TaRDIS

D3.5: Third Report on Programming Model and APIs

Revision: **1.0**

Work package	WP3
Task	T3.1, T3.2
Due date	31/08/2025
Submission date	19/09/2025
Deliverable lead	Alceste Scalas (DTU)
Version	1.0
Authors	João Costa Seco (NOVA), Cláudia Soares (NOVA), Frederico Metelo (NOVA), Carla Ferreira (NOVA), João Leitão (NOVA), António Ravara (NOVA), Diogo Jesus (NOVA), Nuno Fernandes (NOVA), Nuno Preguiça (NOVA), Diogo Paulico (NOVA) Carlos Reis (CMS), Carlos Coutinho (CMS) Sebastian Alexander Mödersheim (DTU) Ping Hou (OXF) Dušan Jakovetić (UNS), Lidija Fodor (UNS), Miroslav Popovic (UNS), Ivan Prokić (UNS), Simona Prokić (UNS), Miloš Simić (UNS), Tamara Ranković (UNS), Miodrag Djukic (UNS), Pavle Vasiljevic (UNS) Sotirios Spantideas (NKUA) Roland Kuhn (ACT)
Internal Reviewers	Lidija Fodor (UNS), Eleni Leligkoy (UNIWA)
Abstract	This document reports the third revision of the programming model and APIs which will be offered by the TaRDIS toolbox. It focuses on the differences and improvements since their previous iteration (reported in Deliverable D3.3) and documents the status of the toolbox APIs (whose components were selected in Deliverable D7.2, and are being used in the development of the TaRDIS use cases).
Keywords	decentralised programming toolbox, models, APIs

Grant Agreement No.: 101093006
Call: HORIZON-CL4-2022-DATA-01

Topic: HORIZON-CL4-2022-DATA-01-03
Type of action: HORIZON- RIA

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

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 No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

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

DEM: Demonstrator, pilot, prototype, plan designs

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

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

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

EXECUTIVE SUMMARY

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

The main contribution of this deliverable is the third revision of the TaRDIS programming models and TaRDIS toolkit APIs. This deliverable builds upon D3.3 (second version of TaRDIS models and APIs) and documents the developments and improvements since the submission of D3.3 — which are mostly consequences of the lessons-learned with the ongoing work on WP7 (implementation and evaluation of the use cases using the TaRDIS toolbox). This deliverable also introduces the specification of a new form of managed swarm application tailored towards applications based on federated learning (Section 2.4).

At the time of writing this Deliverable, the documentation of the TaRDIS models and APIs is being assembled in a programmer-oriented format under the TaRDIS wiki. Therefore, most sections of this document consist of brief summaries outlining the differences and improvements w.r.t. the contents of D3.3, with references to the relevant parts of the TaRDIS wiki (and other source code repositories, when applicable) for further technical details.

Funded by the European Union

TABLE OF CONTENTS

Funded by the European Union

ABBREVIATIONS

API	Application Programming Interface
AGV	Automated Guided Vehicle
BDS-3	BeiDou 3rd Generation navigation satellite system
CDF	Cumulative Distribution Function
DCR	Dynamic Condition Relation
DER	Distributed Energy Resources
DL	Deep Learning
DNN	Deep Neural Network
DP	Differential Privacy
DRFL	Deep Reinforcement Federated Learning
DSO	Distribution System Operator
ERP	Enterprise Resource Planning
FL	Federated Learning
FLaaS	Federated Learning as a Service
G2G	Galileo 2nd Generation of satellites
HTTP	Hypertext Transfer Protocol
IFC	Information Flow Control
IoT	Internet of Things
ISL	Inter-Satellite-Link
IP	Internet Protocol
IPFS	InterPlanetary File System
JS	JavaScript
LEO	Low Earth Orbit
LSTM	Long Short-Term Memory
MES	Manufacturing Execution System
ML	Machine Learning

Funded by the European Union

MPST	Multiparty Session Types
ODTS	Orbit Determination and Time Synchronization
P2P	Peer-to-Peer
PNT	Position, Navigation and Timing
PTB-FLA	Python/MicroPython Test Beds for Federated Learning Algorithms
SGAM	Smart-Grid Architectural Model
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Funded by the European Union

1 INTRODUCTION

This report documents the ongoing work on the design and development of the TaRDIS programming toolkit — specifically, its programming models and APIs.

The definition and development of the programming model and APIs is a collaborative effort that requires a close collaboration among all project partners. The TaRDIS toolbox is shaped by the experiences collected in WP7 for the implementation and evaluation of the project use cases.

The components of the TaRDIS toolbox were described in Deliverable D7.2 ("Report on preliminary validation of the toolbox"), and their status was previously reported in Deliverables D3.1 and D3.3. This document provides a further update on the status of the toolbox components, focusing on how they offer programming models and APIs to software developers that will use TaRDIS. For ease of reference, this document uses the tool codes that can be found in Table 3 (page 61) of Deliverable D7.2 (e.g., "T-WP3-01").

At the time of writing this Deliverable, the documentation of the TaRDIS models and APIs is being assembled in a programmer-oriented format under the TaRDIS wiki:

> https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/home
>
> ***NOTE:*** *At the time of writing this report, the TaRDIS documentation wiki is under heavy development and not yet publicly available. To obtain access for reviewing purposes, please contact the TaRDIS project coordinator.*

This document has the following structure:

- Section 2 provides an updated outline of the TaRDIS programming model, and the methodology and considerations leading to its design. The main update is Section 2.4, which introduces the specification and tooling of a new form of managed swarm application tailored towards federated learning.
- Section 3 provides an overview of the APIs that will be offered by the TaRDIS toolbox.
- Section 4 discusses the limitations and challenges in the development and organisation of the TaRDIS models and APIs, and planned work to address them during the remaining months of the TaRDIS project.

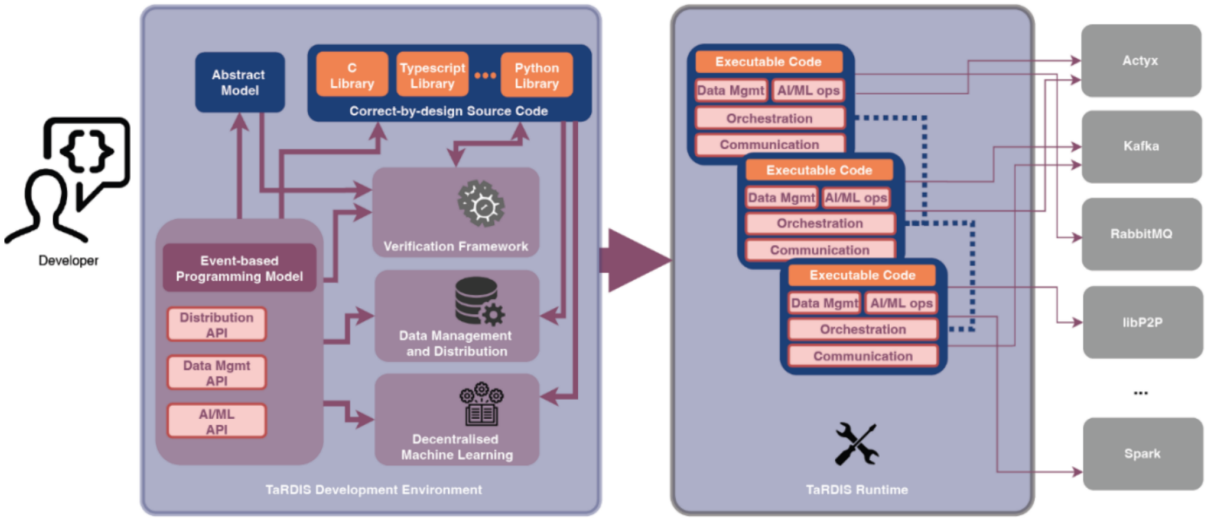
The conclusion (Section 5) summarises the main outcomes and outlines the next steps.

Note: to ease readability, the structure of this document closely follows (where possible) the structure of Deliverable D3.3: i.e., unless otherwise noted, if a section number X.Y appears both here and in D3.3, then section X.Y in this document is intended as a status update over the corresponding section X.Y of D3.3.

Funded by the European Union

2 PROGRAMMING MODEL OVERVIEW AND DESIGN METHODOLOGY

The TaRDIS project proposal envisions an event-driven programming model and toolbox allowing application programmers to take advantage of various facilities (communication, verification, machine learning, monitoring and reconfiguration) helping them develop safe and reliable distributed swarm applications. Such facilities are made available through the "TaRDIS Runtime" — which provides various higher-level APIs and abstractions over lower-level libraries and services; moreover, the proposal envisions dedicated IDE support to simplify the programmers' tasks. This vision is summarised in the figure below (from the TaRDIS project proposal); here the "abstract model" is an abstract representation of a TaRDIS application, which enables the use of the toolbox facilities for software verification and correct-by-construction software development.



This vision has been refined and made more concrete during the first phases of the TaRDIS project. Since Deliverable D3.1, WP3 has collected and organised the feedback of the research work packages (WP4, WP5, WP6) and their application to the requirements and implementation of the project use cases (stemming from WP2 and WP7). The present deliverable is one of the outcomes of this collaborative process.

The rest of this section provides an overview of the TaRDIS approach based on managed and free-form swarm elements (Section 2.1), and then illustrates them in more detail (Sections 2.2, 2.3, 2.4, and 2.5).

2.1. THE TARDIS APPROACH: MANAGED VS. FREE-FORM SWARM ELEMENTS

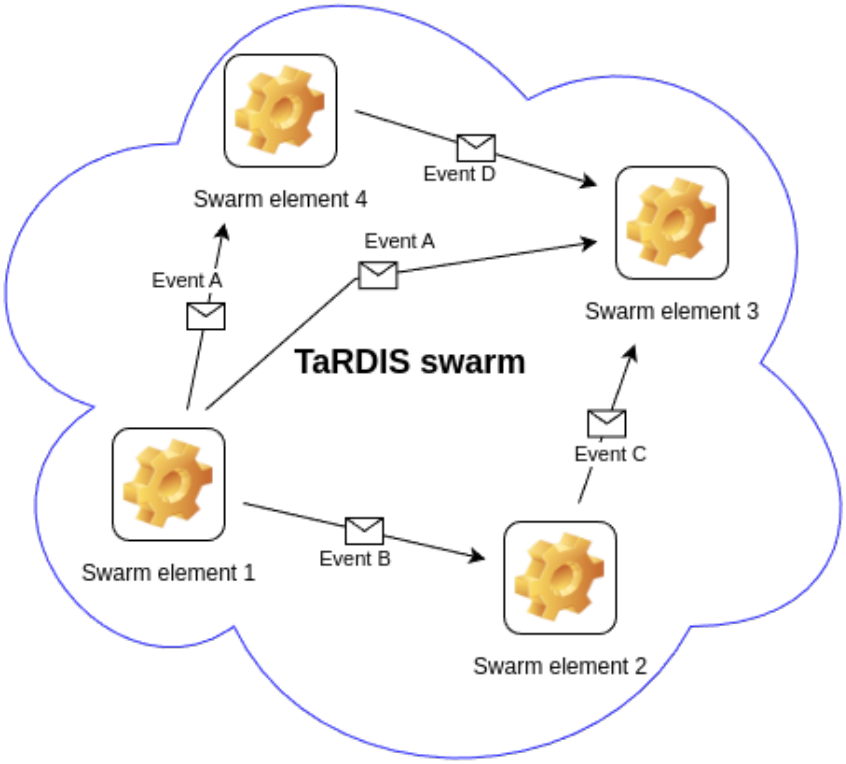
Note: This section summarises the contents of Section 2.1 in Deliverable D3.1 (with some revised terminology). It also appeared in a similar form in D3.3; we repeat the contents here to make this document self-contained. The only update since D3.3 is the mention of the new PTB-FLA approach for managed swarm applications.

Given the variety of the use cases and their requirements, the TaRDIS toolbox is being designed to support the development of **swarm applications** combining two main kinds of

Funded by the European Union

swarm elements: **managed swarm elements** and **free-form swarm elements**.[1] The intuition is the following (and is depicted in the figure below):

- a **TaRDIS swarm application** is an ensemble of concurrent, distributed, and possibly heterogeneous **swarm elements** which interact over a network in an event-driven fashion, using the communication facilities provided by the TaRDIS toolbox;
- a **managed swarm element** delegates its main event loop to the **TaRDIS execution engine**, which invokes relevant parts of the program code in an event-driven fashion. Managed swarm elements are constrained to a specific programming style, but programmers have more complete access to the higher-level APIs and verification capabilities of the TaRDIS toolbox;
- a **free-form swarm element** can communicate with other elements of a TaRDIS swarm application by directly using the TaRDIS APIs, with more control of its main event loop. This gives programmers more freedom in structuring their code, but they may not have complete access to the higher-level APIs and verification capabilities of the TaRDIS toolbox.



Besides the updated terminology, the contents of Sections 2.1.1, 2.1.2, and 2.1.3 of Deliverable D3.1 are still relevant.

Since Deliverables D3.1 and D3.3, the further analysis and initial redevelopment of the TaRDIS use cases has led us towards refining the specific programming models supported by the TaRDIS toolbox. The next sections describe:

[1] In D3.1 they were called “internal services” and “perimeter services,” respectively. Starting with D3.3, we have revised the terminology for clarity.

Funded by the European Union

- Three approaches to the design of managed swarm applications. The approaches offer different takes on the event-driven programming nature of the TaRDIS toolbox, and one may be preferred over the other depending on the nature of the application and the background of the programmers:
    - **state-oriented specifications** based on **swarm protocols** (Section 2.2). This approach is adopted for the development of the ACT use case;
    - **event-oriented specifications** based on DCR graphs (Section 2.3). This approach is adopted for the development of the EDP use case;
    - swarm applications via the **PTB-FLA paradigm** (Python Testbed for Federated Learning Algorithms, Section 2.4). This form of managed swarm application is new to this deliverable and is adopted for the development of the GMV use case.

- An overview of free-form TaRDIS applications based on **Babel** (Section 2.5).

2.2. State-Oriented Managed Swarm Specifications via Swarm Protocols

Note: this section also appeared in a similar form in D3.3; we repeat the contents here to make this document self-contained. This part of the TaRDIS toolbox is quite stable and has had no major design updates since D3.3; the updates were focused on improving the tooling and the library internals.

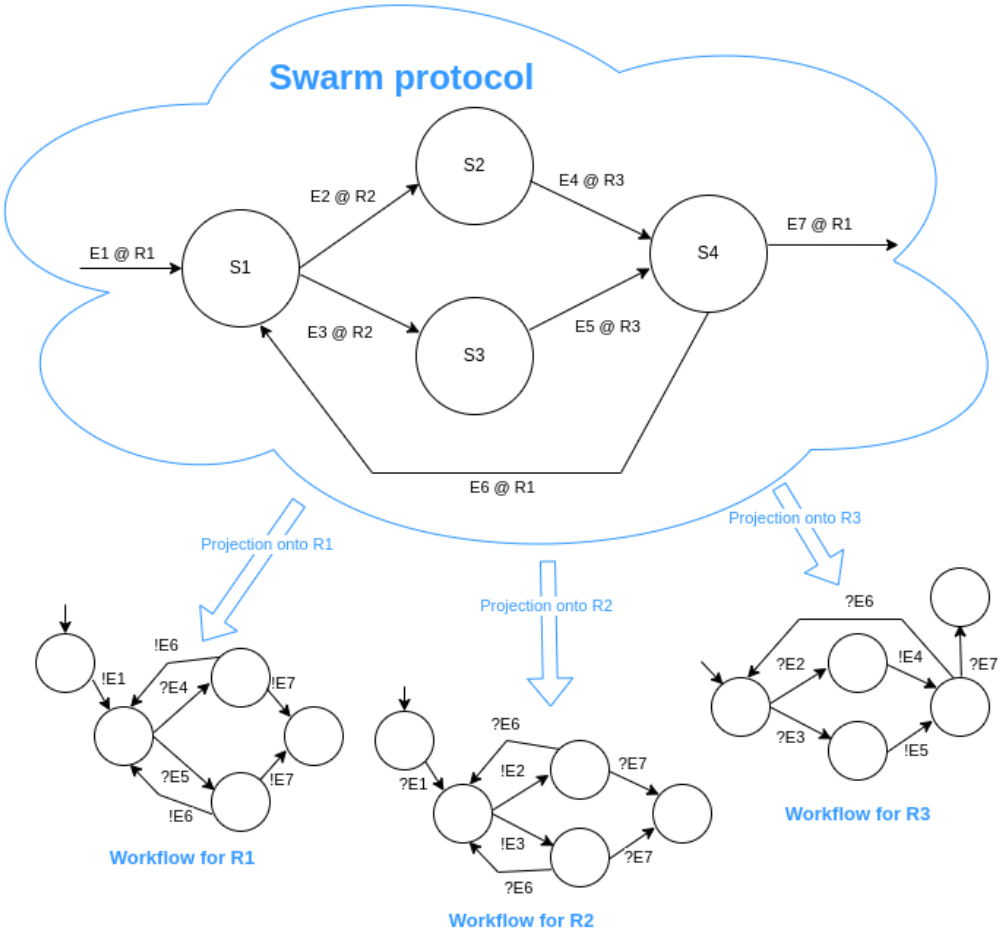
This approach to the specification of managed swarm applications was described in Section 3.1.1 of Deliverable 3.1. What follows is a brief recap (which also appeared in D3.3). The updated documentation is available in the TaRDIS documentation wiki. (For more details, see Section 3.2.1.1.)

The TaRDIS toolkit will provide tools allowing for:

- the specification of **swarm protocols**, offering a global bird's eye view of the intended behaviour of all components that might join a swarm application (each one implementing a specific **role**) to produce and consume **events**;
- the **projection (i.e., synthesis) of local workflows out of a swarm protocol**, ensuring that the local behaviour of a swarm element is compatible with the rest of the swarm.

The idea is illustrated in the figure below:

Funded by the European Union



In the figure above:

- The swarm protocol describes a "global distributed state machines" where the nodes S1,...,S4 represent states, and the edges correspond to the intended interactions between roles R1, R2, and R3; such roles, in turn, may produce and consume events E1...E7 (the notation "E @ R" means that event E is produced by some swarm participant having role R). These events advance the overall state of the protocol.
- The swarm protocol is projected into workflows for the roles R1, R2, and R3: for instance, the workflow for role R2 says that a swarm participant implementing role R2 is expected to await event E1, and then emit one of the events E2 or E3, and then await E6 (looping back to a previous state) or E7 (which terminates the workflow).

The availability of the global swarm protocol specification has two advantages:

1. it provides an intuitive overview of the system behaviour that can be easier to understand by non-experts, and
2. enables better analysis of the system behaviour using the analysis methodologies and tools developed in WP4 (see the TaRDIS Deliverables D4.1 and D4.2).

Concretely, the TaRDIS swarm protocol, workflow model, and execution engine are being designed and developed upon improved versions of the *machine runner*[2] and *machine check*[3] tooling created and released (under Open Source license) by the (former) project partner

[2] https://www.npmjs.com/package/@actyx/machine-runner
[3] https://www.npmjs.com/package/@actyx/machine-check

Actyx: their approach is described in recent papers[4] [5] and is being improved with the ongoing work of WP3, WP4, WP6, and WP7.

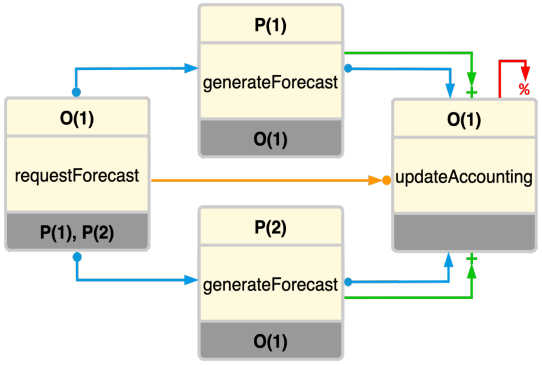
2.3. Event-Oriented Managed Swarm Specifications via DCR Choreographies

Note: this section also appeared in a similar form in D3.3; we repeat the contents here to make this document self-contained. This part of the TaRDIS toolbox is quite stable and all design updates since D3.3 were focused mainly on increasing language expressiveness and including a visual editor for DCR choreographies.

This approach to the specification of managed swarm applications was described in Deliverable 3.1 (Section 2.2.1). This section provides a summary and some revised terminology, reporting on the progress since Deliverable 3.1. Updated documentation, as well as links to additional resources, can be found in the TaRDIS documentation wiki.[6]

ReGraDa / DCR Choreographies (previously, ReGraDa / DCR Graphs) provide a declarative, event-driven, and stateful approach to the specification of workflows within a swarm application. The language supports both graphical and textual representation, the two being interchangeable, providing a high-level abstraction that is both intuitive and human-readable, as well as machine-executable.

The figure below illustrates both representations. The scenario was described in Section 4.2.5.1 of Deliverable 3.1, modelling an energy-generation forecast workflow within an Energy Community. The textual representation was previously detailed throughout Section 4.2.5 of Deliverable 3.1. The graphical notation, in the style of DCR Choreographies, maps the textual representation, abstracting away some details for the sake of presentation.



(r: requestForecast) [?] [O(1) -> P(1), P(2)]
(g1: generateForecast) [? : Integer}] [P(1) -> O(1)]
(g2: generateForecast) [? : Integer[P(1) -> O(1)]
(u: updateAccounting) [g1.value+g2.value] [O(1)]
;
r *→ g1, g2
r →* u
g1 *→ u

In summary, the specification reflects the intended global behaviour of the swarm, defining messages (and data) exchanged between swarm participants, together with control-flow constraints (beyond basic data dependencies) enforcing the system's business logic. Both messages and constraints are associated to (stateful) events. Swarm participants drive the

[4] Roland Kuhn, Hernán C. Melgratti, Emilio Tuosto: Behavioural Types for Local-First Software. ECOOP 2023: 15:1-15:28. https://doi.org/10.4230/LIPIcs.ECOOP.2023.15

[5] Roland Kuhn, Alan Darmasaputra: Behaviorally Typed State Machines in TypeScript for Heterogeneous Swarms. ISSTA 2023: 1475-1478. https://doi.org/10.1145/3597926.3604917 - https://doi.org/10.48550/arXiv.2306.09068

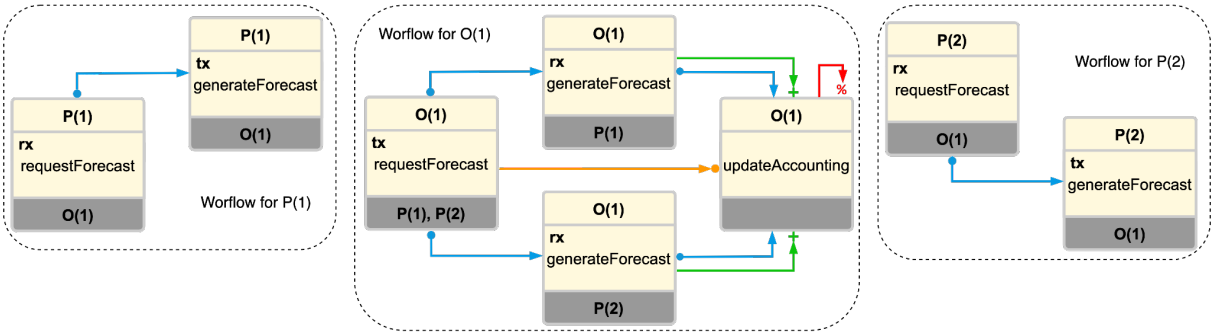
[6] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/The-TaRDIS-programming-model

Funded by the European Union

workflow by executing events, thereby updating their internal state and triggering communication with other participants.

Control-flow relations shape the workflow, by preventing/allowing the execution of specific events, based on the type of constraint and/or the state of events. As detailed in Section 2.2.1 of Deliverable 3.1, control-flow constraints can impose different semantics, according to their type, enabling the specification of complex business processes and workflows, in an intuitive and flexible manner.

By leveraging ReGraDa / DCR Choreographies, the TaRDIS toolkit can support more intricate workflows that go beyond simple state machines. The language enables the projection of local behaviour out of a global specification, reflecting local executable workflows for each participant role, as depicted below:

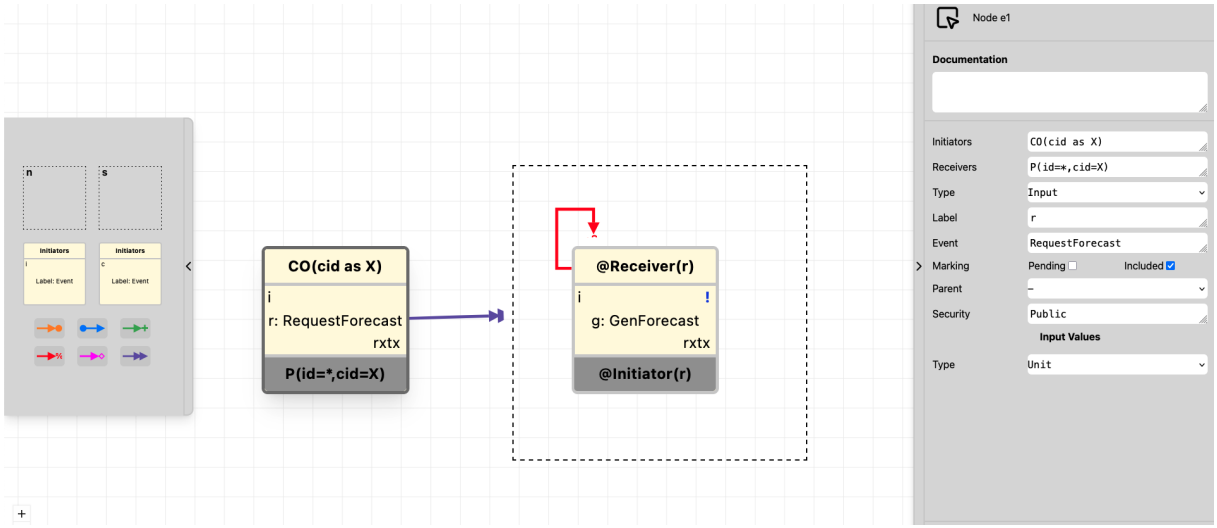


For local specifications, events are additionally annotated with *tx* (respectively, *rx*) to convey, from the local viewpoint, the transmission (respectively, receiving) of a message. The prototype tool currently leverages the Babel framework (Section 2.4 below) to enact each participant's behaviour. Local specifications are translated into Java code running directly on Babel, enabling decentralised execution of different swarm participants as distributed nodes. Both language and prototype tools are under active development and currently being extended to support the dynamic creation of data, behaviour and participants. The inclusion of additional features from DCR Choreographies, such as time constraints, is also being planned.

Since D3.3, **new features were added to the language**, namely, the spawn relation that allows for the dynamic creation of swarms, distinguishing different communities through the message parameters. The demonstration, available in the TaRDIS DCR[7] repository can be executed in a network of deployed babel nodes or locally using containers for each node. Notice in the figure below, depicting the DCR Editor tool, that, when a community orchestrator (**CO(cid=X)**) executes an event "RequestForecast", multiple prosumers are targeted in the message (**P(id=*, cid=X)**) which means that, for each of the prosumers registered in the orchestrator's community, a "GenForecast" event is created at each Receiver, allowing them to respond directly to the community orchestrator (the Initiator). No other prosumers even see the request, or get a new event to respond. This mechanism increases the expressiveness of the language enabling the refinement of participant's groups and dynamic creation of events.

[7] https://codelab.fct.unl.pt/di/research/tardis/wp3/TaRDIS-DCR-Demo

Funded by the European Union



The described example is depicted in the Figure above using the Visual DCR Editor. The compiler associated with the editor then produces code that is run on parametric babel nodes initialized with the role of each participant. The prototype is available on a Codelab repository, along with a detailed presentation of the illustrated scenario, and initial documentation is provided on the TarDIS wiki.[8] [9]

2.4. Correct by Construction Managed Swarm Applications via PTB-FLA

As mentioned above, the initial approach for specifying managed swarm applications was introduced in Deliverable 3.1 (Section 2.2.1) and then extended in D3.3. This section introduces a new model of managed swarm, which complements state-oriented and event-oriented managed swarms (Sections 2.2 and 2.3 above). The goal of this new model is to provide a swarm application development approach that is more tailored for scenarios where federated learning or Time Division Multiplexing (TDM) communication play a central role. Supplementary resources can be found in the TaRDIS documentation wiki.[9]

PTB-FLA (Python Testbed for Federated Learning Algorithms) provides a lightweight and structured approach to design, development and execution of swarm applications in a managed way. It presents a unified framework for specifying and running applications based on both generic Federated Learning Algorithms (FLAs) and generic algorithms for Time Division Multiplexing (TDM) communication, through the use of a restricted programing model based on callback functions and the Single Program Multiple Data (SPMD) Paradigm. The PTB-FLA system architecture consists of the application launcher process, a distributed application $A=\{a_1, a_2,\ldots,a_n\}$ representing a set of application instances, and distributed testbed $T=\{t_1,t_2,\ldots,t_n\}$ representing the corresponding testbed instances, where n is the total number of instances in both A and T. (Fig. 1)

[8] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/The-TaRDIS-programming-model
[9] https://codelab.fct.unl.pt/di/research/tardis/wp3/TaRDIS-DCR-Demo

Funded by the European Union

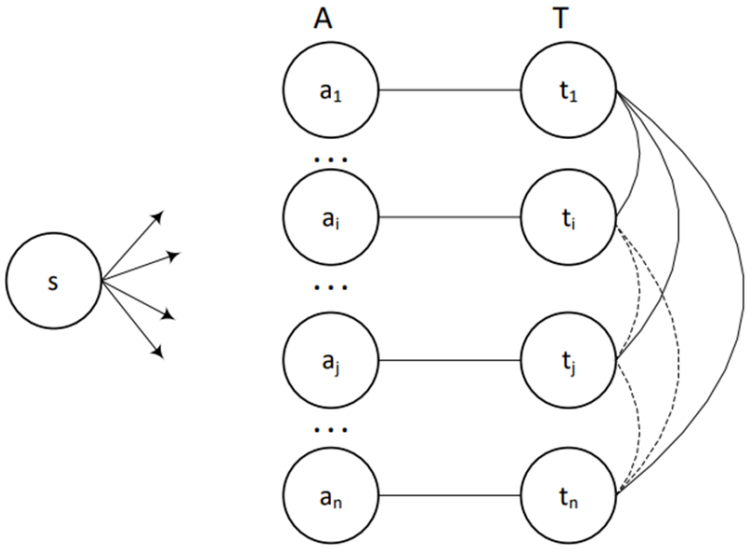


Fig. 1: PTB-FLA system architecture

By abstracting away low-level inter-node communication, synchronization, and system interaction, PTB-FLA aims at providing a guided, trustworthy and reliable approach to the development of swarm applications, that are based on premade well-defined primitives. Moreover, all the abstractions provided by PTB-FLA's high level API have been formally verified using communicating sequential processes (CSP) and the Process Analysis Toolkit (PAT) (further described in Section 3.2.4).

PTB-FLA further builds upon its abstractions by providing a formalised development paradigm for FLAs that guides developers through a structured, incremental process for transforming machine learning applications into fully fledged federated learning swarm applications. The methodology is based on four transformation phases, each producing semantically equivalent code while progressively integrating distributed execution aspects of FL.[10] In addition to this, the PTB-FLA approach offers a guideline for developing swarm applications based on it with the help of LLMs (such guidelines were developed as part of the TaRDIS project).[11]

In the case of the generic TDM algorithms (get1Meas and getMeas), the approach taken by the developers follows the process of converting a sequential program into a distributed one, by using the Single Program Multiple Data (SPMD) pattern. This entails defining a single program that behaves differently based on its identifier and the data available to it. In this approach the synchronization between parts of the program, which uses a shared memory, is replaced by the exchange of data between application instances through message-passing communication. This process can well be demonstrated by the approach taken in the GMV use case (Fig. 2), where a sequential Kalman filter application for orbit determination has been converted through the use of the SPMD pattern.

The development methodology of the GMV use case follows the same principle as does every swarm application written with PTB-FLA, correctness by construction. After each integration phase (i.e. converting a sequential piece of code into a distributed one), the validated output

[10] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, and I. Basicevic, A Federated Learning Algorithms Development Paradigm, in: J. Kofron, T. Margaria, C. Seceleanu (Eds.), Engineering of Computer-Based Systems, Lecture Notes in Computer Science, Vol. 14390, Springer, Cham, 2024, pp. 26–41, https://doi.org/10.1007/978-3-031- 49252-5_4

[11] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, I. Basicevic, Developing Elementary Federated Learning Algorithms Leveraging the ChatGPT, in: Proceedings of the 31st Telecommunications Forum (TELFOR 2023), IEEE Xplore, 2023, pp. 1-4, https://doi.org/10.1109/TELFOR59449.2023.10372714.

Funded by the European Union

of that phase remains unchanged. This incremental approach ensures that a complex swarm application is constructed in a managed manner, where each added element introduces new features. The end result is a fully distributed swarm application, which performs autonomous orbit determination and time synchronization (ODTS) for a large constellation of satellites in LEO.

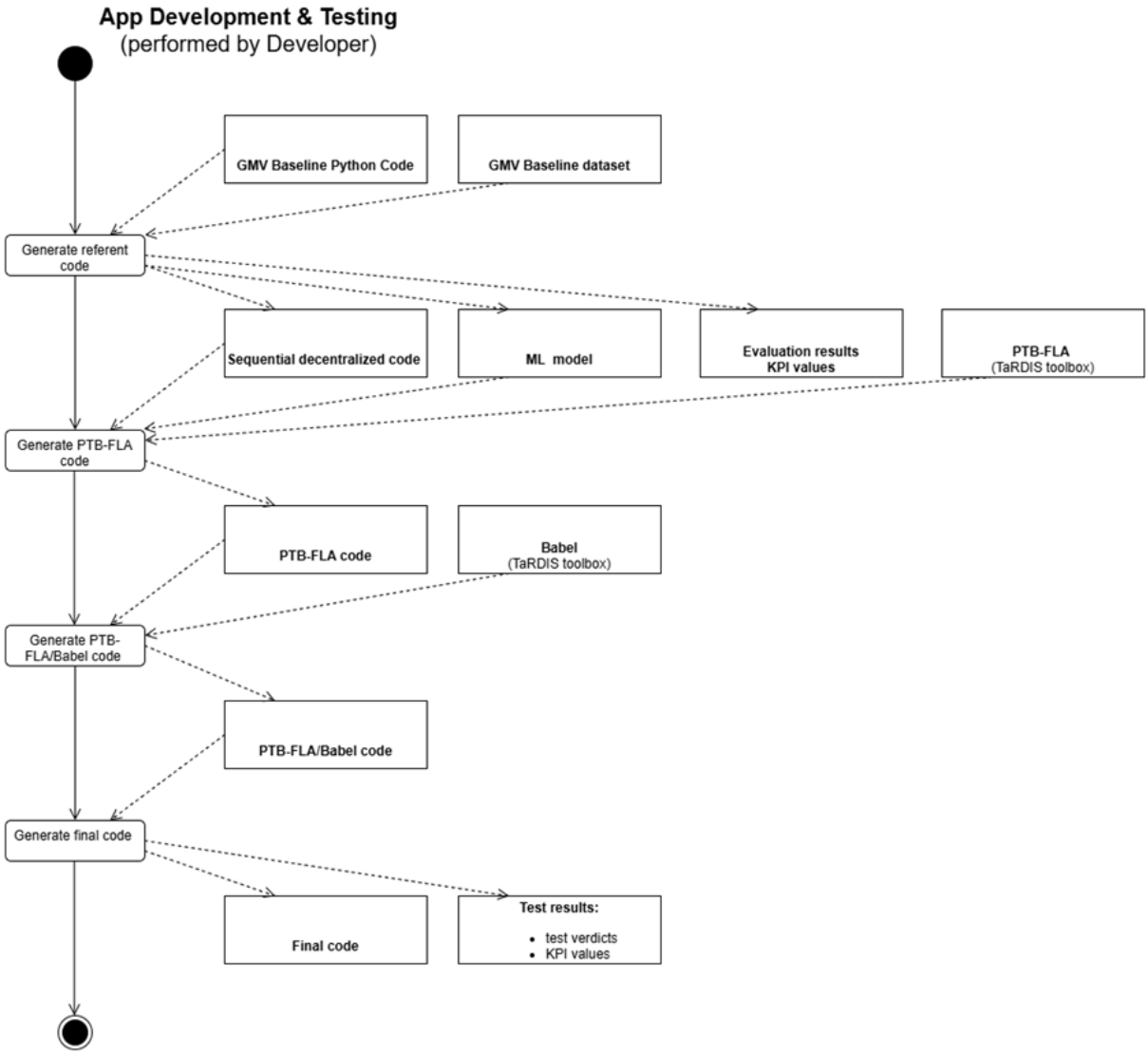


Fig 2: GMV Use Case development methodology activity diagram

The execution of application instances on a single host is handled by the integrated PTB-FLA launcher, whereas in the case of MPT-FLA, which supports distributed execution across a network a dedicated distributed launcher is provided as a separate tool. Additionally, the PTB-FLA toolkit is enriched by the TaRDIS IDE integration: in fact, the IDE provides a way to select elements of PTB-FLA for application development, as well as means to launch local application instances through a GUI.

Using the combination of proven correct abstractions and clearly formulated development methodology, PTB-FLA-based swarm applications achieve correctness by construction in a structured, reliable, and highly managed way, improving the overall development experience, which seems to be the main setback whilst developing swarm applications.

Funded by the European Union

PTB-FLA and its examples are publicly available in the Github Repository[12], while its API descriptions are available on the TaRDIS wiki.[13]

2.5. Free-Form TaRDIS Swarm Elements and Applications

Note: this section also appeared in a similar form in D3.3 (as Section 2.4); we repeat the contents here to make this document self-contained while presenting the new functionalities added to some of TaRDIS tools.

A free-form TaRDIS swarm element is a program that does not delegate its main execution loop to the TaRDIS execution engine, and does not follow the TaRDIS swarm specification-based development approach outlined in Sections 2.2 and 2.3. A free-form TaRDIS swarm element might decide to directly control its main execution loop (or delegate it to other libraries, e.g. Babel described below, or GUI toolkits like Qt), and may use only selected (and typically lower-level) TaRDIS APIs for specific purposes - e.g. producing or awaiting some events, accessing communication or AI/ML primitives. Generally speaking, a free-form TaRDIS swarm element will use the lower-level APIs provided by the TaRDIS toolbox, and may not take advantage of all the TaRDIS verification facilities (especially those for managed swarm elements outlined in Sections 2.2 and 2.3).

In the context of TaRDIS, free-form applications are typically developed resorting to **Babel-Swarm** or **Babel-Android**, two evolutions of the Babel framework (whose origin predates TaRDIS) that were produced in the context of TaRDIS WP6. The **Babel framework (T-WP6-04)** was designed to aid in the development, prototyping, and execution of distributed protocols with a focus on performance and dependability. Its primary goal is to simplify the creation of distributed algorithms by abstracting away complex low-level aspects, such as communication handling, timeouts, and concurrency management. This allows researchers, practitioners, and educators to develop and experiment with distributed systems and protocols without being overwhelmed or delayed by implementation details. Babel is particularly suited for protocols requiring fault-tolerance, enabling more efficient testing and comparison of various solutions, and to develop abstractions that can be easily reused in the context of different distributed applications. This makes Babel an attractive solution to support free-form TaRDIS swarm elements.

Babel promotes an event-driven programming model where protocols can interact through a structured set of operations. It provides mechanisms for developing handlers for events like timers, network messages, and inter-protocol communication (requests, replies, and notifications), all of which are asynchronous. Babel's design ensures that protocols can be executed independently within the same process, with a dedicated thread for each protocol. Evidently, for this to be achieved, protocols relinquish the control of their main loop to the Babel framework, and handlers for all types of events can never block this main execution thread of the protocol. This model not only simplifies the process of translating algorithmic specifications into prototypes but also enhances performance by efficiently managing concurrency and resource utilization, shielding programmers from potentially complex concurrency issues.

Babel's main properties include flexibility, modularity, and adaptability to different network configurations. Its networking abstraction, called "channels," allows for various communication methods (e.g., P2P or client-server) that can be tailored to specific protocol needs. Additionally, Babel's architecture supports the independent development and execution of protocols, which encourages reuse and scalability. The framework's flexibility in managing protocol interactions

[12] https://github.com/miroslav-popovic/ptbfla

[13] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs

Funded by the European Union

and network communications makes it suitable for diverse use cases, from peer-to-peer systems to consensus protocols, demonstrating competitive performance while maintaining ease of development.

In the context of TaRDIS, we developed two new variants of this framework, respectively called Babel-Swarm and Babel-Android. These new variants extend the functionality of Babel with features that are relevant for swarm applications, being suitable to run on a wide variety of devices, including mobile phones, small computers (e.g., raspberry pi), laptops, desktops, or even servers.

Babel-Swarm enhances the original Babel by introducing features for self-configuration, autonomic reconfiguration, and security. This variant supports the automatic discovery of network contacts, allowing swarm nodes to join and configure themselves with minimal user intervention, avoiding common human errors. It also enables protocols to adapt runtime parameters in response to changing network conditions, system workload evolution, or other operational conditions, improving performance and resilience. Moreover, Babel-Swarm incorporates security mechanisms, such as node authentication via self-signed certificates and secure communication channels, essential for applications that require trustworthy interactions within swarms. All these features are exposed to programmers in simple ways, and entwined with the usual operation of the Babel. This will be further detailed in the upcoming Deliverable D6.2 (as the Babel developments are part of WP6).

Babel-Android ports the Babel-Swarm framework's functionalities to mobile environments, allowing developers to build distributed protocols optimized for Android devices. This adaptation enables Babel's event-driven model to run on mobile devices, thereby supporting peer-to-peer and edge-computing applications that require direct device-to-device communication in a decentralised manner.

In addition to Babel-Swarm and Babel-Android, we are also developing **Micro-Babel**, a lightweight variant of the framework designed specifically for highly constrained environments such as microcontrollers and other embedded devices. Unlike its counterparts, which are implemented in higher-level languages and target more resource-rich environments (mobile phones, laptops, or servers), Micro-Babel is written in C and optimized for minimal memory footprint and efficient execution on platforms where computational power, storage, and energy are scarce. The design of Micro-Babel focuses on providing a streamlined subset of the Babel abstractions, retaining the essential event-driven programming model while simplifying or omitting features that are less critical in embedded contexts. This allows developers to deploy distributed protocols on low-cost hardware, such as IoT nodes, wireless sensors, or actuator devices, without requiring extensive runtime support.

Funded by the European Union

3 TaRDIS APIs STATUS AND PROGRESS

This section outlines the ongoing work on the design and development of the TaRDIS toolkit APIs. Here we use the term “API” in a broad sense, covering all facilities that will be made available to programmers that use the TaRDIS toolkit to develop swarm applications; this includes both APIs in the “classic” sense (i.e. the specification of functions, procedures, and methods callable by user’s code), and facilities and tooling supporting the programmer (e.g. the verification facilities developed in WP4).

This section is structured as a status update with respect to Section 3 of Deliverable D3.3: it focuses on the improvements and changes occurred since then, and refers to the relevant sections of the TaRDIS wiki for further technical details and documentation. For easier readability, the section numbers match the section numbers of D3.3. The titles of most subsections highlight which WP and task is working on the related APIs and features of the TaRDIS toolbox.

- Section 3.1 outlines the TaRDIS APIs.
- Section 3.2 outlines the analysis and verification facilities.
- Section 3.3 outlines the APIs for AI and machine learning-related functionality.
- Section 3.4 outlines the data management and distribution primitives.

3.1. APIs Outline

This section provides an overview of the TaRDIS swarm APIs. The structure follows the corresponding section of D3.3. Overall, there have been no major design/architectural updates since D3.3, and most of the work on TaRDIS has focused on toolkit implementation improvements tailored towards the use cases implementation and evaluation (as part of WP7).

These APIs, sketched below, are divided between those conceived for managed swarm elements (Section 3.1.1), those conceived for free-form swarm elements (Section 3.1.2), and those giving access to machine learning functionality (Section 3.1.3). The APIs deal with aspects such as the creation and validation of a swarm protocol, the creation of a role in a swarm protocol, the creation of a workflow to handle that role, finding individuals running specific workflows or that are part of a specific role, creating communication overlays and channels, and handling events and messages.

3.1.1 Event-Driven APIs for Managed Swarms: Instantiating a TaRDIS Swarm

The managed swarm element APIs, as part of the core TaRDIS APIs were defined early in the project and are documented in the TaRDIS Wiki.[14]

[14] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Core-TaRDIS-APIs

3.1.2 Event-Based Input-Output APIs for Free-Form Swarm Elements

The free-form swarm elements I/O APIs, as part of the core TaRDIS APIs were defined early in the project and are documented in the TaRDIS Wiki, which is updated throughout the project.[13]

3.1.3 Machine Learning APIs

The machine learning APIs, as part of the core TaRDIS APIs were defined early in the project and are documented in the TaRDIS Wiki.[13]

3.2. Analysis and Verification Facilities

This section summarises the status of the APIs related to program analysis and verification. The structure of the following subsections is based on the TaRDIS WP4 project tasks (matching the structure of Section 3.2 of Deliverable 3.3):

- Task 4.1 - Specifying and Verifying Communication Behaviour (Section 3.2.1)
- Task 4.2 - Specifying and Analysing Data Consistency (Section 3.2.2)
- Task 4.3 - Specifying and Analysing Security Properties (Section 3.2.3)
- Task 4.4 - Specifying and Analysing Security Properties (Section 3.2.4)

3.2.1. Specifying and Verifying Communication Behaviour - T4.1

3.2.1.1. Correctness-By-Construction Guarantees for Managed TaRDIS Applications

Since the submission of Deliverable D3.3, the support for the development of managed TaRDIS applications has been improved by refining the tooling for two kinds of specifications:

- State-oriented specification of managed TaRDIS applications via swarm protocols (outlined in Section 2.2). The corresponding tooling is documented on the Actyx developers website, and on the TaRDIS wiki. The tooling includes a graphical editor for swarms protocols called **WorkflowEditor (T-WP3-01)** and the **machine-runner (T-WP4-01)** and **machine-check (T-WP4-02)** libraries. The underlying swarm communication is handled by the **Actyx middleware (T-WP6-03)**.[15] [16] Since D3.3, the tooling **has been extended** with bugfixes, performance improvements, and by adding support for the compositional specification and development of swarms. This new functionality allows for creating large swarms by automatically adapting and reusing smaller swarms, while maintaining the correctness properties guaranteed by the framework. We have developed initial documentation and a demonstration.[17]
- Event-oriented specification of managed TaRDIS applications via DCR choreographies (outlined in Section 2.3), supported by the tool **DCR Editor (T-WP3-03)**[18] [19] and DCR Compiler (Part of the DCR Editor) and the integrated (Sec)ReGraDa-IFC verification

[15] https://developer.actyx.com/
[16] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/The-TaRDIS-programming-model/Actyx-tools
[17] https://people.compute.dtu.dk/alcsc/tmp/tardis-swarm-proto-docs-artifact.zip
[18] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/The-TaRDIS-programming-model
[19] https://codelab.fct.unl.pt/di/research/tardis/wp3/TaRDIS-DCR/-/wikis/home

tool (**T-WP4-11**) documented in the TaRDIS DCR wiki. The choreographies are built with the DCR Editor, integrated in the TaRDIS Integrated Environment using either a visual language, or upon choice of the developer, a textual language. Syntactic and semantic verifications are performed during the construction of the choreography, like the existence of senders and receivers, type checking of data exchanged in messages, and more importantly, information flow security, guaranteeing that no unauthorized participants in the choreography can see, or infer events and data that are not in their security compartment, given by the role they have on the system. Since D3.3, the tooling **has been improved and extended** with bugfixes and new features of the language being supported, such as spawn relations and parametric specification of participants in messages. These new functionalities have significant impact in the projection algorithm, allowing for a more dynamic creation of large swarms and extending the correctness and security properties guaranteed by the information flow control (IFC) checker. We have developed initial documentation and a demonstration, available in the DCR TaRDIS repository.

3.2.1.2. Communication Behavioural Properties

To specify and verify communication properties such as communication safety, deadlock freedom, and liveness, an extensible toolchain based on Multiparty Session Type (MPST) theory known as **Scribble (NuScr, T-WP3-02 and T-WP4-05)** is utilised. This toolchain provides a language for defining global communication structures, also known as global protocols. It allows manipulation of these protocol specifications to generate APIs that can be directly implemented in distributed systems, ensuring critical safety guarantees.

The Scribble toolchain is employed specifically within Task T4.4, which addresses the verification of membership protocols and communication primitives designed under WP6. These distributed protocols are crucial for constructing and maintaining overlay networks, which typically have numerous liveness properties and limited safety properties due to their probabilistic nature. Scribble's mechanisation ensures these liveness properties are verified, contributing to the robustness of the distributed systems.

Since D3.3, the Scribble toolchain has continued to be used in Task T4.4 for modelling and verifying communication protocols based on namespaces, supporting the WP6 tool for distributed configuration management (see Section 3.2.4 for details).

Scribble is available as open-source software on GitHub[20] and can be used both as a standalone command-line application and as a library for integrating multiparty protocol handling into other projects. Additionally, it offers a web-based interface for direct protocol prototyping without requiring local installation.[21] Initial documentation for Scribble is available on the TaRDIS wiki.[22]

3.2.1.3. Join Pattern Matching API with “Fair Matching” Guarantees

The **JoinActors library (T-WP3-03)** implements join patterns in Scala 3. Join patterns are a coordination mechanism for concurrent message-passing programs allowing to declaratively

[20] https://github.com/nuscr/nuscr
[21] https://nuscr.dev/nuscr
[22] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/Scribble

Funded by the European Union

specify how to react to combinations of incoming messages, and synchronize distributed computations. The library will be used in the implementation of the Actyx use case.

Since Deliverable D3.3, the JoinActors library has been extended with optimised pattern matching algorithms, leading to significant performance improvements w.r.t. the original publication.[23] The API has also been improved since D3.3.

The JoinActors library is available as Open Source software on GitHub,[24] and has some preliminary documentation available on the TaRDIS wiki.[25]

3.2.1.4. P4R-Type: Verified Control Plane API for Software-Defined Networking

As reported in Deliverable D7.2 (Section 4.4), the **P4R-Type library (T-WP4-04)** for verified software-defined networking in Scala 3 will not be included in the TaRDIS toolbox.

3.2.1.5. Typestate Checking

In software systems, resources are stateful and operations performed on them may depend on properties of the state. For instance, one cannot pop from an empty buffer or withdraw from an account without sufficient balance. These properties about state, when declared in the code implementing the system, are called *typestates*. A simple way of representing them is like finite automata, where transitions from a certain state correspond to operations that can be performed safely (i.e., without eventually crashing the application or leading it to an incoherent state).

The key idea of **JaTyC**[26] **(T-WP4-06)** is to associate a typestate annotation with every stateful class, declaring the object's states, the methods that can be safely called in each state, and the states resulting from the calls. The tool statically verifies that when a Java program runs: sequences of method calls obey to object's protocols; objects' protocols are completed; null-pointer exceptions are not raised; subclasses' instances respect the protocol of their superclasses. Moreover, it supports protocols to be associated with classes from the standard Java library or from third-party libraries and supports "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.

The role of JaTyC in TaRDIS is to provide support for Free-Form development with the TaRDIS toolbox. Currently, a Proof-of-Concept integration with Babel is underway.

Since Deliverable D3.3, we developed a tutorial presentation of JaTyC, showing step-by-step how to develop an distributed application with it,[27] and developed a full formal presentation of the type system that tackles a very significant subset of the Java language.[28]

[23] Philipp Haller, Ayman Hussein, Hernán C. Melgratti, Alceste Scalas, Emilio Tuosto: Fair Join Pattern Matching for Actors. ECOOP 2024: 17:1-17:28. https://doi.org/10.4230/LIPIcs.ECOOP.2024.17
[24] https://github.com/a-y-man/join-actors
[25] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/JoinActors
[26] https://github.com/jdmota/java-typestate-checker
[27] Lorenzo Bacchiani, Mario Bravetti, Marco Giunti, João Mota, and António Ravara: Automated Verification of Stateful Java Programs with JaTyC. Submitted
[28] Lorenzo Bacchiani, Mario Bravetti, Marco Giunti, João Mota, and António Ravara: Typestate Checking in Java: A Foundational Approach. Submitted

Funded by the European Union

3.2.2. Specifying and Analysing Data Consistency - T4.2

3.2.2.1. AtomiS - Data Centric Concurrency (an extended Java Compiler)

Data-Centric synchronisation (DCS) shifts the reasoning about concurrency restrictions from control structures to data declaration. It is a high-level declarative approach that abstracts away from the actual concurrency control mechanism(s) in use. **AtomiS**[29] **(T-WP4-07)** requires only qualifying types of parameters and return values in interface definitions, and of fields in class definitions. The latter may also be abstracted away in type parameters, rendering class implementations virtually annotation-free. From this high level specification, a static analysis infers the atomicity constraints that are local to each method, considering only the method variants that are consistent with the specification, and performs code generation for all valid variants of each method. The generated code is then the target for automatic injection of concurrency control primitives that are responsible for ensuring the absence of data-races, atomicity-violations and deadlocks. In short, AtomiS is used to mark resources which need to be accessed in mutual exclusion; a type-checking and inference system ensures race freedom.

This tool has no dependencies with other TaRDIS tools, but can be used in subsequent releases of some of them. When developing concurrent applications that share resources, or in particular in the case of orchestrated TaRDIS tools like FAUNO (WP5-05), Babel (WP6-04), or Distributed Management of Configuration based on Namespaces (WP6-08), a critical aspect is the identification of the right concurrency control features and where to place them. AtomiS can thus be used to optimise and produce by-construction thread-safe versions of these tools. An evaluation of the readiness and effectiveness of AtomiS is under way, comprising a compilation scheme compatible with Java and the JVM.

Work on the foundations of the approach are underway. **Since Deliverable D3.3,** we established a roadmap to mechanically prove the algorithms sound and closed the first set of significant proofs (MSc thesis just submitted).

3.2.2.2. Ant - Anticipation of Method Execution in Mixed Consistency Systems

Distributed applications (widely common these days) need to replicate data to make it available. Eventually, possible conflicts must be solved. A typical example is a shared set: inserts can always happen, as sets do not have repeated elements (although the local view of the set may be outdated), but removals require causal and/or eventual "coordination" (if one cannot remove a non-existing value, as in some contexts, this can block or crash the device).

Ant (T-WP4-08) is an approach to determine statically operations that can safely commute with other operations in replicas of a distributed system. The information is used to allow a run-time system to anticipate calls to commutable operations. The theory behind is described in papers published recently[30] [31]. The aim is to reduce the programmer's effort by only requiring simple and intuitive annotations at data declaration. The goal is to use the annotations to automatically identify all accesses to replicated data; operations accessing such data are either conflict-free with other operations or may require coordination.

[29] Hervé Paulino, Ana Almeida Matos, Jan Cederquist, Marco Giunti, João Matos, and António Ravara: *AtomiS: Data-Centric Synchronization Made Practical*. OOPSLA 2023: 116-145. https://dl.acm.org/doi/10.1145/3622801
[30] https://doi.org/10.1145/3555776.3577725
[31] https://arxiv.org/abs/2212.14651

Funded by the European Union

The Ant approach as been implemented in a tool - REPL[32] that performs compile-time commutativity analysis for the Java language, computing the commutativity of pairwise method calls from the input given at data declaration, parameters' values and fields' states.
This tool has no dependencies with other TaRDIS tools. The approach and/or the tool can be particularly useful to control at runtime the execution of operations in the swarm replicas, ensuring eventual data-consistency. Since Deliverable D3.3, we have been working on a version of the tool expressive enough to cover a significant subset of Java.

3.2.2.3. VeriFx

The tool **VeriFx (T-WP4-09)** has been developed as a framework for the design and verification of replicated data types (RDTs), with a focus on providing strong formal guarantees. It features a specialized domain-specific language (DSL) that enables developers to define RDTs with automated proof capabilities, ensuring their correctness. Verified RDTs can be transpiled into mainstream programming languages, such as Scala and JavaScript, to facilitate their integration into real-world systems. The tool also includes libraries for implementing and verifying Conflict-free Replicated Data Types (CRDTs) and Operational Transformation (OT) functions, which are critical for ensuring consistency in distributed and collaborative environments. Currently, we have been exploring the adaptation of the tool for analyzing the EDP and GMV use cases using VeriFx specifications, with a focus on defining useful RDTs tailored to these applications.

The tool is still under development but an initial version is available in Zenodo,[33] and some preliminary documentation is available on the TaRDIS wiki.[34]

Convergence and Invariants without Coordination. A relevant new contribution **implemented in VeriFix after D3.3** is the No-Op tool, based on an approach described in a recent publication.[35] No-Op is a sound methodology to develop with replicated data-types. A detailed presentation of the approach can be found in deliverable D4.3.

3.2.3. Specifying and Analysing Security Properties - T4.3

The tools **(Sec)ReGraDa (T-WP4-11)** and **IFChannel (T-WP4-08)** provide security verification in TaRDIS applications that are defined as DCR graphs and as a general program, respectively. This is a form of information flow analysis, roughly checking that data of a given security label cannot flow (by programmer mistake) into a location of a lower security label. This includes both explicit flows (writing data into a location of lower security) and implicit flows (conditional writing operations where the condition depends on higher-level data than the writing data). This is meant as a static analysis, i.e., the analysis is performed before the execution is deployed, not as a dynamic/monitoring analysis, as it is impossible to prevent implicit flows in the dynamic setting. **(Sec)ReGraDa has been implemented as a procedure for DCR graphs and used on the EDP use case; this is with the assumption that sending messages of a given security level *l* over a secure channel is like writing into a variable of level *l*. The IFChannel provides the formal justification for this, i.e., that sending the message encrypted with a key of security level *l* over an insecure channel is providing similar information flow guarantees. Here we have to accept, however, that, without the use of onion-routing, an attacker who can observe messages being exchanged can also learn some information about the workflow, e.g., linking messages between the same partners. This is part of an ongoing investigation into further hardening channels**

[32] http://hdl.handle.net/10362/164248
[33] https://zenodo.org/records/7982416
[34] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/VeriFx
[35] https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ECOOP.2025.4

Funded by the European Union

without going to full onion-routing; at least some basic results are expected to be completed within the runtime of TaRDIS.

At the core of this analysis is the concept of information-flow-secure channels. The IFChannel module extends standard information flow results to using secure channels that are implemented cryptographically. The idea is that we can transmit confidential information also over public/insecure channels that may be controlled by an intruder, as long as the data is suitably encrypted. We currently assume that these encryption operations are with suitable long-term keys (public-key encryption or symmetric-key encryption), but even under this restriction, it is far from trivial to prove that this is secure. In fact, we follow here the standard concept of non-interference: given two states of the system that only differ in data that is above the security level *l* of an intruder, then executing any verified TaRDIS program produces two states that only differ in data above security level *l*; therefore, the intruder is unable to distinguish them, and thus unable to learn anything about the data above the intruder's security level *l*. We have to make however a major adaptation to the standard notion of non-interference, since it is trivially broken if the intruder can break cryptography; we have thus made a Dolev-Yao style intruder model where the intruder cannot break cryptography, but can of course make observations like the transmission of messages. This actually also means that the verification needs to check that transmissions do not depend on a non-public security level. **We now have proved this security result in the proof assistant Isabelle/HOL: if a program using cryptographic channels satisfies the information flow conditions, then our non-interference holds, i.e., an intruder who cannot break cryptography cannot learn anything about the secret data. The check that the information flow holds is completely automated within the Isabelle/HOL environment for a while-programming language. The adaptation to a larger programming language and integration with Babel is ongoing.**

A point that we have not completely solved so far is integrity, which can be done by a complementary information flow analysis. Indeed, the information flow analysis itself is simple, but the non-interference result for integrity does not follow in general if we have to assume that an intruder can block and replay messages on the network. There are some mechanisms to ensure consistency, but they are often not directly compatible with swarm protocols we want to deploy, and in fact not even needed when we only require eventual consistency. For example in the EDP use case, the intruder may block a request or offer of energy, but cannot manipulate the requested amount or offered price. We plan to define suitable requirements for programs so that we can guarantee the integrity goals.

The verification of the DCR graphs requires that channels are implemented by cryptographic protocols; these protocols typically have a setup part (like TLS handshake) and a transmission part (like TLS transport). Moreover, there may be administrative protocols for distributing and updating keys, e.g., when swarm nodes join or leave a given group. These protocols should not be for the TaRDIS users to specify and implement, but available through the TaRDIS library in the Babel framework. To verify these protocols, we will use the tool **PSPSP (T-WP4-09)**, an existing tool for protocol verification that we want to adapt for the internal use of the TaRDIS project. The planned adaptations are: 1. the integration with the model of security labels from **T-WP4-8/11**; 2. the support for an abstract payload type according to vertical compositionality results, i.e., verifying that a channel provides secure transmission for arbitrary payload messages, even if these are known or chosen by the intruder; 3. improvements of the user interface; 4. possibly support for algebraic properties.

Finally, we also want to explore another synergy between the research groups at DTU and NOVA, namely the use of cryptographic choreographies (via the tools **IFChannel, T-WP4-10, and CryptoChoreo, T-WP4-12, and (Sec)ReGraDa DCR, T-WP4-13**). This allows to equip

Funded by the European Union

choreographies (similar to DCR graphs) with cryptographic operations and derive from them the program that implements this locally for the different roles of the choreography. This is a correct-by-construction approach: we check that the choreography is actually executable (e.g., preventing mistakes where, under certain circumstances, one party sends a message that cannot be received by another) and that all agents perform all checks that they can make (e.g., preventing mistakes where a programmer forgets to make some possible checks on a received message). We plan to deploy this to describe the protocols that we want to verify with **PSPSP (T-WP4-09),** but we are also considering directly integrating it into the DCR graphs and the generation of implementation used there. **We have devised a formal semantics as a translation from Choreographies to local behaviours of each role; this parameterised over two problems on algebraic theories of the cryptographic operators (e.g., the properties of exponentiation in the use of Diffie-Hellman). While in general these problems are undecidable, for many theories of interest, they are efficiently decidable. The implementation of this tool is ongoing.**

Documentation about IFChannel, PSPSP, CryptoChoreo, (Sec)Regrada, and DCR Choreographies may be found in the TaRDIS wiki.[36] [37] [38] [39] [40]

3.2.4. Deployment and Orchestration Integration - T4.4

The work conducted in T4.4 is directly related to the tools developed in WP5 and WP6:

- WP5 develops the tool **PTB-FLA** (**T-WP5-04**, Section 3.3.1.1) that provides correct orchestration of two federated learning algorithms (centralised and decentralised) and two TMD communication based algorithms (pairwise and universal). All these algorithms were formalised in communicating sequential processes (CSP) algebra and verified by the Process Analysis Toolkit (PAT), in the following steps: (1) in D3.1, the two federated learning algorithms were formalised using the ad hoc approach, (2) in D3.3, the two federated learning algorithms were formalised using an approach based on systematic construction of CSP models from Python code that follows a restricted actor-based programming model, and (3) **after D3.3**, the two TMD communication based algorithms were formalised using the ad hoc approach. We would like to remark that the approach used in step 2 above could serve as a basis for developing a tool for the automatic translation of certain classes of Python code to CSP models. In addition to the research based on CSP and PAT, after D3.3, we started working on the session typing theory to support the modelling and verification of processes that implement federated learning protocols, including those in PTB-FLA. We built upon the asynchronous "bottom-up" session typing approach by adding support for input/output operations directed towards multiple participants at the same time. We enhanced the flexibility of the introduced typing discipline and allowed for safe process replacements by introducing a session subtyping relation tailored for this setting. We formally proved several safety and liveness properties.

[36] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/IFChannel
[37] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/PSPSP
[38] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/CryptoChoreo
[39] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/(Sec)Regrada
[40] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/DCR-Choreographies

Funded by the European Union

- After D3.3, we continued our work on supporting the WP6 tool for the distributed management of configurations. This tool relies on a model based on namespaces. The existing model is extended by introducing a new dataspaces layer and protocols for data discovery using the concept of hierarchical namespaces and transforming non-shareable resources, such as files and folders, into collaborative resources. In T4.4, we worked on this model in two directions: (i) ensuring that the model provides accurate access and redistribution of shareable (CPU, RAM, disk) and non-shareable resources through graph transformation theory, and (ii) modelling and verifying communication protocols, by applying multiparty session types and the Scribble tool (T-WP3-02 and T-WP4-05).

3.3. Artificial Intelligence and Machine Learning APIs

This section summarises the status of the APIs related to AI and machine learning. The structure of the following subsections is based on the TaRDIS WP5 project tasks (matching the structure of Section 3.3 of Deliverable 3.1):

- Task 5.1 - AI/ML Programming Primitives (Section 3.3.1)
- Task 5.2 - AI-Driven Planning, Deployment, and Orchestration (Section 3.3.2)
- Task 5.3 - Lightweight and Energy-Efficient ML Techniques (Section 3.3.3)

3.3.1. AI/ML Programming Primitives - T5.1

3.3.1.1. Python and MicroPython Test Beds for Federated Learning Algorithms (PTB-FLA and MPT-FLA) APIs

Python and MicroPython Test Beds for Federated Learning Algorithms (**PTB-FLA and MPT-FLA, T-WP5-04**) APIs implementation is publicly available in the ptbfla GitHub repository.[41]

The Python Test Bed for Federated Learning Algorithms (PTB-FLA) API is provided by the PtbFla class in the ptbfla module, whereas the MicroPython Test Bed for Federated Learning Algorithms (MPT-FLA) API is provided by the PtbFla class in the mp_async_ptbfla module. (Note: both classes have the same name to lighten porting legacy applications from PTB-FLA to MPT-FLA.) Both APIs include a constructor, two generic Federated Learning Algorithms (FLAs), two generic algorithms for TDM (Time Division Multiplexing) communication i.e., peer data exchange, and a destructor. The MPT-FLA API also includes the start method. (Note: all the MPT-FLA API methods, except the constructor and the destructor, are Python asyncio coroutines, so they should not be called as functions but must be awaited using the await keyword.)

Both generic FLAs (i.e., fl_centralized and fl_decentralized methods) and both generic algorithms for TDM communication (i.e., get1Meas and getMeas) have been formalised using CSP (Communicating Sequential Process) calculus and verified using the model checker PAT (Process Analysis Toolkit). Two key properties were checked:

- **Deadlock Freedom (Safety):** Ensures that the system will not reach a state where no progress is possible.
- **Termination (Liveness):** Ensures that the system will eventually complete its tasks.

[41] https://github.com/miroslav-popovic/ptbfla

Funded by the European Union

More details on the PTB-FLA and MPT-FLA APIs are available in the TaRDIS wiki.[42]

3.3.1.2. Flower-based Federated Learning (FFL) APIs

Three Flower-based federated learning APIs within the TaRDIS toolkit were introduced in D3.1 and initially described in D3.3. These are:

- the **Flower-based Federated Learning training models API** (**T-WP5-01**, FFL training models API);
- the **Flower-based Federated Learning input data preprocessing API** (**T-WP5-02**, FFL input data preprocessing API);
- the **Flower-based Federated Learning Machine Learning inference and evaluation API** (**T-WP5-03**, FFL ML interface and evaluation API).

These APIs have been implemented through corresponding tools, as integral parts of the TaRDIS architecture: The Flower-based FL model training tool (T-WP5-01), the Data preparation for Flower-based FL model training tool (T-WP5-02) and the Flower-based FL model inference and evaluation tool (T-WP5-03). The unified Flower-based FL tool has been introduced in D5.2, as a synthesis of the mentioned APIs. The tool enables the usage of model preparation, training and evaluation in one place. It is built upon the open-source Flower framework and offers a set of custom FL solutions.

The FFL training models API enables FL ML model training, by supporting a list of provided AI/ML decentralised solutions. Beside the implementations already reported in D3.3 (FedAvg, pFedMe, pFedMeNew, Anomaly detection and Distributionally robust FL), the tool has been complemented with some new directions. Two solutions have been implemented for the anomaly detection problem with noisy labels, a problem defined in the context of the ACT use case. The first solution was motivated by the HUNOD[43] method (Hybrid UNsupervised Outlier Detection), which integrates two machine learning methodologies: clustering using the K-means algorithm and representational learning through autoencoders. The method is aimed to provide a robust solution to scenarios with inexact or noisy labels. The tests on public data showed promising results (see D5.2). However, in order to further enhance the robustness and accuracy of anomaly detection, an anomaly transformer[44] based model was also implemented and integrated in the tool. This is currently being tested on synthetic data. Additionally, some existing functionalities have been refined and complemented, regarding the Flower-based FL tool. For instance, the tool has been adjusted to become deployment-ready for real edge device clients. Also, a lightweight technique, knowledge distillation has been integrated to the tool, resulting from the collaboration between T5.1 and T5.3.

The FFL input data preprocessing API is responsible for supporting the preparation and transformation of raw data into a format that is suitable for the FL ML model training process. The FFL ML inference and evaluation API provides valuable output regarding the relevant data and the trained model. The tool contains preprocessing and evaluation/inference capabilities corresponding to the supported FL approaches. The detailed descriptions of the novelties regarding the different aspects of the Flower-based FL tool, will be described in D5.3.

The Flower-based FL tool is designed in a user-friendly manner, by providing a GUI-based interface (beside the command-line approach), that guides the user during setting up the

[42] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs

[43] M. Savic, J. Atanasijevic, D. Jakovetic, N. Krejic. Tax evasion risk management using a Hybrid Unsupervised Outlier Detection method. Expert Syst. Appl. 193, 2022, https://doi.org/10.48550/arXiv.2103.01033

[44] J. Xu, H. Wu, J. Wang, M. Long. Anomaly Transformer: Time Series Anomaly Detection with Association Discrepancy. arXiv preprint arXiv:2110.02642, 2022, https://doi.org/10.48550/arXiv.2110.02642

Funded by the European Union

training process. It is aligned with the mentioned APIs, but represents an integrated solution, where even non-expert users can easily train an FL model. More details on the FFL APIs are available in the TaRDIS wiki.[45]

3.3.1.3. Fedra (T-WP5-09): A decentralised federated learning framework enabling secure P2P model training on edge devices

This is a new TaRDIS toolbox component w.r.t. Deliverable D3.1. **Fedra (T-WP5-09)** provides a decentralised federated learning framework integrated with p2p communications between the participating nodes, specifically designed for swarm systems. Fedra is model-agnostic, in the sense that different ML algorithms can be seamlessly integrated and utilised to train ML federated models, including models for forecasting, resource allocation, anomaly detection.

Fedra's architecture involves the orchestration of several components, each playing a crucial role in the decentralised learning process:

- **P2P Communication Layer (P2PHandler)**: (i) Acts as the nervous system of the Fedra network; (ii) Manages all inter-node communications, from initial peer discovery to ongoing model update exchanges; (iii) Utilises libp2p to ensure secure, efficient, and anonymous peer-to-peer interactions.
- **Data Management and Preprocessing (DataLoaderHandler)**: (i) Serves as the sensory input system, preparing and feeding data to the learning models; (ii) Handles diverse data types and structures, ensuring compatibility across different learning tasks; (iii) Implements advanced preprocessing techniques to optimize learning efficiency.
- **Core Operations Module (Operations)**: (i) Functions as the brain of each node, performing critical computations; (ii) Manages serialization and deserialization of model updates, crucial for efficient network transmission; (iii) Implements the federated averaging algorithm, the key to collaborative learning in a decentralised setting.
- **Network State Management (NetworkState)**: (i) Acts as the collective memory of the network; (ii) Keeps track of the status and contributions of all participating nodes; (iii) Enables informed decision-making for adaptive learning strategies.
- **Orchestration Engine (Main Script - fedra.py)**: (i) Serves as the conductor, coordinating all components to work in harmony; (ii) Manages the lifecycle of the learning process, from initialization to convergence; (iii) Implements high-level learning strategies and protocols.

This architectural design ensures that Fedra is not just a tool for federated learning, but a comprehensive ecosystem for decentralised, collaborative AI development. More information related to the operation of Fedra can be found in the TaRDIS wiki[46] and on GitHub[47].

3.3.1.4. FLaaS (T-WP5-10): Federated Learning as a Service

FLaaS (FL-as-a-Service), developed by Telefónica, is a practical FL framework for mobile environments that allows app developers to perform cross-device and cross-app FL. As mentioned in Deliverable D7.2 (section 2.3.8), the initial version of FLaaS was built outside of the TaRDIS project, and the development and subsequent integration of the TaRDIS tools into FLaaS resulted in an improved and more modular version of FLaaS.

[45] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs

[46] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs

[47] https://github.com/anaskalt/fedra

The baseline FLaaS provided an orchestration layer for distributing, aggregating, and monitoring model updates across heterogeneous devices, demonstrating the feasibility of deploying federated learning in telecom and IoT environments. However, limitations in privacy, scalability, efficiency, and sustainability highlighted the need for an improved version.

System Enhancements

The improved FLaaS integrates three TaRDIS components (used as part of the Telefónica use case within the TaRDIS project):

- **Privacy-preserving Differential Privacy** (both local and global): A dedicated module integrates local and global differential privacy. In local DP, devices perturb updates before transmission, providing per-user guarantees. In global DP, the server injects noise into the aggregated model, preserving utility while ensuring compliance with regulations such as GDPR and the EU AI Act.
- **Split Learning** for resource-efficient orchestration: Split learning allows devices with limited CPU, memory, or energy budgets to compute only the initial layers of a neural network, offloading the remainder to the server. This hybrid FL-SL approach reduces device burden while maintaining model quality
- **Knowledge Distillation** for energy efficiency: Knowledge distillation enables lightweight student models to be trained under the supervision of larger teacher models, reducing computation and communication costs without sacrificing accuracy. This makes FLaaS sustainable for mobile and IoT deployments.
- **Hierarchical FL** (helper based) for scalable aggregation: Helper-based aggregation introduces lightweight intermediate aggregators (helpers) that perform partial, regional aggregation before forwarding results upstream. This hierarchical design relieves the central bottleneck, enables horizontal scaling to larger cohorts, and reduces end-to-end latency and bandwidth consumption.

Deployment and Validation

The enhanced modules have been validated through laboratory experiments and pilot deployments. Results demonstrate High device availability, predictable scalability via helper-based, hierarchical aggregation, Low-latency training, Strong user acceptance across heterogeneous devices. With these improvements, FLaaS has evolved into a comprehensive platform for scalable, privacy-preserving, and energy-aware federated learning. By integrating TaRDIS tools, Telefónica strengthens its position in trusted AI deployment across the cloud-edge continuum.

3.3.2. AI-Driven Planning, Deployment, and Orchestration - T5.2

3.3.2.1 - PeersimGym: An environment for Task Offloading in Edge Systems

Since D3.3 the **PeersimGym (T-WP5-11)** tool **has been further developed** as an environment for task offloading in edge systems. Building on the Peersim-based simulation and the PettingZoo environment, which provides a Markov Game abstraction for agent interaction, several extensions have been introduced. These include utilities to adjust task dataset sizes across device classes, additional artificial topologies to broaden experimental settings, and a sparse event-based reward mechanism to better reflect realistic scenarios. The new reward system issues positive feedback for successful task completions or offloading and negative feedback for dropped tasks, with optional shaping using the original reward. These updates support the requirements of concurrent work in the FAuNO Standalone tool.

Funded by the European Union

3.3.2.2 - FAuNO: Federated AI Network Orchestrator

The WP5 team has implemented a prototype extension of FAuNO for use with real computer networks. The extension, FAuNO Standalone Framework, decouples the FAuNO orchestration framework from PeersimGym and other simulated environments, enabling integration with communication frameworks such as Babel. This decoupling has been achieved through the development of the FAuNO Standalone node that can process tasks, exchange information, and orchestrate workloads independently. Each node comprises three components: (i) an event-driven engine, (ii) an extensible task processing module allowing clients to load Python classes for handling application-specific tasks, and (iii) an orchestration module supporting both the continued training of federated reinforcement learning agents and information exchange. These components replicate the functionality of the simulation while enabling deployment in real systems. Integration with Babel is in progress, after which testing will begin.

At the orchestration level, FAuNO will be deployed on FAuNO Standalone nodes, supporting both continued training and the federated component. In order to make this feasible, ongoing work focuses on evaluating FAuNO with sparse reward mechanisms and experimenting with alternative reward shaping functions. Details on the FAuNO Standalone tool are available in the TaRDIS wiki.

3.3.3. Lightweight and Energy-Efficient ML Techniques - T5.3

Three techniques are included in the TaRDIS toolkit for making the inference of an ML model more lightweight in terms of energy-efficiency and inference latency. These techniques (which were only briefly outlined in Deliverable D3.1, and have been further developed since then) involve the transformation of Deep Neural Networks (DNNs) ML models, as the DNNs are one of the key contributors to the energy consumption at swarm systems. The three methods of interest are: Early-Exit (EE) of inference (Sections 3.3.3.1 and 3.3.3.2), Knowledge Distillation (KD) (Section 3.3.3.3), and Pruning (Section 3.3.3.4).

Detailed description of the functionality of these tools can be found on the TaRDIS wiki.[48]

3.3.3.1. Early-Exit tool (T-WP5-06)

This method can be implemented in a DNN that includes multiple hidden layers. Compared to the initial version of the EE tool that was described in D3.1, we have performed several modifications that are described below and also in the Early-Exit GitHub repository.[49]

The function to be exposed is imported as follows:

```
from early_exit.get_early_exit import create_networks
```

It is used as follows:

1. Define your model (optional train it on your dataset)
2. Define a class that splits your model into nn.Sequentail Sublocks that, in turn all belong to an nn.Sequential Container, named "self.net" (see demo.ipynb, SplitModel class)
3. Call the function with the following parameters:

[48] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs

[49] https://github.com/Ilias-Paralikas/early_exit

Funded by the European Union

Parameter	Type	Description
`model`	`Split Model`	The split model as defined in step 2.
`input_shape`	`Tuple`	Shape of the input data (e.g., `(1, 3, 32, 32)`).
`thresholds`	`List of floats`	Threshold values for early exits, if exceeded the exit is taken.
`neurons_in_exit_layers`	`List of lists of integers`	Number of neurons in each exit layer.
`epochs`	`Integer`	Number of epochs to train the model.
`train_dataloader`	`torch.utils.data.DataLoader`	DataLoader for training data.
`test_dataloader`	`torch.utils.data.DataLoader`	DataLoader for test data.
`optimizer`	`torch.optim`	Optimisation algorithm for training (e.g., `torch.optim.SGD`).
`optimizer_parameters`	`dict`	Parameters to configure the optimizer (e.g., `{'lr': 0.001}`).
`criterion`	`torch.nn`	Loss function to train the model .(e.g.,`torch.nn.CrossEntropyLoss()`).
`training_method`	`String` or `Integer`	Method of training (`'whole_network'`, `'all_exits'`, or int).

NOTE on training method.

- The `whole_network` option will train the whole network as well as the exits

Funded by the European Union

- The `all_exits` option will train ONLY the added exits (works well for pretrained networks that we don't want to degrade the performance of the body)
- If an integer is provided as a parameter, it will train the specified exit. Obviously, the number must not exceed the number of exits.

Returns:

Parameter	Type	Description
`networks`	`List of EarlyExitNetworkSegmentor instances`	A list of the separate parts of the network.
`train_losses`	`List`	A list of the training loss.
`test_accuracies`	`List of strings`	The accuracies after training.

3.3.3.2. D-Exit tool (part of T-WP5-06)

The D-exit tool (its operation is also described in its GitHub repository[50]) is a distributed inference system that implements EE strategies across multiple nodes. It allows for efficient inference by potentially terminating the process early at different stages of the network, distributed across edge and cloud devices. The Dexit architecture is designed to be flexible, scalable, and efficient. It comprises several key components that work in concert to enable distributed early exit inference.

```
[Edge Device] <---> [Cloud1 Node] <---> [Cloud2 Node]
       ^                ^                ^
       |                |                |
       v                v                v
[P2P Network Layer (libp2p)]
       ^
       |
       v
[Network State Management]
```

The core components of the D-exit tool are the following:

1. **Edge Device**: The Edge Device serves as the entry point for inference requests. It is typically a resource-constrained device (e.g., smartphone, IoT sensor) that initiates the inference process. The key responsibilities of the device are: (i) Initial processing of input data; (ii) Making early exit decisions based on confidence thresholds; (iii) Forwarding complex cases to Cloud1 Node.

[50] https://github.com/anaskalt/dexit?tab=readme-ov-file

2. **Cloud1 Node**: The Cloud1 Node acts as an intermediate processing unit with more computational power than the Edge Device. The key responsibilities of the Cloud1 Node are: (i) Processing more complex inference tasks; (ii) Implementing its own early exit strategy; (iii) Forwarding the most challenging cases to Cloud2 Node.
3. **Cloud2 Node**: The Cloud2 Node represents the final and most powerful computational resource in the DEXIT network, while its responsibilities include: (i) Handling the most complex inference tasks; (ii) Providing high-accuracy results for challenging inputs; (iii) Supporting the overall network by processing overflow from other nodes.
4. **P2P Network Layer (libp2p)**: The P2P Network Layer, implemented using libp2p, forms the communication backbone of DEXIT, offering: (i) Decentralised peer discovery; (ii) Efficient message routing between nodes; (iii) Support for various transport protocols; (iv) NAT traversal capabilities.
5. **Network State Management**: The Network State Management component keeps track of the overall system state, including peer statuses, inference requests, and results. Its key responsibilities include: (i) Maintaining a real-time view of the network topology; (ii) Tracking the status of ongoing inference tasks; (iii) Managing the distribution of workload across nodes.

Moreover, the key software component of the D-exit tool are:

1. **P2PHandler (network/handler.py)**: The P2PHandler is responsible for managing all P2P network operations within DEXIT. Its key functionalities include: (i) Network initialization and peer discovery; (ii) Message publishing and subscription; (iii) Direct messaging between peers; (iv) Handling of inference requests and results.
2. **NetworkState (utils/state.py)**: The NetworkState class manages the overall state of the DEXIT network, while enabling: (i) Tracking peer statuses; (ii) Managing inference requests and results; (iii) Providing network state summaries.
3. **CIFARDataLoader (data/dataloaders.py)**: The CIFARDataLoader handles data loading and preprocessing for the CIFAR10 dataset, which is used for testing and demonstration purposes in DEXIT. Its functionalities include: (i) Loading and preprocessing CIFAR10 dataset; (ii) Creating DataLoaders for efficient batch processing; (iii) Supporting customizable sample sizes for testing.
4. **Early Exit Models (early_exit/)**: The early exit models are custom neural network architectures that support multiple exit points for inference. Their key features include: (i) Multiple intermediate classifiers (exit points); (ii) Confidence thresholds for early termination; (ii) Adaptive computation based on input complexity.

3.3.3.3. Knowledge Distillation (T-WP5-07)

The KD method targets to transform a large DNN model (with multiple hidden layers) to a smaller one that is more compact, more energy and computationally-efficient, without losing significant accuracy in the DNN output/prediction. Compared to the initial version of the EE tool that was described in D3.1, we have modified the KD functionality that is described in its GitHub repository.[51]

The function to be exposed is imported as follows

```
from knowledge_distilation import knowledge_distillation_train
```

and the input parameters are described in the following table.

[51] https://github.com/Ilias-Paralikas/Knowldedge_Distillation

Parameter	Type	Description
`teacher_model`	`torch.nn.Module`	The pre-trained teacher model used for knowledge distillation.
`student_model`	`torch.nn.Module`	The student model that will learn from the teacher model.
`n_epochs`	`int`	The number of epochs to train the student model.
`trainloader`	`torch.utils.data.DataLoader`	The DataLoader providing the training data.
`criterion`	`torch.nn`	The loss function used to compute the loss.
`optimizer`	`torch.optim`	The optimizer class used to update the model parameters (e.g., `torch.optim.Adam`).
`optimizer_params`	`dict`	A dictionary of hyperparameters for the optimizer (e.g., `{'lr': 0.001}`).
`teacher_percentage`	`float`	The percentage of teacher model's output to be used in the loss calculation. Default is 0.5.
`temperature`	`float`	The temperature parameter for softening the logits. Default is 2.

The Returns of the KD tool are:

Parameter	Type	Description
`student_model`	`torch.nn.Module`	The trained student model.
`training_losses`	`List`	A list with the training losses per epoch.

3.3.3.4. Pruning (T-WP5-08)

The pruning tool transforms a DNN in a more lightweight version by nullifying the neuron connections that have a negligible impact on the DNN performance. To this end, the pruning functionality streamlines the inference process, in terms of latency and conservation of energy

Funded by the European Union

and computational resources. The updated description can be found in the tool's GitHub repository.[52]

The function to be exposed is import as follows:

```
from pruning import prune_model
```

and the following parameters are used as input:

Parameter	Type	Description
`model`	`torch.nn.Module`	The PyTorch model to be pruned.
`sparse_ratio`	`float`	The ratio of sparsity to be applied to the model.
`input_shape`	`tuple`	The shape of the input tensor that the model expects.
`pruned_layer_types`	`list`	A list of layer types to be considered for pruning (default: `['Linear', 'Conv2d', 'Conv3d', 'BatchNorm2d']`).
`exclude_layer_names`	`list` or `None`	A list of layer names to be excluded from pruning (default: `None` will automatically detect the names, but it could possibly cause an error).
`prunner_choice`	`str` or `None`	The choice of pruner to be used (default: `None`, will select `L1NormPruner`).

The return from the Pruning tool is a wrapper function for the nni pruning method, including the updated ML model.

Parameter	Type	Description
`model`	`torch.nn.Module`	The pruned PyTorch model.

Note that when loading the model, the relative path to the model definition has to be the same as when the model was first created.

3.4. Data Management and Distribution Primitives

This section summarises the status of the APIs related to communication membership and monitoring/reconfiguration. The structure of the following subsections is based on the TaRDIS WP6 project tasks (matching the structure of Section 3.4 of Deliverable 3.3):

- Task 6.1 - Decentralised Membership and Communication APIs (Section 3.4.1)

[52] https://github.com/Ilias-Paralikas/Pruning

- Task 6.2 - Decentralised Data Management and Replication APIs (Section 3.4.2)
- Task 6.3 - Decentralised Monitoring and Reconfiguration APIs (Section 3.4.3)

3.4.1. Decentralised Membership and Communication APIs - T6.1

The TaRDIS toolbox will provide several types of distributed (and decentralised) protocols that provide different abstractions (potentially with different guarantees) for TaRDIS applications. The distributed abstractions that we consider in TaRDIS are the following:

a) Overlay Networks (Section 3.4.1.1) - superseded by Membership Abstraction APIs (Section 3.4.1.2) which define and maintain a logical network interconnecting the different components of a TaRDIS application, and that self-manages in case of changes on the system affiliation (components joining, leaving, or failing) and potentially to other dynamic aspects of the environment (e.g., reliability of a network link) or system (e.g., variations in the workload).
b) Communication Primitives (Section 3.4.1.3), which operate on top of overlay networks, provide the fundamental mechanisms that allow different components of a TaRDIS application to interact, exchange information, and coordinate.
c) Communication APIs for managed swarm elements (Section 3.4.1.4), which are dedicated to applications designed and developed using swarm protocol specifications (Section 2.2) or DCR Graphs (Section 2.3).

Concrete implementations of these abstractions are provided as part of the TaRDIS toolbox. In the following, we discuss relevant updates (when applicable) to the APIs of these abstractions.

3.4.1.1. Overlay Network Specifications and APIs (T-WP6-01)

This tool aimed at providing a general purpose API for selecting different overlay networks to support the operation of swarm applications based on the functionalities required by the application, avoiding the programmer to be required to know the details about the implementation and operation of the individual overlay networks. As reported previously, practical usage of this tool revealed that it was too cumbersome to use, being intrusive for developers. Due to that we abandoned it and no further development or evolution happened in this context.

3.4.1.2. Membership Abstractions and APIs

TaRDIS has developed several different decentralised membership abstractions that can support a wide range of swarm application functionalities and scenarios. This includes several variants of the HyParView protocol[53] that differentiate between them according to the functionality that they provide. We have evolved the original protocol to support the self-discovery mechanisms of Babel-Swarm, that allows a process relying on this version of HyParView to leverage on the self-discovery mechanisms of Babel to locate a contact node already in the network to introduce the new node to the swarm with no need of human intervention. Two other versions of this protocol were developed that respectively support security mechanisms introduced in Babel-Swarm (authentication and secure communication channels) and autonomic management (allowing runtime parameters to be switched at runtime to better cope with operational conditions of the swarm).

[53] J. Leitao, J. Pereira and L. Rodrigues, "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast," 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07), Edinburgh, UK, 2007, pp. 419-429, doi: 10.1109/DSN.2007.56.

Funded by the European Union

We also have an implementation of the X-BOT protocol[54] [55] that was also evolved to take advantage of the self-discovery mechanisms of Babel-Swarm. This protocol allows to improve the random membership of the service over time taking into account an optimisation function (in our implementation latency between swarm elements).

We have implemented a Global Membership service that allows elements in the swarm to obtain a best-effort global view of the system membership, which while not being useful to support the operation of critical components in swarm systems, can be useful for operators interacting and managing the system.

We also developed Geonesia, a membership protocol which introduces a hierarchical clustering approach tailored for community-oriented scenarios, motivated by the EDP use case. Unlike purely random or opportunistic membership services, Geonesia organizes swarm elements into clusters based on locality, by maintaining a hierarchical structure across these clusters. This allows the system to balance **l**ocal efficiency (fast interactions within clusters) with global scalability (coordinated communication across the hierarchy).

Finally, we are developing a gossip opportunistic membership service tailored for highly dynamic and resource-constrained environments such as satellite networks, particularly motivated by the GMV use case. This service departs from traditional membership abstractions by adopting an opportunistic gossip-based approach to disseminate membership information (and optionally other control information), being well-suited to scenarios where connectivity is limited and intermittent, and communication windows between nodes are unpredictable.

All of the membership abstractions discussed above currently rely on a common API defined in the context of Babel, named **Babel Protocol Commons**,[56] that fundamentally rely on the events referenced in the table below. This API has remained unchanged from the early stages of the project as they were designed to be simple, yet expressive enough to allow interaction will all developed solutions.

Request: GetNeighborsSampleRequest	Request to get a sample of neighbors (Host format) from the Active view up to a number (provided in the event).
Reply: GetNeighborsSampleReply	Generated in response to the previous request.
Notification: NeighborUp	Indicates the Host of a local neighbor that became available.
Notification: *NeighborDown*	Indicates the Host of a local neighbor that is no longer available.

[54] J. Leitão, J. P. Marques, J. Pereira and L. Rodrigues, "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlay Networks," in IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 11, pp. 2175-2188, Nov. 2012, doi: 10.1109/TPDS.2012.29.

[55] J. C. A. Leitao, J. P. d. S. F. M. Marques, J. O. R. N. Pereira and L. E. T. Rodrigues, "X-BOT: A Protocol for Resilient Optimization of Unstructured Overlays," 2009 28th IEEE International Symposium on Reliable Distributed Systems, Niagara Falls, NY, USA, 2009, pp. 236-245, doi: 10.1109/SRDS.2009.20.

[56] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-protocolcommons

Funded by the European Union

3.4.1.3. Communication Abstractions and APIs

TaRDIS provides multiple communication abstractions in the context of Babel-Swarm (and also Babel-Android) that provide a point-to-multipoint communication model, and operate on top of membership abstractions discussed above. In particular we provide several broadcast primitives with different semantics, including a flood broadcast primitive, several alternatives of an eager gossip broadcast primitive (that provide different guarantees provided to the programmer in terms of self-configuration, self-management, and security — similarly to the different variants of the the HyParView protocol discussed above), a one-hop broadcast primitive, and complementary to these, two variants of an Anti-Entropy mechanism.

We are currently finishing the development of a solution for streaming data relying on an API based on publish-subscribe.

In parallel, we are developing Micro-Babel, a lightweight variant of Babel designed to run on small embedded systems such as microcontrollers, where memory, processing power, and energy are scarce. Micro-Babel also provides a reduced but efficient set of communication abstractions, carefully adapted to the limitations of embedded and resource-constrained devices. While it does not replicate the full spectrum of broadcast and gossip primitives available in Babel-Swarm, Micro-Babel retains essential mechanisms for reliable point-to-point and lightweight gossip-based dissemination, ensuring interoperability with the broader TaRDIS communication layer. These abstractions are designed to minimize memory and bandwidth usage, while still enabling embedded nodes to actively participate in swarm coordination. This makes it possible, for instance, for microcontroller-based devices to contribute sensor data, receive control commands, or engage in collaborative tasks alongside more powerful swarm peers, thereby reinforcing the heterogeneity and scalability of the TaRDIS ecosystem. The development of these primitives in Micro-Babel has entered their final development stage.

The primitives discussed above rely on a common API also defined in Babel Protocol Commons that include the events listed below for self-containment. It should be noted that the API exposed by these components have mostly remained stable from Deliverable 3.2.

Request: BroadcastRequest	Requests the broadcast of some information for a particular application or protocol.
Notification: BroadcastDelivery	Notifies of the reception of data to a given application or protocol from a broadcast communication primitive.
Notification: OneHopBroadcastDelivery	Similar to the previous one but reserved to be used by one hop broadcast solutions (the motivation for this event in the API is to simplify the co-existence of a regular broadcast protocol and a one-hop broadcast in the same application).
Notification: IdentifiableMessageNotification	This is an event used by a broadcast protocol to notify an anti-entropy protocol of a new message received that should be synchronized over time with other (and potential new) neighbors in the swarm.

Funded by the European Union

Request: MissingIdentifiableMessageRequest	This event is used by an anti-entropy protocol to request a broadcast protocol to transmit a message that has been identified as missing by some swarm neighbor.

3.4.1.4. Communication APIs for Managed Swarm Elements

If a software developer chooses to adopt one of the managed approaches to the development of TaRDIS swarm applications, then they are given access to higher-level communication APIs that mostly hide the details of the underlying message exchanges. Therefore, the communication APIs are “embedded” in the correct-by-construction programming facilities illustrated in Section 3.2.1.1.

3.4.2. Decentralised Data Management and Replication APIs - T6.2

TaRDIS has developed several storage solutions that can support different aspects of the operation of swarm systems. In general all these solutions expose similar APIs, based on the ones reported in Deliverable 3.1 (and materialized in Babel Protocol Commons[57] under the *storage* package), although to support some of their functionality we have allowed these APIs to evolve as required, and will enrich the adaptors (discussed below in Section 2.4.2.4) to simplify the integration of these solutions into the programming ecosystem of TaRDIS.

3.4.2.1 PotionDB

PotionDB was designed with partial geo-replication in mind. Thus, we assume PotionDB instances to be spread at different locations across the globe. Each location only replicates a subset of the whole data. The system administrator has control over where each object is replicated. This allows accounting for data locality to ensure fast access to data, while keeping replication and storage costs controlled. Objects without locality on their access pattern can be replicated everywhere if desired.

The development of PotionDB has been concluded, and as such its APIs have not changed since the last reporting. For self-containment we report those APIs below.

begin(clk) -> txid	Begins a transaction - clk, if provided, is used to set causal dependencies; returns the transaction identifier.
commit(txid) -> clk	Commit transaction with identifier txid; returns clk to be passed in begin for setting up causal dependencies - if null, transaction was rolled back.
rollback(txid)	Rolls back transaction with identified txid.
get(txid, id) -> value	Gets the value of object id in the context of transaction txid.
read(txid, id, op) -> value	Executes read-only operation op in object id in the context of transaction txid, returning the result value.

[57] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-protocolcommons

upsert(txid, id, op) -> ok	Executes update operation op in object id in the context of transaction txid; return ok if the operation succeeded.

PotionDB also provides a data definition API that allows to create and delete buckets, and to create views. Even if buckets have no associated data type, i.e., any object type can be stored in any bucket, we expect that applications store objects of the same type in each bucket. A document or a table row can be stored in PotionDB as a map CRDT, with each element of the map having its own type.

```
CREATE VIEW (MonthlyTopSales, views) WITH
YEAR == ANY (SELECT DISTINCT SaleDate.Year FROM sales),
MONTH == ANY (SELECT DISTINCT SaleDate.Month FROM sales) AS
SELECT ProdName, SaleDate.Month, SaleDate.Year, SUM(Price -
Cost) AS SumProfit, COUNT(*) AS NumSales
FROM Sales
WHERE SaleDate.Month == [MONTH] AND SaleDate.Year == [YEAR]
GROUP BY ProdName
ORDER BY SumProfit DESC, NumSales DESC
LIMIT 10
```

The create view receives a view specification defined in a language based on SQL. The example aboveshows the specification of a view that maintains the top 10 products with most profit in sales for each month. CREATE VIEW specifies the key prefix and bucket of the materialized view objects. The FROM specifies which container(s) are used in the computation of the view, while SELECT specifies the attributes of the view, which can include simple attributes or aggregations. It is possible to define aggregations over groups with the GROUP BY clause, restrict the number of elements with a LIMIT clause, and order the results using an ORDER BY clause.

In recurrent queries, it is common that a generic query is instantiated with different values. To support this, our language allows the use of variables in the view definition. In the example, variables YEAR and MONTH lead the system to maintain for each pair (year, month), the top 10 sales.

3.4.2.2 Arboreal

The Arboreal[58] data management system is a key-value store replicated storage solution designed to operate across cloud and edge infrastructures, primarily targeting large-scale stateful swarm applications with elements scattered throughout large geographical areas. Its unique design overcomes the limitations of traditional data replication for edge computing, where data typically resides in centralized data centers, leaving edge nodes as mere caches. Instead, Arboreal allows full read and write access at edge nodes, supporting low-latency interactions directly at the edge. By doing so, it enables applications requiring real-time data manipulation, such as augmented reality, autonomous vehicles, and live analytics, to perform efficiently without the latency of communicating back to a central data repository.

[58] P. Fouto, N. Preguiça and J. Leitão, "Large-Scale Causal Data Replication for Stateful Edge Applications," *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, Jersey City, NJ, USA, 2024, pp. 209-220, doi: 10.1109/ICDCS60910.2024.00028.

Funded by the European Union

Arboreal development has been concluded and no significant evolution to its API have happened..

Arboreal exposed the following operations for clients (with respective replies):

Request: ReadOperation	A read operation is emitted to a server (edge or cloud) identifying the object to be read using a unique key by a swarm application.
Reply: ReadOperationReply	This constitutes the reply to the previous operation, that both exposes the value read and provides to the client a logical time stamp (using an Hybrid clock).
Request: WriteOperation	A write operation is emitted to a server (edge or cloud) providing the key that identifies the object to be written, and also an (optional) parameter that states how many replicas in Arboreal should execute the operation before the operation completes.
Reply: WriteOperationReply	This constitutes the reply to the previous operation, indicating that the write operation completed (given the optional durability parameter) and provides to the client a logical time stamp (using an Hybrid clock) identifying the logical time that was associated to this operation.
Request: ClientMigration	This operation is issued by a swarm application when it switches the server (edge or cloud) that it was using previously to interact with this system. This operation blocks until the client is free to interact with this replica with guarantees that causal+ consistency will not be violated.
Reply: ClientMigrationReply	This operation constitutes the indication that the previous operation has completed.

More details of its client architecture and evaluation is provided under Arboreal-Others[59].
A reference implementation can be found on the Arboreal Git repository[60].

3.4.2.3 Nimbus

Overview

Nimbus, briefly mentioned in D6.2 as, was not fully detailed as essential features were still being finalized; its development has now been completed, though small adjustments and

[59] https://codelab.fct.unl.pt/di/research/tardis/wp6/public/arboreal-others/arboreal-client
[60] https://codelab.fct.unl.pt/di/research/tardis/wp6/public/arboreal

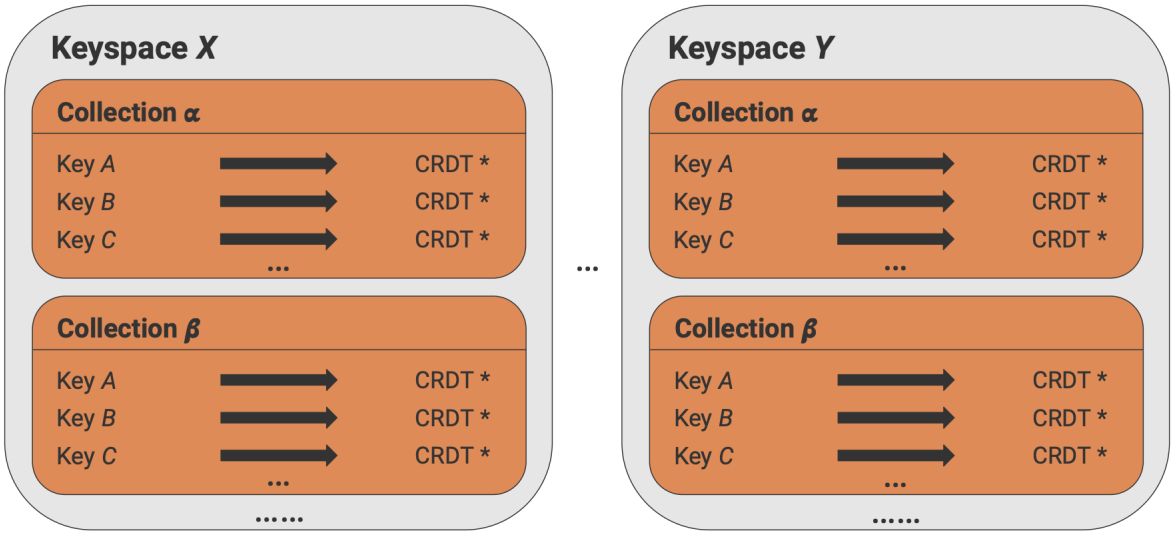
optimisations are still ongoing. Designed as a fully decentralised storage system, Nimbus provides scalable and efficient data storage without relying on a central authority. To achieve this, Nimbus doesn't require any kind of dedicated infrastructure (i.e., cloud or edge nodes) such that each node in the system acts as an independent replica and acts as part of the replicated storage system. This aims to target decentralised applications (such as swarms) where replicas may leave or enter the network at any moment and execute operations in any order, without disrupting the correct functioning of the system as a whole. This way, different applications, such as satellite swarms, telemetry swarm systems in harsh environments, to name a few, can interact directly with each other without relying on an external infrastructure to store and synchronize their data. Moreover, since swarm nodes may be heavily resource-restricted, having every node in the system fully replicate every object becomes too expensive. So, Nimbus offers a partial replication mechanism among its different information units. This way, each node can specify or request which information it will replicate, without leading to an overflow state in each peer in the system.

Nimbus supports access control by allowing the creators of information units to set up access policies and add or remove access in real time. Additionally, it is possible to reconfigure information units on demand to better satisfy swarm needs. While objects are partially replicated across different nodes, a metadata control unit is shared and propagated to all nodes so that every node has the information about which information units were already created, as well as the access policies for each one and other relevant control information.

Data Model

Nimbus provides strong eventual consistency — a consistency model that guarantees all nodes will reach the same state after receiving all updates, regardless of the order in which updates were applied. This is accomplished using CRDTs (Conflict-Free Replicated Data Types).

Nimbus offers a key-value store interface, by having a data model constituted as *keySpaces* and dividing each keySpace into separate *collections*. A collection is represented as a dictionary of key-value pairs, where a key acts as an identifier of an object, and the value is represented as a CRDT (e.g., a counter, set). This offers a rich interface to the developer, by allowing him to choose the data type in which he wishes to encode its data, as well as offering composite types, such as maps, to allow recursive and composite data structures by each application needs (i.e., a tree-like structure of attributes) as depicted below:



Funded by the European Union

Replication Model

Nimbus divides its internal state into two categories: metadata (i.e., control information) and storage data (i.e., CRDT content). Nimbus uses a more aggressive approach for metadata synchronization to ensure every node in the system possesses and can access this information. On the other hand, storage data is partially replicated and only propagated between replicas of the respective information unit.
It supports partial replication through its keyspaces and collections (i.e., information units). Since relying on an autonomic partial replication mechanism can be expensive and complex due to the highly dynamic nature of such systems (e.g., it's difficult to predict when a peer may enter or leave the system), we opted for a mechanism where creators of information units can explicitly modify the replica list of each unit.

Interface

Nimbus client API follows Babel Protocol Commons,[61] a set of common APIs developed in TaRDIS for interacting with distributed systems and their protocols. To achieve this, we provide an event-based architecture for client interaction. When a client wishes to execute an operation, it creates a request with the desired operation and, upon completion, receives a response containing the requested information and any additional details if something went wrong (e.g., lack of permissions, object not found, etc.).

More details of the its architecture is provided in Babel Protocol Commons[62], under the *storage* package, as well as on the Nimbus wiki[63].
An under-development implementation can be found on the Nimbus Git repository.[64]

3.4.2.4 Integrated Storage Solutions

Integrating third-party storage solutions into a system presents numerous challenges, particularly when it comes to aligning external tools with specific project requirements, while providing compatibility and interoperability.

In order to mitigate some of these challenges, as well as offering TaRDIS developers the flexibility of integrating different storage solutions into their projects, the TaRDIS toolbox offers a set of adapters that integrate external storage solutions within the Babel framework. The adapters are built on top of Babel Protocol Commons, a set of common support classes for distributed protocols, that propose common APIs for handling, managing and interacting with these types of protocols and the systems that use them.

Developers can use these storage solutions by placing the desired adapters in their protocol stack and manage their state by leveraging the common APIs offered by the TaRDIS toolbox.

These integrations have been completed, and there is no additional information to be reported since the last deliverable.

3.4.3. Decentralised Monitoring and Reconfiguration APIs - T6.3

The TaRDIS toolbox will provide primitives to support reconfiguration of the applications components in the runtime, as well as acquisition of the decentralised telemetry information from the deployed system, including information about the load and health conditions of

[61] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-protocolcommons
[62] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-protocolcommons
[63] https://codelab.fct.unl.pt/di/research/tardis/wp6/internal-tools/nimbus/nimbus/-/wikis/home
[64] https://codelab.fct.unl.pt/di/research/tardis/wp6/internal-tools/nimbus

Funded by the European Union

different components. Such primitives are part of the tools **T-WP6-08** (Namespaces management) and **T-WP6-09/10** (Telemetry). Our approach for Distributed Management of Configuration based on Namespaces is designed with scalability and fault tolerance in mind.

3.4.3.1 Distributed Management of Configuration based on Namespaces and Telemetry

Acquisition for Containers

The TaRDIS toolbox will provide primitives to export fine-grained stored metrics to other parts of the toolbox, such as machine learning components that require time series of the stored metrics over some period of time. Provided primitives export fine-grained stored metrics using the standard *OpenMetrics* format.[65] This part of the toolbox leverages the cloud-edge continuum, but it relies on open-source solutions such as *Prometheus, Docker, NodeExporter, Grafana* as part of its core. As part of metrics collection, the system collects metrics from few places: (i) nodes, (ii) applications running in containers, and (iii) other parts of the toolbox, where TaRDIS will provide APIs to integrate their specifics such as distributed protocols and probing metrics.

Metrics aggregation is done in two ways: **(i) centralised**, which is more suitable for small swarms, where every swarm node sends its metrics to the control plane which then proceeds with the aggregation. With this strategy, the aggregation is done on the control plane side.This process will converge faster, but it will introduce more load towards the control plane. **(ii) decentralised**, which is more suitable for large swarms, where metrics are aggregated inside the swarm itself using the peer to peer protocols (*HyPerView*[66] and *PlumTree*[67]). With this approach, swarm nodes form a tree in which aggregation is done by propagating partially aggregated results to the higher levels of the tree, until they reach the root. When the root acquires a global view of the swarm metrics at the end of the aggregation cycle, it will send it to the control plane. Moreover, aggregated metrics will also be broadcasted to swarm members, so they can make local decisions based on a global state. This process is more network heavy towards swarm nodes, and it will converge slower, but processing is done locally without burdening the control plane which improves the scalability of the system.

Tree-based distributed aggregation provides more accurate results for continuously changing values, as expected in metric collection, but it is sensitive to dynamic networks with node or link failures. *PlumTree*'s self-healing properties repair the tree, yet root selection remains a challenge since multiple nodes may compete to become root. To address this, nodes are ranked by the aggregation mechanism, and only the highest-ranked node should promote itself. Because nodes may join or leave between rounds, they gossip their scores with neighbors to improve their decision. Temporary situations with multiple roots can still occur, but the system converges as lower-ranked nodes receive cancel responses and give up the root role.When the root fails, nodes rely on the rank list, neighborhood scores, and a maximum delay since the last aggregation to decide when to promote themselves. Only the highest-ranked node in a neighborhood steps up, minimizing false promotions and reducing the time to converge to a single stable tree.

These monitoring and reconfiguration APIs are still under development and are not yet documented; their documentation will be finalised as part of the final TaRDIS toolkit by the end of the project. Some of the already-completed APIs include:

[65] https://asc.di.fct.unl.pt/~jleitao/pdf/dsn07-leitao.pdf
[66] https://asc.di.fct.unl.pt/~jleitao/pdf/srds07-leitao.pdf
[67] https://openmetrics.io/

- Decentralised monitoring:

```
getMetrics(id) → timeseries
getNodeMetrics(id) → timeseries
getApplicationMetrics(nodeId, namespaceId, appId) → timeseries
exposeMetrics(timestamp) → timeseries
exposeMetrics(start, end) → timeseries
customMetrics(data) → status
retentionPeriod(time, metric) → status
healthcheck(topic) → channel
```

- Reconfiguration management:

```
createNS(cId, labels, resList) → nsaid
deleteNS(id) → status
readNS(id) → nsDetails
createChildNS(nsId, labels, resList) → nsId
createConfigSchema(labels, version, schema) → sId
deleteConfigSchema(sId) → sId
getConfigSchema(sId) → schema
getConfigSchemaTimeline(sId) → timeline
createConfig(labels, configList) → status
disseminateConfig(labels, configList, percentage) → status
getConfigTimeline(config) → timeline
```

Full documentation about the available APIs function is available in the project repository, including reconfiguration[68] and metrics.[69] One example of the APIs usage, can be found within a developed command line interface (CLI) tool.[70] The same CLI operations are also available through developed IDE.

Telemetry Acquisition for Babel Applications

To simplify the collection of (performance) metrics from free-form applications, which in the context of TaRDIS are typically developed using Babel-Swarm or Babel-Android, which evolved from the Babel framework, mechanisms were integrated into the framework that empower developers to instrument their applications to monitor application/protocol specific runtime indicators, general hardware, and arbitrary metrics, while minimising the interference of these mechanisms with the code of distributed protocols and applications and ideally with minimal performance.

To achieve this, the integrated metrics collection system supports the most common metric types, such as, counters, gauges, and histograms, but also custom metric types, namely records, which provide developers with a way to log the occurrence of specific events, and statistics gauges, which extend the functionality of gauges by keeping track of additional statistics, such as the minimum, maximum, mean, and standard deviation of the recorded values.

The collected metrics must then be exported, so they can be aggregated and used to observe the behaviour of the system. These can be exported using a multitude of methods, such as exposing it through HTTP endpoints, exporting it to local log files, or publishing them to a Message Queue to be processed.

[68] https://codelab.fct.unl.pt/di/research/tardis/wp6/public/configuration-management/-/tree/main/tools/docs
[69] https://codelab.fct.unl.pt/di/research/tardis/wp6/public/configuration-management/-/tree/main/protostar
[70] https://codelab.fct.unl.pt/di/research/tardis/wp6/public/configuration-management/-/blob/main/README.md

Funded by the European Union

Since exporting can take place using several different methods, the system must support different formats to structure the metrics to be exported. These should include widely used formats, such as the standard OpenMetrics Format, enabling developers to use this system with existing monitoring tools which they may already be using, but also allow developers to specify their own formatting.

While aggregation can be performed by other components of the TaRDIS toolbox, such as the one described in the previous section, the metrics collection system integrated in the Babel framework allows nodes to share collected metrics information amongst themselves using various approaches: a simple one, where nodes send their metrics to a pre-configured *supernode* responsible for aggregation; a broadcast one, where nodes share their metrics with all nodes in the system with a subset of those nodes being *supernodes* and thus responsible for aggregation; and finally a clustered approach, more suitable for large swarms, where nodes form clusters that elect a leader *supernode* to perform aggregations, the leader of each cluster will then send the results of those aggregations to be joined by a small set of pre-configured nodes, helping to provide insight into the inner workings of the swarm.

Aggregations can be used to simply join samples of the same metric from different nodes, or to apply more complex operations, which generate new metrics from the collected samples. These are defined by developers using an interface provided by the system, which allows them to specify the metrics to be considered for the aggregation, the aggregation function to be applied, and any other parameters that may be relevant for the specific aggregation. The results of an aggregation can then be exported using the aforementioned methods and formats so they can be used by other

Full documentation about the Babel's metrics collection system components and API is made available in the Babel Swarm repository[71], and a usage example is provided in a dedicated repository[72].

[71] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-core-swarm
[72] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-metrics-example

Funded by the European Union

4 CHALLENGES AND PLANNED WORK

This section reprises the three challenges in the specification and development of the TaRDIS toolbox programming models and APIs that were mentioned at the end of Deliverable D3.3: cross-language interoperability (Section 4.1) support for device capabilities (Section 4.2), and toolbox cohesiveness (Section 4.3). It also discusses how we are tackling a new challenge, i.e., the development of the TaRDIS developer approach (Section 4.4).

4.1. CROSS-LANGUAGE INTEROPERABILITY

The TaRDIS toolkit aims at being language-independent and offering APIs that can be leveraged by multiple programming languages — but the APIs and facilities provided by WP4, WP5, and WP6 were mostly prototyped and developed using the most suitable programming languages, and prioritising the TaRDIS use cases they are mainly being used in.

As mentioned at the end of D3.3., achieving general cross-language interoperability involves a considerable effort. Since D3.3, we have developed some examples of cross-language translation layers prioritising the interoperability of TaRDIS toolkit components — specifically, enabling:

1. the development of swarm applications that combine PTB-FLA (Section 2.4) and Babel (Section 2.5), and
2. the monitoring of applications based on swarm protocols (Section 2.2) using any language that supports the widespread Protobuf standard.[73] We have developed a demonstrator that integrates swarm protocols with tooling based on the Java Virtual Machine — in particular, Babel (Section 2.2) and fair join pattern matching (Section 3.2.1.3).

These examples show that **cross-language interoperability between the TaRDIS toolkit components and external libraries and tools is indeed achievable with targeted effort**. However, It is unlikely that, before the end of the project, we will be able to develop, test, and release more general and ready-to-use cross-language interoperability tools.

4.2. SUPPORTING DEVICE CAPABILITIES FOR SWARM REDEPLOYMENT

The TaRDIS project proposal aims at offering a toolbox with programming models where *"peers and code can be (re-)deployed based on device capabilities [...] (WP4, WP5)."* Such a requirement appeared prominent in our initial assessment of the project use cases — but after further study and refinement, all use case specifications leverage swarms where the device capabilities are known in advance, and/or are not significantly constrained w.r.t. the minimum computational requirements. For this reason, the need for supporting a detailed specification of device capabilities (and thus, the capability-based redeployment of the swarm roles and functionality) has decreased, and is not central in the ongoing work on use case development and evaluation (WP7).

Since D3.3, we have been developing a demonstration showing how devices with different capabilities can be orchestrated when developing applications based on swarm protocols (Section 2.2). This is a promising proof-of-concept, but it is unlikely that this feature of the TaRDIS toolkit will be further developed.

[73] https://protobuf.dev/

4.3. Ensuring the Cohesiveness of the TaRDIS Toolbox

In D3.3 we mentioned that the TaRDIS toolbox has been experiencing an organic growth, chiefly driven by the use case requirements, and the varying expertise and technical background of the project partners. This creates the challenge of ensuring that the toolbox models and APIs will be perceived as cohesive by its prospective users.

We also mentioned that we plan to ensure toolbox cohesiveness as part of the WP3 work towards defining the *TaRDIS development approach*. This work is ongoing (but delayed, as explained in Section 4.4 below).

4.4. Finalising the TaRDIS Development Approach

The TaRDIS project proposal includes the objective of distilling and documenting a *TaRDIS developer approach* based on the experience gained in implementing and evaluating the project use cases. This was initially planned as part of this Deliverable D3.5.

This work is still ongoing (we plan to deliver it as part of D3.6) and is proceeding along the lines discussed in D3.3. We are collecting “TaRDIS use case recipes” that describe how the toolkit is used in the project use cases, and from this we are documenting how the TaRDIS usage integrates with several mainstream software development methodologies. Since toolbox cohesiveness is also an integration issue, WP3 also plans to work on this topic in collaboration with Task 7.4 (integration).

Funded by the European Union

5 CONCLUSION

This report has documented the current status in the development of the TaRDIS toolbox, with a focus on its models and APIs, documenting the progress since the previous iteration of this Deliverable (i.e., D3.3).

The outcomes of this deliverable have been made possible by the close collaboration between the project partners, who are assembling the documentation of their respective contributions on the TaRDIS wiki (which is often referenced in this document).

This Deliverable formally marks the conclusion of the TaRDIS project tasks T3.1 (models) and T3.2 (APIs) — but in practice, the TaRDIS toolkit is still being actively improved and refined as the use case implementation and evaluation proceeds (as part of WP7), and new features are added to the TaRDIS IDE (covered in D3.2, D3.4, and the upcoming D3.6). This means that, concretely, the TaRDIS programming models and APIs will continue to be improved until the very end of the project, with the release of the final version of the TaRDIS toolkit and IDE (coinciding with Deliverable D3.6).

Funded by the European Union