*:	<input type="text" value="eu.tardis.natures.tardis"/> <input type="button" value="Browse..."/>
icon*:	<input type="text" value="assets/tardis-logo.jpg"/> <input type="button" value="Browse..."/>

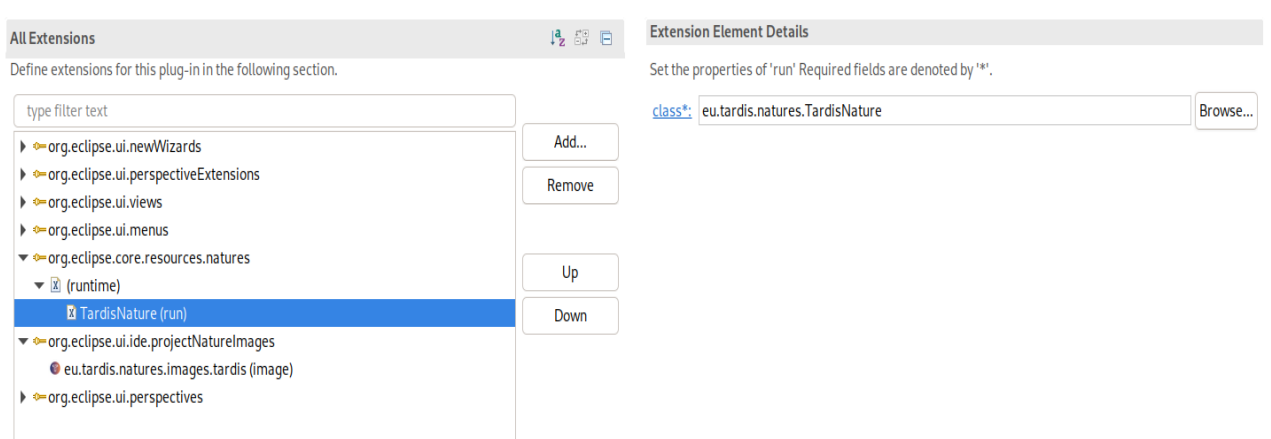
*Figure 12: Adding an image to the Eclipse project nature*

The next step is about creating a new nature, so select the org.eclipse.core.resources.natures and set the values on Figure 13:

Extension Details	
Set the properties of the selected extension. Required fields are denoted by "*".	
ID*:	eu.tardis.natures.tardis
Name:	TaRDIS Nature

Figure 13: Defining properties for creating a new nature in Eclipse

The following step is to open this extension, making sure to have it setup like the definition in Figure 14:



The screenshot shows the Eclipse IDE interface. On the left, the 'All Extensions' view displays a tree of extensions. The extension 'TardisNature (run)' is selected. On the right, the 'Extension Element Details' view shows the 'class\*' field set to 'eu.tardis.natures.TardisNature'.

Figure 14: Extension setup in Eclipse

Then, click on 'class\*:' to generate the class file and edit it to become like the one in Figure 15:

```

1 package eu.tardis.natures;
2
3 import org.eclipse.core.resources.IProject;
4
5
6
7 public class TardisNature implements IProjectNature {
8
9     public static final String NATURE_ID = "eu.tardis.natures.tardis";
10
11     private IProject project;
12
13     @Override
14     public void configure() throws CoreException {
15     }
16
17     @Override
18     public void deconfigure() throws CoreException {
19     }
20
21     @Override
22     public IProject getProject() {
23         return this.project;
24     }
25
26     @Override
27     public void setProject(IProject project) {
28         this.project = project;
29     }
30
31 }
32

```

Figure 15: Definition of nature class in Eclipse

## Create a custom project with the custom nature defined

This example creates a wizard and a page for the wizard to enable the creation of a TaRDIS nature project.

This starts by creating a wizard page. To do that, create a new class that extends the **org.eclipse.jface.wizard.WizardPage**, and then, on the createControl method, just set the form needed to create the project. With such a page created, the next step is to create the wizard. This is performed by adding the extension **org.eclipse.ui.newWizards**, and creating a wizard with this configuration as described in Figure 16:

### Extension Element Details

Set the properties of 'wizard' Required fields are denoted by '\*'.

id*:	<input type="text" value="eu.tardis.wizards.newproject"/>
<a href="#">name*</a> :	<input type="text" value="Create a TaRDIS Project"/>
<a href="#">class*</a> :	<input type="text" value="eu.tardis.wizards.NewProjectWizard"/> <input type="button" value="Browse..."/>
<a href="#">icon</a> :	<input type="text" value="assets/tardis-logo.jpg"/> <input type="button" value="Browse..."/>
category:	<input type="text" value="eu.tardis.wizards.categories.tardis"/>
project:	<input type="text" value="true"/> <input type="button" value="v"/>
<a href="#">finalPerspective</a> :	<input type="text"/> <input type="button" value="Browse..."/>
preferredPerspectives:	<input type="text"/>
helpHref:	<input type="text"/>
<a href="#">descriptionImage</a> :	<input type="text"/> <input type="button" value="Browse..."/>
canFinishEarly:	<input type="text" value="false"/> <input type="button" value="v"/>
hasPages:	<input type="text" value="true"/> <input type="button" value="v"/>

Figure 16: Defining the TaRDIS wizard in Eclipse

Then click on the 'class\*:' link to generate the code that is needed. This class has a function called **performFinish**, which is the handler when the finish button is clicked, so it's the proper place to create a project and add the code in Figure 17:

```
final IProject project = ResourcesPlugin.getWorkspace().getRoot().getProject(projectName);
project.create(monitor);
project.open(monitor);
```

Figure 17: Defining the code for the wizard behaviour in Eclipse

Then, it is necessary to add the page that contains the form to create the project, instantiating the page and adding it in the wizard constructor, as described in Figure 18:

```
this.page1 = new NewProjectWizardPage("Create a TaRDIS Project");
addPage(this.page1);
```

Figure 18: Adding a project Wizard in Eclipse

The next step is to indicate the nature of this project. To do that, on the creation method add the code defined in Figure 19:

```
final IProjectDescription description = project.getDescription();
final String[] natures = description.getNatureIds();
final String[] newNatures = new String[natures.length + 1];
System.arraycopy(natures, 0, newNatures, 0, natures.length);
newNatures[natures.length] = TardisNature.NATURE_ID;
description.setNatureIds(newNatures);
project.setDescription(description, monitor);
```

Figure 19: Definition of the nature of the current Eclipse project.

## Create a custom perspective

Perspectives are elements that enable developers to customize the IDE layout. To create a perspective, add the extension **org.eclipse.ui.perspectives**, and add a new extension with the configuration in Figure 20:

**Extension Element Details**

Set the properties of 'perspective' Required fields are denoted by '\*'.

<b>id*:</b>	<input type="text" value="eu.tardis.perspectives.tardis"/>		
<b>name*:</b>	<input type="text" value="TaRDIS Perspective"/>		
<b>class*:</b>	<input type="text" value="eu.tardis.eu.tardis.perspectives.TardisPerspective"/>	<input type="button" value="Browse..."/>	
<b>icon:</b>	<input type="text" value="assets/tardis-logo.jpg"/>	<input type="button" value="Browse..."/>	
<b>fixed:</b>	<input type="checkbox"/>		
<b>defaultShowIn:</b>	<input type="text"/>	<input type="button" value="Browse..."/>	

Figure 20: Define a perspective in Eclipse

Click on the 'class\*:' link to create the code and configure the layout on the `createInitialLayout` method. The final step is to indicate that when a project is created, the IDE should change to

this perspective. To have that behaviour add the following code after the project creation (see Figure 21):

```
Display.getDefault().syncExec(new Runnable() {
    public void run() {
        final IWorkbenchWindow workbenchWindow = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
        try {
            workbenchWindow.getWorkbench().showPerspective(TardisPerspective.PERSPECTIVE_ID, workbenchWindow);
        } catch (WorkbenchException e) {
            e.printStackTrace();
        }
    }
});
```

*Figure 21: Code to use the defined perspective in Eclipse*



### 1.2.4 Microsoft Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs on a desktop and is available for Windows, MacOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, C#, Java, Python, PHP, Go, .NET) [20]. This application 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 [21].

#### Languages Support

It can support all languages, either by using an already existing extension or by creating your own.

#### Customization

You can customize your editor with extensions, snippets, themes, languages support, keymaps and notebook renderers. All of this is 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

#### Pros

- Easy customization
- Community
- Extensions
- Lightweight and fast
- Cross-platform
- Intelligent code completion
- Debugger support

#### Cons

Not a full IDE for all languages, i.e., it relies on being a flexible editing platform with a rich marketplace of extensions, some of which give the platform the ability to become an IDE for certain specific languages.

### Example: create a plug-in

This example creates a plug-in to add a view with the TaRDIS icon, and 3 buttons to create, open or clone a project.

#### Create a TaRDIS separator

To add a separator for TaRDIS we need to add the following structure to the **package.json** file inside the 'contributes' tag, as seen in Figure 22:

```
"viewsContainers": {
  "activitybar": [
    {
      "id": "tardis",
      "title": "TaRDIS",
      "icon": "assets/activity-icon.png"
    }
  ]
},
```

Figure 22: Creating a separator for TaRDIS in VSCode

#### Create a view for the separator

The separator needs to render a view. Therefore, in this case, the strategy is to use the **welcome** views, adding the following configuration (see Figure 23):

```
"views welcome": [
  {
    "view": "tardis-projects",
    "Contents": "Get started with [TaRDIS](https://www.project-tardis.eu) by choosing any of the following actions.\n[Create Project](command:tardis.commands.create-project)\n[Clone Repository](command:git.clone)\n[Open Project](command:vscode.openFolder)"
  }
],
```

Figure 23: Add commands to the VSCode welcome views

Then, indicate that this view will be rendered inside the separator (see Figure 24):

```
"views": {
  "tardis": [
    {
      "id": "tardis-projects",
      "name": "Projects"
    }
  ]
}
```

Figure 24: VSCode configuration

## Define and register a command

It is simple to configure a create button to define and register a command. It is just a matter of configuring it in the VSCode configuration (see Figure 25):

```
"commands": [  
  {  
    "command": "tardis.commands.create-project",  
    "title": "TaRDIS: Create Project"  
  }  
],
```

Figure 25: Adding an action to the create button in VSCode

The final step is to register these changes in the **extension.ts** file, as depicted in Figure 26:

```
import * as vscode from "vscode";  
import * as fs from "fs";  
  
export function activate(context: vscode.ExtensionContext) {  
  // Register Create Project Command  
  context.subscriptions.push(  
    vscode.commands.registerCommand(  
      "tardis.commands.create-project",  
      async () => {  
        const projectName: string | undefined =  
          await vscode.window.showInputBox({  
            placeholder: "TaRDIS project name",  
            validateInput: (value: string) => {  
              return value.trim().length > 0  
                ? null  
                : "Enter a valid project name";  
            },  
          });  
        if (projectName) {  
          const projectLocation: vscode.Uri[] | undefined =  
            await vscode.window.showOpenDialog({  
              title: "Select the TaRDIS project location",  
              openLabel: "Select",  
              canSelectFiles: false,  
              canSelectFolders: true,  
              canSelectMany: false,  
            });  
          if (projectLocation && projectLocation.length > 0) {  
            const projectFolder: string =  
              projectLocation[0].fsPath + `/${projectName}`;  
            if (fs.existsSync(projectFolder)) {  
              vscode.window.showErrorMessage(  
                `TaRDIS project destination already exists.`  
              );  
            } else {  
              fs.mkdirSync(projectFolder);  
              vscode.commands.executeCommand(  
                `vscode.openFolder`,  
                vscode.Uri.parse(projectFolder)  
              );  
            }  
          } else {  
            vscode.window.showErrorMessage(`TaRDIS project not created.`);  
          }  
        } else {  
          vscode.window.showErrorMessage(`TaRDIS project not created.`);  
        }  
      }  
    )  
  );  
}  
  
export function deactivate() {}
```

Figure 26: Registering configuration options in VSCode

### 1.3 DISCUSSION OF RESULTS

After thoroughly evaluating the four major IDEs, IntelliJ Idea Community Edition was excluded from consideration due to limitations in supported languages and essential features behind a paywall. The focus then shifted to NetBeans, Eclipse and Visual Studio Code, which are compared in Table 1 regarding their most relevant features.

*Table 1: Comparison between selected IDEs*

Feature	Eclipse	NetBeans	Visual Studio Code
<b>Community Support</b>	Strong, established community with a wealth of plugins and community-driven resources	Solid community but not as extensive as Eclipse's	Active community, but Eclipse has a more robust and long-standing presence
<b>Customization</b>	Excels in customization, with a plugin architecture that supports extensive tailoring of the IDE	Decent customization options but may not be as feature rich as Eclipse	Good customization through extensions but may not match Eclipse's level of features
<b>Documentation</b>	Offers extensive and detailed documentation, facilitating issue troubleshooting	Comprehensive documentation but may not be as extensive as Eclipse's	Well-documented with a vast array of tutorials and guides.
<b>Flexibility</b>	Highly flexible and extensible, excelling in allowing developers to adapt the IDE to their workflow	Flexible but not as versatile as Eclipse in terms of deep customization	Flexible and lightweight but may lack some advanced features
<b>Language Support</b>	Strong language support, covering a broad spectrum of programming languages	Good language support but may not cover as many languages as Eclipse	Excellent language support with a wide range of extensions.
<b>Ecosystem and Integration</b>	Well-established ecosystem with support for numerous plugins and integrations	Good integration capabilities but might not have the extensive ecosystem found in other IDEs	Part of the Microsoft ecosystem, providing seamless integration with other Microsoft tools

Considering all the above factors, with special reference to the community support, extensive customization, thorough documentation, flexibility, and broad language support are essential criteria, Eclipse emerges as a strong choice for implementing development projects like the TaRDIS toolbox. Its long-standing reputation and feature-rich environment make it a reliable and robust IDE for a wide range of development tasks. Additionally, it has been the choice for developers since a long time ago, becoming well-established in the development communities.

Nevertheless, in most recent years, VSCode 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.

The initial stage of providing an IDE for developers to integrate the TaRDIS toolbox will therefore be based on the Eclipse platform, which will be described in the following sections of the document. In a follow-up deliverable, the team intends to include a version of the toolbox customisation for the VSCode platform, and, as best as possible, try to cope with both platforms on the development and integration of the TaRDIS toolbox modules.

## 2 INTRODUCTION TO THE ECLIPSE IDE

In the ever-evolving landscape of software development, choosing the right IDE can significantly impact a developer's productivity, code quality, and overall development experience.

Among the array of open-source IDEs available, Eclipse emerges as a standout choice. With its robust features, extensive plug-in ecosystem, and active community support, Eclipse has solidified its position as a preferred tool for developers across various domains.

In this dynamic environment, the TaRDIS project adds a compelling dimension to Eclipse's capabilities. Leveraging the IDE's versatile plug-in ecosystem, the TaRDIS project aims to extend Eclipse and tailor it to the specific needs of the project's toolbox.

### 2.1 WORKSPACES

Eclipse workspaces play a pivotal role in facilitating efficient project management and development workflows. A workspace in Eclipse is essentially a directory on the file system where all projects, configurations, and metadata are stored. This concept is particularly well-suited for aggregating several projects seamlessly [22].

The following subsections present a breakdown of how Eclipse workspaces are advantageous for aggregating multiple projects.

#### Organizational Structure

Eclipse workspaces provide a structured and organized environment for managing multiple projects. Each project resides within the workspace, allowing developers to compartmentalize and maintain a clear hierarchy.

#### Unified Development Environment

By utilizing a single workspace, developers can create a unified development environment where all related projects coexist. This ensures consistency in settings, configurations, and project dependencies.

#### Shared Resources

Workspaces allow projects to share resources efficiently. Common libraries, configuration files, and other shared assets can be managed centrally within the workspace, streamlining maintenance and updates across multiple projects.

#### Cross-Project Navigation

Eclipse offers features for seamless navigation between projects within the same workspace. Developers can easily switch between projects, view source code, and manage dependencies without having to open separate instances of the IDE.

## Build and Deployment Coordination

Eclipse workspaces facilitate coordinated build and deployment processes across multiple projects. Developers can define build paths, manage dependencies, and ensure that changes in one project are reflected appropriately in others within the same workspace.

## Search and Refactoring

Eclipse provides powerful search and refactoring tools that operate at the workspace level. This allows developers to perform comprehensive searches across all projects, making it easier to identify and modify code elements consistently.

## Preferences

Eclipse workspaces allow developers to set preferences at the workspace level. This is beneficial when there is a need for uniform settings or configurations across multiple projects, ensuring consistency in development practices.

## Collaboration and Version Control

Workspaces seamlessly integrate with version control systems, enabling collaborative development on projects stored in repositories. Developers can share workspaces, making it easier to collaborate on multiple projects concurrently.

## 2.2 PLUG-IN DEVELOPMENT ENVIRONMENT

The Plug-in Development Environment provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP products.

PDE also provides comprehensive OSGi tooling, which makes it an ideal environment for component programming, not just Eclipse plug-in development. [23]

### Components

The PDE offers a comprehensive suite of tools to empower developers in crafting their own plug-ins. These tools encompass UI components, APIs, Build utilities, and an Incubator.

#### UI Components

Models, builders, editors and more to facilitate plug-in development in the Eclipse IDE. [23]

#### APIs

Eclipse IDE and build process integrated tooling to maintain API. [23]

#### Build Utilities

Ant based tools and scripts to automate build processes. [23]

#### Incubator

Development of new tools that are not ready to be added to the Eclipse SDK. [23]

## 3 ECLIPSE PLUG-IN INTEGRATION FOR TARDIS

### 3.1 INSTALLATION

Eclipse offers users multiple avenues for installing the TaRDIS plug-in. The developer can choose to install it through the Eclipse Marketplace or manually add the plug-in file to the “**dropins**” folder within the IDE installation directory. It's essential to restart the IDE to apply the changes after the installation.

Note: Make sure you have a compatible IDE version.

#### Marketplace

- Open Eclipse IDE and navigate to the “**Help**” menu.
- Select “b” from the dropdown.
- In the Marketplace dialog, search for “**TaRDIS**” in the search bar.
- Locate the TaRDIS plug-in in the search results and click the “**Install**” button.
- Follow the on-screen instructions to complete the installation process.
- Restart the IDE when prompted to apply the changes.

#### Manual

- Obtain the TaRDIS plug-in file in a suitable format, such as a JAR file.
- Navigate to the Eclipse IDE installation directory.
- Create a folder named “**dropins**” in the root of the installation directory if it doesn't already exist.
- Copy the TaRDIS plug-in file (JAR) into the “**dropins**” folder.
- Restart the Eclipse IDE to allow the plug-in to be recognized and loaded.

### 3.2 EXTENDING THE TARDIS PLUG-IN

The TaRDIS plug-in for Eclipse IDE is an open-source project, enabling users to customize and enhance its functionality.

#### Customize

- Clone the Repository<sup>1</sup>
- Open Eclipse IDE
- Import the Cloned Project
- Modify the Code
- Build and Test
- Document

---

<sup>1</sup> <https://zenodo.org/doi/10.5281/zenodo.10871115>



## Contribute

- Fork the Repository<sup>2</sup>
- Clone Your Fork
- Create a Branch
- Make Changes
- Test
- Document
- Commit Your Changes
- Create a Pull Request
- Stay engaged

## 3.3 HOW TO USE THE TARDIS PLUG-IN

Once the plug-in is successfully installed, you can immediately leverage its diverse range of features to enhance your development experience.

### Features

Explore and take advantage of the available functionalities to optimize your workflow and make the most out of the plug-in's capabilities.

### Create a TaRDIS Project

To create a TaRDIS project, you can either use the shortcut "Create a TaRDIS Project" located on the left side of your IDE or navigate to the "File" menu and select "New" (Figure 27).

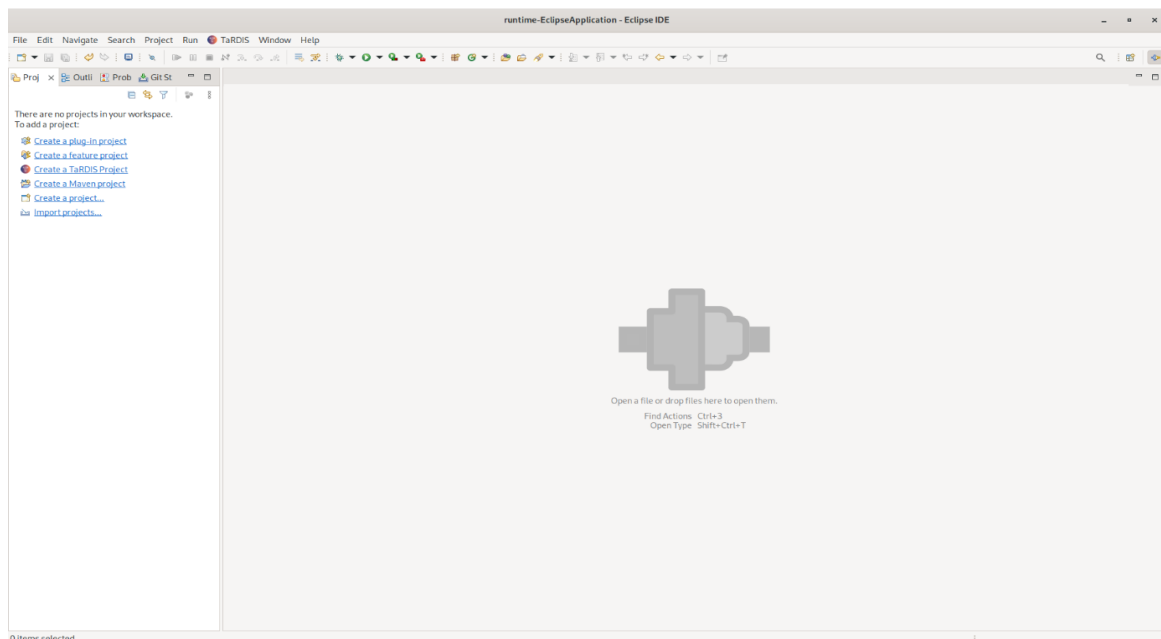
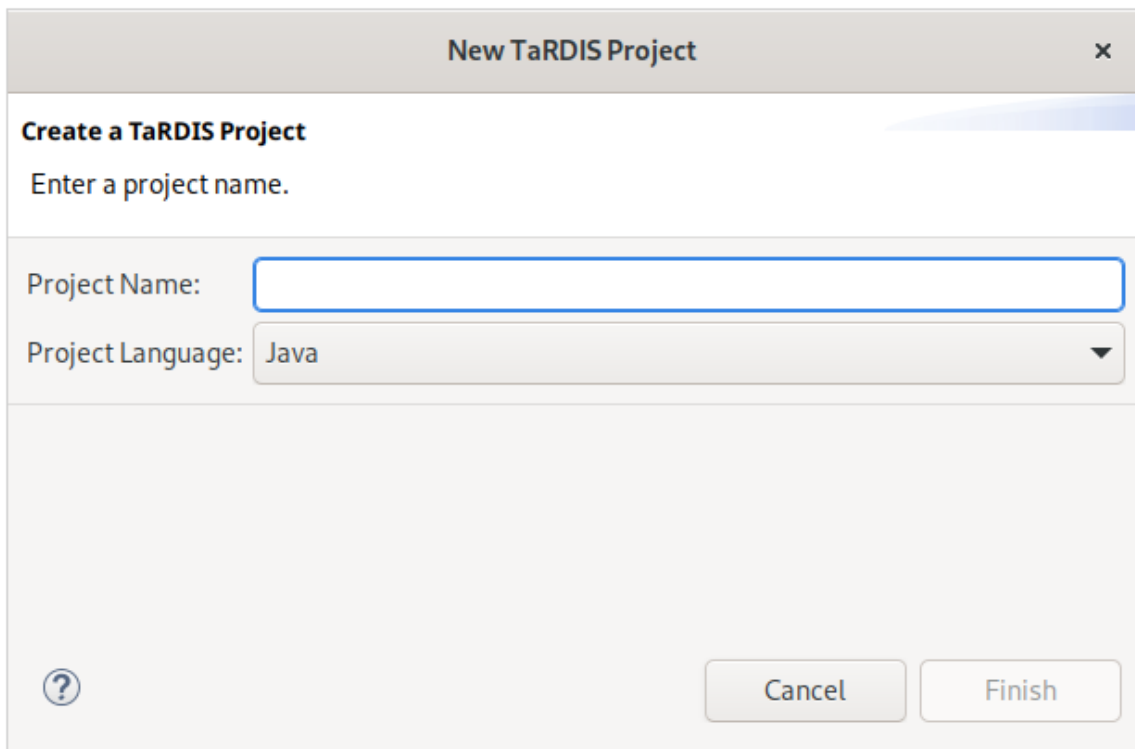


Figure 27: Clean installation of Eclipse IDE with the TaRDIS plug-in

<sup>2</sup> <https://zenodo.org/doi/10.5281/zenodo.10871115>

A wizard will appear (see Figure 28), allowing you to choose between creating a Java project or a TypeScript project.

To create a Java project, you will need to have Java version 17 and Maven installed. This project includes a library called Babel [24], which is used for implementing communication between nodes.

A screenshot of a dialog box titled "New TaRDIS Project". The dialog has a close button (X) in the top right corner. Below the title bar, it says "Create a TaRDIS Project" and "Enter a project name.". There is a text input field for "Project Name:" and a dropdown menu for "Project Language:" with "Java" selected. At the bottom left, there is a help icon (question mark in a circle). At the bottom right, there are two buttons: "Cancel" and "Finish".

**New TaRDIS Project** [X]

**Create a TaRDIS Project**

Enter a project name.

Project Name:

Project Language: Java ▼

?

Cancel Finish

*Figure 28: Wizard for creating a TaRDIS project*

Once the project is created, you can effortlessly run, debug, build, and use the IDE in its usual manner.

The code for this example is open source and freely available on the link:

<https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/eclipse/tardis-eclipse-plugin>

## 3.4 EXAMPLES

Two examples have been developed in Eclipse, using the TaRDIS plug-in:

- a Java implementation<sup>3</sup> featuring a communication system based on HyParView [24].
- and a TypeScript counterpart<sup>4</sup> that focuses solely on simulating calls to the TaRDIS API.

### Eclipse Java communication system

In the Java example, the communication system is intricately built upon the HyParView protocol [24]. Delving into the code reveals details about how nodes interact, information is disseminated, and the system effectively handles failures and maintains resilience. It is crucial to explore the specifics of the HyParView implementation, including node discovery mechanisms, gossip protocols, and the handling of membership changes within the distributed system. Additionally, understanding any unique features or optimizations achieved through the collaboration of the TaRDIS plug-in and HyParView can provide valuable insights into building robust and scalable distributed systems.

The code for this example is open source and freely available on the link:

<https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/eclipse/examples/java-babel-hyparview>

### Babel Classes

The integration of Babel in the Java project has added additional functionality, allowing the generation of specific Babel classes. These classes can be easily accessed through the TaRDIS menu (see Figure 29).

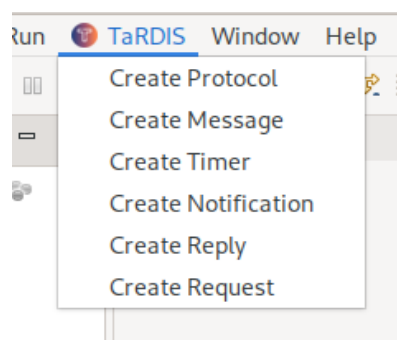


Figure 29: Menu for creating specific Babel Java classes

When selecting a class, a wizard interface will appear, allowing users to customize the class name. Subsequently, the generated code will create a class that either implements or extends the Babel base classes. It's important to note that all the essential logic must be manually implemented within this class. This ensures that the generated code serves as a foundational

<sup>3</sup> <https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/eclipse/examples/java-babel-hyparview>

<sup>4</sup> <https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/eclipse/tardis-eclipse-plugin/-/tree/main/assets/ts>

structure, with users responsible for defining and refining the specific behaviours and functionality of their customized class.

## Protocols

Protocols encode all the behaviour of the distributed system being designed. Each protocol is modelled as a state machine whose state evolved by the reception and processing of (external) events [24].

## Messages

Messages are the data exchanged in the communication channels [24].

## Timers

Timers are essential to capture common behaviours of distributed protocols. They allow the execution of periodic actions (e.g., periodically exchange information with a peer), or to conduct some action a single time in the future (e.g., define a timeout) [24].

## Notifications, Replies and Requests

These enable inter-protocol communication by dispatching notifications to subscribed protocols, accommodating protocol-specific replies and requests [24].

## Eclipse Typescript calls to the TaRDIS API

The TypeScript example concentrates on simulating calls to the TaRDIS API. The objective of this simulation should be thoroughly examined to understand how it contributes to the overall functionality of the system. Analysing the TypeScript code that interacts with the TaRDIS API reveals patterns or abstractions used in the simulation. This may involve mock data, simulated responses, or controlled behaviour designed for testing purposes.

A wizard will appear (see Figure 28), allowing you to choose between creating a Java project or a TypeScript project.

In order to set up a TypeScript project, you will first need to install Node. This project is designed to simulate calls to the TaRDIS API.

This plug-in is built with the Wild Web Developer plug-in to provide support for TypeScript development.

The code for this example is open source and freely available on the link:

<https://codelab.fct.unl.pt/di/research/tardis/toolkit/ide/eclipse/tardis-eclipse-plugin/-/tree/main/assets/ts>

## 4 TARDIS CANDIDATE APPLICATIONS

The TaRDIS project encompasses the development of different applications. Here is a first list of the tools that are being proposed to be part of the TaRDIS toolbox, namely:

- WorkflowEditor
- Ant
- Data preparation for Flower-based Federated Learning (FL) model training
- Flower-based FL model inference and evaluation
- Flower-based FL model training
- Early-Exit (Lightweight Functionality)
- Federated AI Network Orchestrator
- IFChannel
- Java Typestate Checker (JaTyC)
- Knowledge Distillation (Lightweight Functionality)
- P4R-Type
- Pruning (Lightweight Functionality)
- PSPSP
- PTB-FLA-based FL model training
- ReGraDaIFC

### 4.1 WORKFLOWEDITOR (WP4)

#### Purpose

The premise of the workflow-based communication model offered by TaRDIS is that the shape of the interaction between swarm participants is designed, evolved, and maintained by domain experts who are typically not peer-to-peer network programmers. It is therefore necessary and beneficial to offer a graphical representation of workflows which facilitates the collaboration of programmers and domain experts. The WorkflowEditor tool is integrated with the TaRDIS development environment for this purpose, diagrammatically visualizing the workflow as implemented in the program code and allowing modifications to that code by interacting with the diagram.

#### Inputs & Outputs

Since the editor is a bidirectional tool, it transports information and edits both ways:

- The program code is shown in graphical form, also reflecting updates to the code by updating the diagram.
- User interaction that changes the diagram is propagated to the underlying program code, changing it accordingly.

The second function of the editor is to apply behavioural analysis (as per WP4) to the workflow and highlighting any problems found. This should be done both in the program code—as is customary for e.g., type checking errors—and in the diagram. While the former can build on well-established user interface features (like coloured underlines), the latter needs original UX innovation.

### Dependencies on other tools

This integration relies on identifying workflow definitions in source code and reliably extracting the shape of the workflow as well as the source code locations where changes originating in the editor need to be applied. In practice, this feature might only be available if the application-under-development is an internal TaRDIS application (see Deliverable D3.1, section 2.1.1). Most practical would be using the IDE's state of the art plugin for the language in use and leveraging type inference information to find this needle in the haystack.

The editor will need to use the analysis tools from WP4 to check the workflow for well-formedness and find possible errors and their locations so that they can be highlighted.

### Flow Functionality

Initially, the definition of a workflow is started by writing down a definition without states and transitions—this should be aided by an IDE function that instantiates an empty workflow template in the source code editor. Then, all edits of the definition should occur via the graphical editor that can be opened (e.g., from a context menu) when the cursor is positioned within the definition in the source code.

It is important to note that the source code representation of a workflow is equally important for the development process even though it is rarely edited directly, because reading code on GitLab is an important part of the development lifecycle—and GitLab won't display the workflow as a diagram.

### HMI

While the workflow definition language has not yet been defined for any of the targeted host languages (like Typescript and Kotlin), the overall visual appearance might be something like the UML editor used in place of the more capable WorkflowEditor so far (see Figure 30):

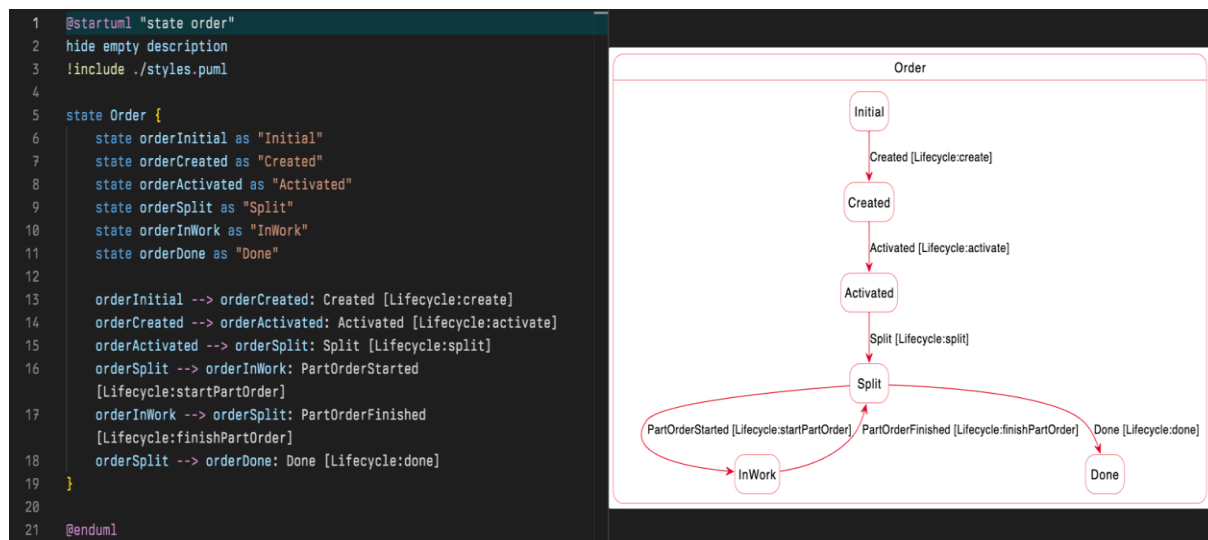


Figure 30: TaRDIS WorkFlowEditor

The code on the left will use a syntax tailored to swarm protocols and be embedded inside the host language instead of using separate files. This allows static type information to be reused between both parts—e.g., ensuring that the structure of a transmitted event as declared in the workflow matches the runtime data structure declared in the programming language.

The diagram on the right will no longer be passive, it will be editable to allow non-programmers to work with the workflow, to design, evolve, and maintain it.

## 4.2 ANT (WP4)

Ant is a language-based call anticipation static analysis tool.

### Purpose

Rather than providing a generic semantic model, agnostic to the language in which the code is written (and sometimes even abstract on the way the model is built), this tool defines its analysis directly on the code, with a language-based algorithm relying instead on formal definitions of program semantics. This analysis is able to produce a conflict table that, for each pair of method calls, defines a system of inequalities that must be satisfied at runtime in order to anticipate a call. A full description of the tool can be found in [25].

### Inputs

The inputs of this tool are the code to be analysed. This can be source code or even, in the example presented in [25], Java bytecode.

### Outputs

The outputs of this tool are the static analysis of the code, stating e.g., non-Locally Permissible methods, non-commutative methods, and anticipatable methods.

## Dependencies on other tools

None

## Flow Functionality

Static analysis over executables or source code. For more information, please refer to [25].

## HMI

Command-line.

## 4.3 DATA PREPARATION FOR FLOWER-BASED FL MODEL TRAINING (WP5)

### Purpose

Enables data preprocessing and preparation for ML training. The process of preparing and preprocessing a data set for applying an ML algorithm represents an inevitable practice, as a raw data set may contain a variety of irregularities, e.g., duplicates, outliers, missing values, etc. Additionally, the need for additional features, such as pseudo-labelling may also emerge. The goal of this tool is to bridge the gap between raw data and FL ML applications, by transforming the data into a format that is suitable for FL ML algorithms.

### Inputs

Input parameters for preparation (as command-line arguments) including the chosen preprocessing approach/es. The user may choose the data set that needs to be pre-processed and prepared, as well as the desired preprocessing task (e.g., profiling, cleansing, transformation...) with possible adjustment of the performed steps (e.g., remove the duplicates and the outliers).

### Outputs

The process results in a transformed data set that is stored in a predefined location, while the output is the status of the performed activities (e.g., whether the process was completed successfully, including additional information where possible such as number of removed duplicates, etc.).

## Dependencies on other tools

This tool requires (and depends on) some well-known Python libraries: Flower, PyTorch (torch and torchvision), Tensorflow, Numpy, Scipy, Scikit-learn, and Pandas.

## Flow Functionality

The process could be described by the following steps:

- Step 1: Initiate the process by specifying the input parameters by the user. Here, the user selects the data set, the task to be performed and possibly adjusts the steps of the task
- Step 2: Perform the preprocessing. The user initiates this after specifying the inputs



- Step 3: Return the status of the performed preprocessing. When the process is finished, a transformed data set is saved and the status of the preprocessing and preparing is available to the user

## HMI

Command-line by default, but with possible GUI support as well.

The command-line approach could have the following form:

```
Please choose a dataset for preprocessing:
1-MNIST
2-CIFAR
....
*enter the corresponding number

$1

You chose the MNIST data set.
Please select a preprocessing task:
1-Cleansing
2-profiling
3-transformation
4-pseudo labeling
...
*enter the corresponding number

$1

You chose cleansing.
The following problems will be detected and the mentioned actions will be performed:
duplicates: delete
outliers: delete
missing values: delete
Do you want to change any of these? (Y/N)

$N

Using default settings.
Setup ready. Do you want to start the preprocessing? (Y/N)

$Y

Preprocessing starting...
Preprocessing completed.
Status: success
Number of duplicates: 12
Number of outliers: 11
Number of missing value occurrences: 9
```

*Figure 31: Data Preparation for Flower-Based FL Model Training*

The possible GUI support could have a form similar to the following mock-up (Figure 32):

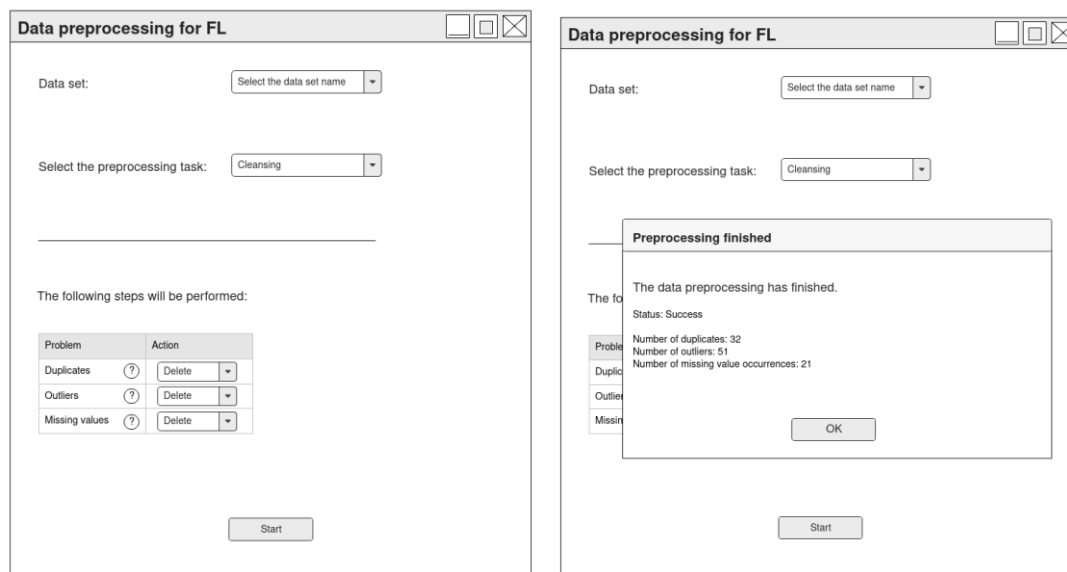


Figure 32: Data Preparation for Flower-Based FL Model Training Mock-up

## 4.4 FLOWER-BASED FL MODEL TRAINING (WP5)

### Purpose

Provides FL solutions and enables model training. This tool supports a list of FL ML algorithms, implemented in the Flower framework, and enables the process of training of the desired model. The user of the tool does not need to have expertise in the field and can perform the training by simply selecting the task that should be solved.

### Inputs

The most important input is the target FL algorithm: the user specifies the task to be solved, e.g., supervised classification, and the system offers a list of suitable ML models and algorithms from the list of available models and algorithms. Also, the user needs to specify the input parameters for the algorithm (as command-line arguments) including the target data set specification, hyperparameters (optional, with predefined values offered), advanced setup parameters (optional, e.g., number of desired clients).

### Outputs

The produced ML model (saved at a predefined location) and the training status (whether it was successfully finished).

### Dependencies on other tools

This tool requires (and depends on) some well-known Python libraries: Flower, PyTorch (torch and torchvision), Tensorflow, Numpy, Scipy, Scikit-learn, and Pandas.

## Flow Functionality

The process could be described by the following steps:

- Step 1: Initiate the process by specifying the task to be solved, e.g., supervised classification
- Step 2: Pick a model and an algorithm from the offered list (with default values offered)
- Step 3: Specify the input parameters: select the data set, possibly adjust the algorithm parameters (or choose the predefined values), optionally adjust the advanced parameters (as the desired number of clients for instance)
- Step 4: Perform the FL algorithm training: when everything is selected and set up, the user initiates the training process
- Step 5: Once the training is completed, the output is the model, the tool provides a status information to the user

## HMI

Command-line by default, but with possible GUI support as well.

The command-line approach could have the following form (Figure 33):

```

Please select a task for solving:
1-supervised classification
2-anomaly detection
...
*enter the corresponding number

$1

You chose supervised classification.
Please select one of the recommended ML models:
1-Logistic regression
2-K-Nearest Neighbors (KNN)
3-Support Vecot Machine (SVM)
...
*enter the corresponding number

$1

You chose Logistic regression.
Please select one of the recommended ML algorithms:
1-FedAvg
2-pFedMe
3-FedProx
...
*enter the corresponding number

You chose Logistic regression.
The default parameters are:
penalty norm: l2
Tolerance: 1e-4
Maximum iterations: 100
Warm start: true
Do you want to change any of these? (Y/N)

$N

Using default parameters.
The available advanced options are:
Number of clients -default value: 10
Do you want to adjust the advanced options? (Y/N)

$N

Using default advanced options.
Please choose a dataset:
1-MNIST
2-CIFAR
...
*enter the corresponding number

$1

You chose the MNIST data set.
Setup ready. Do you want to start training the model? (Y/N)

$Y

Training starting...
Training completed.
The model is available in C:\User\TrainedModel

```

Figure 33: Flower-Based FL Model Training

The possible GUI support could have a form similar to the following mock-up (Figure 34):

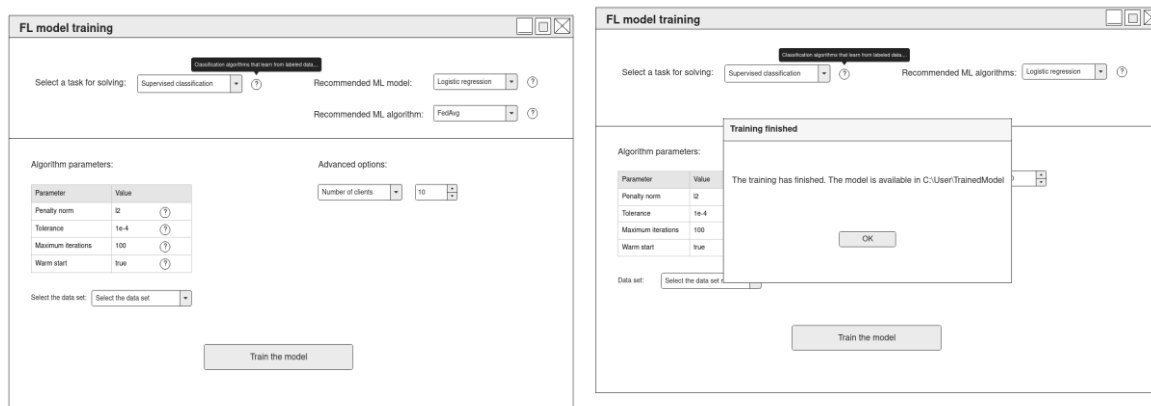


Figure 34: Flower-based FL model training Mock-ups

## 4.5 FLOWER-BASED FL MODEL INFERENCE AND EVALUATION (WP5)

### Purpose

Enables gaining output for the relevant data on a trained model, by means of inference and evaluation. Inference is an important aspect regarding FL. It provides some valuable output for the data set of interest on the trained model. Additionally, this tool also provides evaluations, by offering different metrics for the obtained results where possible.

### Inputs

The inputs to the tool are the model (previously trained), the input parameters (as command-line arguments) including the target data specification, and the type of inference/evaluation to be performed.

### Outputs

The process results in the output, prediction, metric... The form of the output is highly dependent on the chosen inference/evaluation approach. It may be a set of predictions, accuracy values, etc.

### Dependencies on other tools

This tool requires (and depends on) some well-known Python libraries: Flower, PyTorch (torch and torchvision), Tensorflow, Numpy, Scipy, Scikit-learn, Pandas, and Matplotlib.

### Flow Functionality

The process could be described by the following steps:

- Step 1: Initiate the process by providing the model and specifying the input parameters by the user (select the data set and adjust parameters, if applicable).
- Step 2: Perform the inference/evaluation: the user initiates this process after setting up the necessary input.

- Step 3: Return the output to the user in a corresponding form, that depends on the chosen approach.

## HMI

Command-line by default, but with possible GUI support as well.

The command-line approach could have the following form (Figure 35):

```

Please select a model for inference_evaluation
Please provide the file name:

C:\User\Models\MyModel

Model loaded...
Please select the test data set:
1-MNIST
2-CIFAR
...
*enter the corresponding number

$1

You chose the MNIST data set.
Please select the inference/evaluation type:
1-prediction
2-forecasting
3-anomaly detection
...

$1

You chose prediction.
Setup ready. Do you want to start the inference/evaluation? (Y/N)

$Y

Inference/evaluation starting...
Inference/evaluation completed.
The results are available in C:\User\Results.
    
```

Figure 35: Flower-Based FL Model Inference and Evaluation

The possible GUI support could have a form similar to the following mock-up (Figure 36):

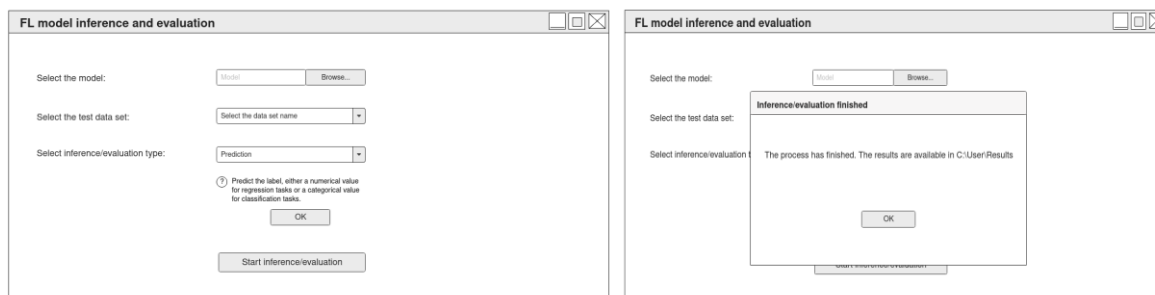


Figure 36: Flower-Based FL Model Inference and Evaluation Mock-ups

## 4.6 EARLY-EXIT (LIGHTWEIGHT FUNCTIONALITY) (WP5)

### Purpose

The purpose of this tool is to take as input a trained Deep Neural Network (DNN) model that consists of several hidden layers and to provide an alternative version of the model that includes multiple exits. Once the new model is trained, its inference latency is reduced compared to the original model; however, the trade-off is the decrease in model accuracy.

## Inputs

The inputs to the early-exit tool are:

- the base ML model (for example .pt) that the developer wants to transform to its early-exit version.
- the type of the ML task that the base original model was trained for (classification or regression).
- the number of desired exits/outputs that shall be included in the new DNN architecture.
- the accuracy threshold that the early-exit version of the DNN model shall achieve in order to be reliable.
- the training dataset that the original model was trained upon is required for training also the early-exit version (labels are required at all the model exits).

## Outputs

The new early-exit version of the DNN, according to the specified input parameters. The output can also include a metadata file with information related to the model accuracy, the training dataset, or the new architecture of the DNN (for instance where in the architecture the model exits are located).

## Dependencies on other tools

Tensorflow, Keras and Pytorch libraries are required for performing the training of the DNN model. No dependencies on other TaRDIS tools are foreseen at the moment.

## Flow Functionality

The flow of the functionality that is included in the early-exit tool can be summarized as follows:

- Step 1: The process is initiated by the developer, by providing a DNN model to the tool, along with the input parameters and the training dataset. The goal is to transform this DNN to an early-exit version of the model.
- Step 2: The training process using the new dimensions (hidden layers and locations of exit in the architecture) of the early-exit DNN is initiated, based on the provided training dataset until the required accuracy threshold is reached.
- Step 3: Once the early-exit version of the DNN is trained, it can be used for fast inference. Therefore, the tool outputs the early-exit version of the ML model while the accuracy level is within limits.

## HMI

Command-line by default. GUI support TBD - will not be of much use for this tool (maybe showing the training process/training loss of the early-exit model).

## 4.7 FEDERATED AI NETWORK ORCHESTRATOR (WP5)

### Purpose

Learn to orchestrate the network in a federated way.

### Inputs

Metrics of the network, or methods to access the metrics, network topology, data load localizations, config file for the AI models, deployment tools to deploy the model to the network nodes.

### Outputs

AI model (set of weights and architecture), training status.

### Dependencies on other tools

pytorch, Tensorflow 2.x, keras, numpy, scipy

### Flow Functionality

- Step 1: The process is initiated by the developer by inputting the computing network topology with the required specifications/dimensions.
- Step 2: The training starts until the required levels of accuracy/speedup are achieved.
- Step 3: The resulting pruned ML model is the output of this tool.

### HMI

Command line.

## 4.8 IFCHANNEL (WP4)

### Purpose

Verify that the use of channels (generating and reacting to events) respects the secure information flow policy. This tool will be backed by a number of theoretical results showing that a successful analysis of the tool implies specific security guarantees.

### Inputs

Currently: simple imperative style programs that communicate via events, where all variables and sending/receiving events are decorated with security labels from a security lattice. This should be extended to include support for more constructs that are commonly used by TaRDIS users. Also, it assumes that the program is using the TaRDIS library where the library functions have similar security labelling, and in particular a fixed set of channels, see PSPSP tool.

## Outputs

Either a security proof certificate, or elements that violate the security policy (which could indicate an attack but might be a false positive).

## Dependencies on other tools

Possibly requires a tool that performs an extraction from real code into a restrictive model.

## Flow Functionality

The potential problems that are found are e.g., typing errors, so the programmer can then try to modify the code and experiment with different designs until the IFChannel tool is happy.

## HMI

Our development would be a command-line tool, but it can be integrated into IDEs like a compiler, highlighting the lines in the code where problems have been found.

## 4.9 JAVA TYPESTATE CHECKER (JATYC) (WP4)

The Java Typestate Checker (JaTyC<sup>5</sup>) is a tool that verifies Java source code with respect to typestates. A typestate is associated with a Java class with 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.

### Purpose

The tool statically verifies that when a Java program runs:

- sequences of method calls obey to object's protocols.
- null-pointer exceptions are not raised.
- subclasses' instances respect the protocol of their superclasses.
- methods are called in the correct order specified by the protocol.
- protocols of objects are completed.
- support for protocols to be associated with classes from the standard Java library or from third-party libraries.
- support for "droppable" states, which allow one to specify states in which an object may be "dropped" (i.e., stop being used) without having to reach the final state.
- support for transitions of state to depend on boolean values or enumeration values returned by methods.
- invalid sequences of method calls are ignored when analysing the use of objects stored inside other objects by taking into account that the methods of the outer object will only be called in the order specified by the corresponding protocol, thus avoiding false positives.

### Inputs

The tool has as input Java source code.

---

<sup>5</sup> <https://github.com/jdmota/java-typestate-checker>



## Outputs

The outputs of this tool are the analysis reports on the analysed code.

## Dependencies on other tools

Dependent on JDK 11.

## Flow Functionality

Static analysis tool over Java source code files.

## HMI

Command-line.

## 4.10 KNOWLEDGE DISTILLATION (LIGHTWEIGHT FUNCTIONALITY) (WP5)

### Purpose

The purpose of the tool is to provide a more lightweight version of a DNN in terms of NN complexity, hidden layers and number of neurons. To this end, this tool trains a “student” version of a “teacher” DNN model. The goal of the student DNN is to achieve comparable accuracy to the teacher mode, while being more lightweight (less hidden layers in the network architecture). In this way, computational resources at the edge nodes required for inference purposes are reduced, without degrading the model performance.

### Inputs

The inputs required for the KD tool are:

- the base teacher ML model (e.g., in a .pt format) that the ML developer wants to transform to more lightweight version.
- the type of the ML task that the model was originally trained for (classification or regression task).
- the ML developer can select a loss function for the training process of the student network (amongst several options).
- the ML developer shall provide as input the number of hidden layers required for the student model, or alternatively the lightweight rate that he wants to achieve (for instance, what percentage of hidden layers must the student model include compared to the teacher model).
- the accuracy threshold is required, since the knowledge gained from the teacher will be distilled in the student network until the accuracy threshold is reached.
- the training dataset that the original teacher model was trained upon is required for similar training of the student network.

## Outputs

The output of the KD tool is the more lightweight version of the DNN model (student model). The output can also include a metadata file with information related to the achieved accuracy, the training dataset, and the lightweight percentage of the new model.

## Dependencies on other tools

Tensorflow, Keras and Pytorch libraries are required for performing the training of the DNN model. No dependencies on other TaRDIS tools are foreseen at the moment.

## Flow Functionality

The flow functionality of the KD tool can be summarized in the following steps:

- Step 1: the KD training process is initiated by the developer/user, by specifying the dimensionality of the student network and providing the required input parameters, including the base teacher model and the training dataset.
- Step 2: the training process is initiated based on the provided training dataset and the knowledge distillation between the teacher and the student model is conducted until the required level of accuracy is reached.
- Step 3: once the training is completed and the required accuracy is achieved, the KD tool outputs the lightweight version of the input model, i.e., the student model.

## HMI

Command-line by default. GUI support TBD - will not be of much use for this tool (maybe showing the training process/training loss of the student model)

## 4.11 P4R-TYPE (WP4)

### Purpose

This tool P4R-Type [26] generates verified APIs (in the Scala 3 programming language) to programmatically control and reconfigure a software-defined network (SDN) that follows the P4<sup>6</sup> and P4Runtime<sup>7</sup> standards. The tool takes as input a specification of the network packet processing tables (in the standard P4Info format) and generates a strongly typed API which ensures that, if a program attempts to reconfigure the network packet processing rules, then such reconfigurations are valid with respect to the network specification. In the TaRDIS project, the P4R-Type tool could be used to support applications that dynamically control and reconfigure of the overlay network of a running swarm (e.g., to regulate swarm membership); this idea has been discussed between the project partners, but the precise use cases have not been finalised yet. The TaRDIS IDE could support developers by providing shortcuts that invoke the P4R-Type tool and automatically (re)generate the required SDN APIs.

---

<sup>6</sup> <https://p4.org/>

<sup>7</sup> <https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.html>

## Inputs

A P4Info metadata file (generated when deploying P4 programs, as part of the P4 standard).

## Outputs

One or more autogenerated Scala files containing the SDN control API, as a series of type definitions.

## Dependencies on other tools

Scala 3 and SBT build system. The tool has further dependencies that are automatically downloaded by its build scripts.

## Flow Functionality

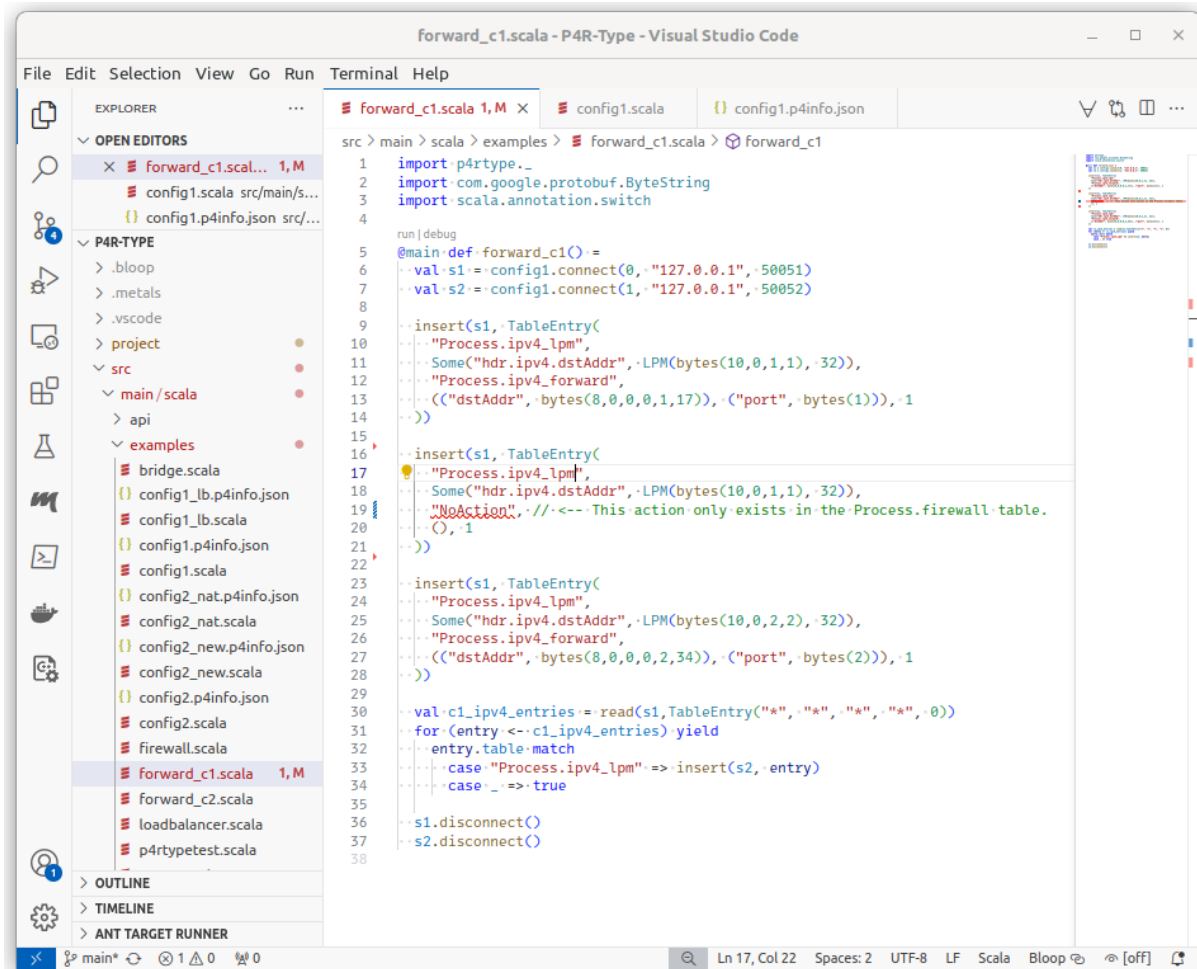
The developer obtains one or more P4Info files corresponding to the deployed P4 configuration in a network. Then, the developer autogenerates a corresponding verified APIs by invoking the P4R-Type tool from the command line. Under an IDE, the tool invocation could be performed e.g., by pressing a button or menu item.

## HMI

Command-line interface + IDE shortcut via e.g., a button or menu item. When the generated API is in use, incorrect network configuration updates would be directly reported by the IDE in the code-under-development, as type errors, using a pre-existing IDE plugin for the Scala 3 programming language (e.g., Metals<sup>8</sup>). For instance, in Figure 37, a program is using an API generated by P4R-Type, and the type error reported on line 19 signals that the packet processing action `NoAction` cannot be used in the packet processing table called `Process.ipv4_lpm` (although that action exists and can be used in other tables in the same network configuration).

---

<sup>8</sup> <https://scalameta.org/metals/>



```

forward_c1.scala - P4R-Type - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
forward_c1.scala 1,M
config1.scala src/main/s...
config1.p4info.json src/...
P4R-TYPE
.bloop
.metals
.vscode
project
src
main/scala
api
examples
bridge.scala
config1_lb.p4info.json
config1_lb.scala
config1.p4info.json
config1.scala
config2_nat.p4info.json
config2_nat.scala
config2_new.p4info.json
config2_new.scala
config2.p4info.json
config2.scala
firewall.scala
forward_c1.scala 1,M
forward_c2.scala
loadbalancer.scala
p4rtypetest.scala
OUTLINE
TIMELINE
ANT TARGET RUNNER
src > main > scala > examples > forward_c1.scala > forward_c1
1 import p4rtype._
2 import com.google.protobuf.ByteString
3 import scala.annotation.switch
4
run | debug
5 @main def forward_c1() =
6   val s1 = config1.connect(0, "127.0.0.1", 50051)
7   val s2 = config1.connect(1, "127.0.0.1", 50052)
8
9   insert(s1, TableEntry(
10     "Process.ipv4_lpm",
11     Some("hdr.ipv4.dstAddr", LPM(bytes(10,0,1,1), 32)),
12     "Process.ipv4_forward",
13     ((dstAddr, bytes(8,0,0,0,1,17)), ("port", bytes(1))), 1
14   ))
15
16   insert(s1, TableEntry(
17     "Process.ipv4_lpm",
18     Some("hdr.ipv4.dstAddr", LPM(bytes(10,0,1,1), 32)),
19     "NoAction", // <--- This action only exists in the Process.firewall.table.
20     (), 1
21   ))
22
23   insert(s1, TableEntry(
24     "Process.ipv4_lpm",
25     Some("hdr.ipv4.dstAddr", LPM(bytes(10,0,2,2), 32)),
26     "Process.ipv4_forward",
27     ((dstAddr, bytes(8,0,0,0,2,34)), ("port", bytes(2))), 1
28   ))
29
30   val c1_ipv4_entries = read(s1, TableEntry("*", "*", "*", "*", 0))
31   for (entry <- c1_ipv4_entries) yield
32     entry.table.match
33     case "Process.ipv4_lpm" => insert(s2, entry)
34     case _ => true
35
36   s1.disconnect()
37   s2.disconnect()
38
Ln 17, Col 22 Spaces: 2 UTF-8 LF Scala Bloop [off]

```

Figure 37: Example of a program using an API generated by P4R-Type

## 4.12 PRUNING (LIGHTWEIGHT FUNCTIONALITY) (WP5)

### Purpose

The purpose of the Pruning tool is to provide a more lightweight version of an input DNN mode by using the pruning method, without significantly degrading its accuracy. To this end, the tool prunes (or eliminates) specific neurons or complete hidden layers of the network architecture that have a negligible impact on the output (and accuracy) of the DNN model. The pruned version is more lightweight, requiring less memory and CPU compared to the original model.

### Inputs

The inputs required by the Pruning tool are the following:

- The base original DNN model (e.g., in a .pt format) that the ML developer wants to transform to more lightweight version by using the pruning technique.
- The compression rate (%) of the new version of the DDN, and the accuracy threshold (%) or optionally the speedup factor (e.g., make the inference 2 times faster) that the pruned version of the DNN model shall achieve.

- The method that will be utilized for pruning, by selecting amongst several available options (for instance, structured pruning may remove specific neurons or complete hidden layers, while unstructured pruning may remove insignificant neurons connections).
- The training dataset, since the updated version of the DNN model needs to be trained against the same dataset as the original model.

## Outputs

The output of the pruning tool is the newer, more lightweight, pruned version of the original DNN model. The output can also include a metadata file with information related to the compression rate, the speedup factor, and the lightweight percentage of the new model.

## Dependencies on other tools

Tensorflow, Keras and Pytorch libraries are required for performing the training of the DNN model. No dependencies on other TaRDIS tools are foreseen at the moment.

## Flow Functionality

The flow functionality of the pruning tool can be summarized in the following steps:

- Step 1: The process is initiated by the developer by providing the original DNN model (to be pruned) to the tool, along with the required specifications (e.g., compression rate, speedup factor, pruning method) and the training dataset that the pruned model will be trained upon.
- Step 2: The training of the pruned model (compression rate of the model is specified by the user) is initiated until the required levels of either accuracy/speedup factor are achieved.
- Step 3: Once the pruned version of the original model has been successfully trained, it is provided back to the user as output of the tool.

## HMI

Command-line by default. GUI support TBD - will not be of much use for this tool (maybe showing the training process/training loss of the pruned model)

## 4.13 PSPSP (WP4)

This is partially an interactive game, one could say: the user trying to convince the tool that everything is fine. This might need specialist work by DTU for the channels we want to offer in TaRDIS, but the general view should be that this becomes more widely applicable and anybody who implements a new kind of channel can use it to verify that channel.

## Purpose

Verify channels that the TaRDIS library provides and check that they are sufficiently strong to transport the information they promise to securely transport. We envision this for the project members to evaluate the channels implemented for the TaRDIS project in the library, not by

the end users of TaRDIS, assuming we have a fixed number of "TaRDIS channel types". However, in general there is no reason why the end users of TaRDIS should not be able to develop new channels protocols and verify them with PSPSP. Note that PSPSP is an existing tool that should be expanded and adapted to TaRDIS.

## Inputs

Relatively abstract protocol description (not directly the implementation) - could be as a choreography (Alice and Bob notation), currently supported is the PSPSP trac transaction format. Moreover, a specification of the channel they are supposed to implement (as an abstract protocol).

## Outputs

Either a security proof certificate or failure to prove the security/verification of the channel may indicate how the implementation is vulnerable to an attack, but since abstract interpretation is involved, it may be a false positive.

## Dependencies on other tools

Relies on the Isabelle proof assistant, possibly a translator from choreography notation to trac.

## Flow Functionality

One has to first design an abstract model of the protocol that the channel implements as well as the security goals for it (this is determined by what security guarantees the channel should provide to its users). Then we have automated methods for checking this model that may just fail to prove it, and that gives a hint either towards an actual security problem of the channel protocol, or that the model/abstraction is too coarse or otherwise inappropriate so that it leads to false attacks. Either way, one may either change the protocol or the model of it and try again.

## HMI

Currently we have it as part of the Isabelle proof assistant and this is what makes it most powerful, as one can basically employ any mathematical reasoning to obtain a security proof. However, we may for users try to offer an interface that helps them both formalize the protocol and goals, as well as understand the problem that prevents the tool from accepting.

## 4.14 PTB-FLA-BASED FL OR ODTs MODEL TRAINING (WP5)

### Purpose

Provides the framework (i.e., execution environment) for FL and ODTs (orbit determination and time synchronization) algorithms by providing 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, as well as the function get1Meas for the TDM (time division multiplexing) communication like the communication between pairs of satellites in the ODTs algorithms. Algorithm developers are advised to use the PTB-FLA development paradigms

(the four-phases or the two-phases) to transform their referent sequential algorithm (manually or by the help of a generative AI tools) into the target PTB-FLA code (the main function plus the client and server callback functions). Developed PTB-FLA based applications can then be executed on the PTB-FLA for the purpose of AI & ML or ODTS model training.

## Inputs

The target FL or ODTS algorithm, input parameters for the algorithm (as command-line arguments) including the target data specification.

## Outputs

ML or ODTS model, training status, etc.

## Dependencies on other tools

None.

## Flow Functionality

- Step 1: Launch the PTB-FLA application by specifying its input parameters (by the user).
- Step 2: Perform the FL or ODTS algorithm training.
- Step 3: Once the training is completed, the output is the model.

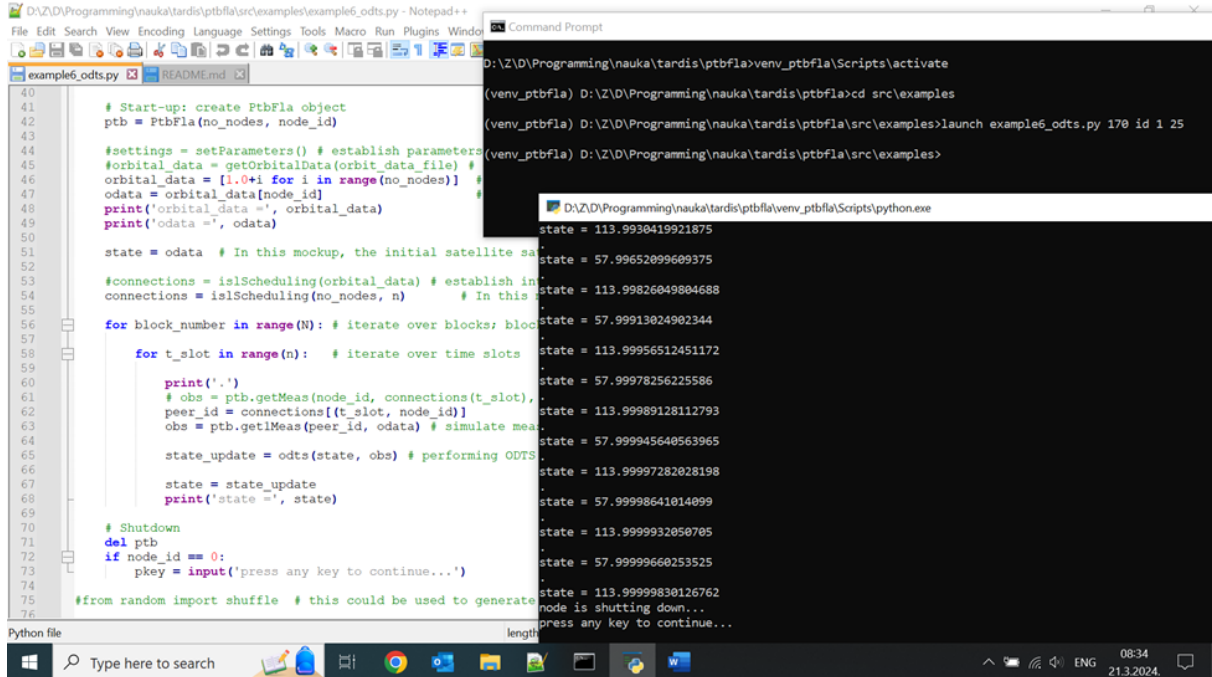
## HMI

Command-line.

## Example

Figure 38 shows the elemental ODTS algorithm execution that comprises three windows. The window in the background shows the Python code. The top-right window shows commands to start the algorithm i.e., the application (in this case, instantiated into 170 instances/processes). The bottom-right window shows the instance with the node identification 0.





```

D:\Z:\D\Programming\nauka\tardis\ptbfla\src\examples\example6_odts.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Windows
example6_odts.py README.md
40
41 # Start-up: create PtbFla object
42 ptb = PtbFla(no_nodes, node_id)
43
44 #settings = setParameters() # establish parameters
45 #orbital_data = getOrbitalData(orbit_data_file) #
46 orbital_data = [1.0+i for i in range(no_nodes)] #
47 odata = orbital_data[node_id]
48 print("orbital_data =", orbital_data)
49 print("odata =", odata)
50
51 state = odata # In this mockup, the initial satellite state
52
53 #connections = is1Scheduling(orbital_data) # establish in
54 connections = is1Scheduling(no_nodes, n) # In this
55
56 for block_number in range(N): # iterate over blocks; block
57
58     for t_slot in range(n): # iterate over time slots
59
60         print('.')
61         # obs = ptb.getMeas(node_id, connections(t_slot),
62         peer_id = connections[(t_slot, node_id)]
63         obs = ptb.getMeas(peer_id, odata) # simulate mea
64
65         state_update = odts(state, obs) # performing ODTs
66
67         state = state_update
68         print("state =", state)
69
70 # Shutdown
71 del ptb
72 if node_id == 0:
73     pkey = input('press any key to continue...')
74
75 #from random import shuffle # this could be used to generate
76
Python file length
D:\Z:\D\Programming\nauka\tardis\ptbfla>venv_ptbfla\Scripts\activate
(venv_ptbfla) D:\Z:\D\Programming\nauka\tardis\ptbfla>cd src\examples
(venv_ptbfla) D:\Z:\D\Programming\nauka\tardis\ptbfla\src\examples>launch example6_odts.py 170 id 1 25
(venv_ptbfla) D:\Z:\D\Programming\nauka\tardis\ptbfla\src\examples>
D:\Z:\D\Programming\nauka\tardis\ptbfla\venv_ptbfla\Scripts\python.exe
state = 113.9930419921875
state = 57.99652099609375
state = 113.99826049804688
state = 57.99913024902344
state = 113.99956512451172
state = 57.99978256225586
state = 113.99989128112793
state = 57.999945640563965
state = 113.99997282028198
state = 57.99998641014099
state = 113.9999932050705
state = 57.99999660253525
state = 113.99999830126762
node is shutting down...
press any key to continue...
  
```

Figure 38: ODTs algorithm execution

## 4.15 REGRADAIFC (WP4)

### Purpose

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.

### Inputs

Editor with syntax highlighting, possible HTML view and editor, and command line tool.

### Outputs

Currently, connects directly to Neo4J and manages the database, provides a web view for execution and inspection. In the future code for the Actyx platform (not sure yet how to manage deployment)

### Dependencies on other tools

NodeJS and OCaml compiler + Actyx framework

### Flow Functionality

Edit processes/graphs in the editor, and use web view to execute, deploy the distributed process as any Actyx application.

### HMI

Command line and web based.



## 5 CONCLUSIONS

This document presents the first approach towards the definition of an Integrated Development Environment tool suited for the TaRDIS toolbox integration efforts. It performed an alternatives analysis on the best-of-breed free and open-source tools in the market that are suitable for satisfying the requirements of the Human-Machine Interfaces (HMI), tool dependencies, visualisation and other needs from the TaRDIS toolbox modules.

From that analysis, a short list of tools was selected for advanced prototyping, which already included the customisation for TaRDIS, the definition of technological stacks and other features related with the project. After a thorough analysis, the tools that were considered best candidates for proceeding, not only due to the analysis of the prototypes, but also from the experts opinion from the consortium, were the Eclipse IDE, a platform that is a reference for many years and has numerous add-ons and features that make it a very stable baseline, with proven results for many years and the choice of many experienced developers.

The team, for this initial IDE definition, and as the TaRDIS toolbox tools are still in their infancy, was limited to the customisation of the base platform, look & feel, and integration of the communication and API technological stack using Babel. A comprehensive tutorial of the installation of the base platform and subsequent customisation adding plug-ins was also depicted in this document.

As future work for the subsequent deliverables of this work-package, the team will include the prototypes and customisation of another IDE platform, Visual Studio Code, as this IDE has been identified as being the choice of the newer generations of programmers. It is becoming a very popular choice, therefore the team decided to also include a similar customisation and integration set of plugins for this IDE in the future deliverables of this TaRDIS work-package.

While it is still not clear whether the team will be able to maintain the development of both IDEs for the whole project, or if all integrations with the TaRDIS toolbox will be possible to be made with both IDEs, it will be a matter of study and development in the project future for the next months.

## REFERENCES

- [1] Shukla, A. (2024). Cloud-Based Lightweight Modern Integrated Development Environments (IDEs) and their Future. *Journal of Artificial Intelligence & Cloud Computing*. SRC/JAICC-236. DOI: [doi.org/10.47363/JAICC/2024\(3\),218,2-3](https://doi.org/10.47363/JAICC/2024(3),218,2-3).
- [2] Birillo, A., Tigina, M., Kurbatova, Z., Potriasaeva, A., Vlasov, I., Ovchinnikov, V., & Gerasimov, I. (2024). Bridging Education and Development: IDEs as Interactive Learning Platforms. *arXiv preprint arXiv:2401.14284*.
- [3] Lyu, M., Zhang, Y., Liu, S., & Chen, L. (2023). Adapting Jupyter for C++ Programming Education: An Empirical Study on Lab Instruction Strategies and Student Perspectives. *Contemporary Education and Teaching Research*, 4(11), 556–561. <https://doi.org/10.61360/BoniCETR232015151101>
- [4] KeyCDN, “Best IDE Software - A List of the Top 10”, <https://www.keycdn.com/blog/best-ide/>, retrieved January 10, 2024
- [5] <https://www.g2.com/categories/integrated-development-environments-ide>, retrieved on January 14, 2024
- [6] Stackify, “Top IDEs: 51 Powerful Dev Environments for Streamlined Development”, <https://stackify.com/top-integrated-developer-environments-ides/>, retrieved on March 10, 2024
- [7] “20 Best IDE Software [2023 Guide]”, <https://thectoclub.com/tools/best-ide-software/>, retrieved on January 11, 2024
- [8] Medium, “The Top 10 IDEs for Programmers: A Comprehensive Guide to Choosing the Best IDE For Your Needs”, <https://medium.com/codex/the-top-10-ides-for-programmers-a-comprehensive-guide-to-choosing-the-best-ide-for-your-needs-c72e97c34591>, retrieved on January 11, 2024
- [9] GoodFirms, “10 Free and Open Source IDEs”, <https://www.goodfirms.co/integrated-development-environment-software/blog/best-free-and-open-source-integrated-development-environment-software>, retrieved on January 11, 2024
- [10] GeeksforGeeks, “Best IDEs for C++ developers”, <https://www.geeksforgeeks.org/best-ides-for-c-c-plus-plus-developers/>, retrieved on January 11, 2024
- [11] TatvaSoft, “Top Java IDEs for Java development”, <https://www.tatvasoft.com/outsourcing/2024/03/java-ides.html>, retrieved on January 11, 2024
- [12] Syndell, “Software Development Tools”, <https://syndelltech.com/software-development-tools-to-improve-productivity/>, retrieved on January 11, 2024
- [13] StudySmarter, “Integrated Development Environments (IDEs)”, <https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/integrated-development-environments/>, retrieved on January 11, 2024
- [14] TechBeamers, “Best IDE for Java development”, <https://techbeamers.com/best-java-ide-web-development/>, retrieved on January 11, 2024

- [15] HackrIO, “Best Java IDEs”, <https://hackr.io/blog/best-java-ides>, retrieved on January 11, 2024
- [16] Sourceforge, “Best Open Source Integrated Development Environments (IDEs)” <https://sourceforge.net/directory/integrated-development-environments-ide/windows/>, retrieved on January 11, 2024
- [17] Apache, Apache NetBeans, <https://NetBeans.apache.org/>, retrieved on January 17, 2024
- [18] JetBrains, IntelliJ IDEA, <https://www.jetbrains.com/idea>, retrieved on February 5, 2024
- [19] Eclipse Foundation. (n.d.). PDE | The Eclipse Foundation. Eclipse. Retrieved January 18, 2024, from <https://eclipse.dev/pde/>
- [20] Microsoft (n.d.). Code editing. Redefined. Visual Studio Code. Retrieved February 14, 2024, from <https://code.visualstudio.com/>
- [21] Microsoft (2024, January 2). *Extension API*. Visual Studio Code. Retrieved February 21, 2024, from <https://code.visualstudio.com/api>
- [22] AMIQ EDA. (2023, December 21). 3.1 What is a Workspace. DVT Eclipse IDE. Retrieved January 18, 2024, from [https://dvt.eclipse.com/documentation/e/What\\_is\\_a\\_Workspace.html](https://dvt.eclipse.com/documentation/e/What_is_a_Workspace.html)
- [23] Vogel, L. (2024, January 8). *Using the Eclipse IDE for Java programming - Tutorial*. vogella.com. Retrieved January 18, 2024, from <https://www.vogella.com/tutorials/Eclipse/article.html>
- [24] Fouto, P., Costa, Á., Preguiça, N., and Leitão, J. (2022, May 5). Babel: A Framework for Developing Performant and Dependable Distributed Protocols. DOI:10.1109/SRDS55811.2022.00022
- [25] Giunti, M, Paulino, H, and Ravara, A. (2023, June 7). Anticipation of Method Execution in Mixed Consistency Systems, in SAC '23: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, March 2023, Pages 1394–1401, DOI:10.1145/3555776.3577725
- [26] Larsen, J. K., Guanciale, R., Haller, P., and Scalas, A. (2023 October). P4R-Type: A Verified API for P4 Control Plane Programs. Proc. ACM Program. Lang. 7, OOPSLA2, Article 290, 29 pages. DOI:10.1145/3622866