TaRDIS

D3.3: Second Report on Programming Model and APIs

Revision: **1.0**

Work package	WP3
Task	T3.1, T3.2
Due date	31/10/2024
Submission date	15/11/2024
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) 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), Miodrag Djukic (UNS) Sotirios Spantideas (NKUA) Roland Kuhn (ACT)
Internal Reviewers	António Ravara (NOVA) Silvia Ghilezan (UNS)
Abstract	This document reports the second revision of the programming model and APIs which will be offered by the TaRDIS toolbox. It focuses on the differences and improvements since the previous iteration (Deliverable D3.1) and documents the status of the toolbox APIs (whose components have been selected in Deliverable D7.2).
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 decentralized, heterogeneous swarm applications deployed in diverse settings.

The main contribution of this deliverable is the second revision of the TaRDIS programming models and TaRDIS toolkit APIs. This deliverable builds upon D3.1 (first version of TaRDIS models and APIs) and documents the developments and improvements since the submission of D3.1 — 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).

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 D3.1, 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
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
MPST	Multiparty Session Types
ODTS	Orbit Determination and Time Synchronization

Funded by the European Union

P2P	Peer-to-Peer
PNT	Position, Navigation and Timing
SGAM	Smart-Grid Architectural Model
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

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 TaRDIS programming model and APIs are central aspects of the project that require a clear alignment between the requirements of the use cases, and the outputs of the research and development work packages. Consequently, the definition and development of the programming model and APIs is a collaborative effort that requires a close collaboration among all project partners.

This document has been developed concurrently with Deliverable D7.2 ("Report on preliminary validation of the toolbox"), and the two deliverables complement each other: specifically, D7.2 documents how the components of the TaRDIS toolbox will be used for the implementation and evaluation of the TaRDIS use cases. Instead, this document provides an update on the status of the toolbox components, focusing on how they will 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.*

Therefore, most sections of this document consist of brief summaries outlining the status and the differences and improvements of the TaRDIS toolbox components w.r.t. the contents of D3.1; further details are made available via references to the relevant parts of the TaRDIS wiki (and other repositories, where applicable).

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.
- 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.

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

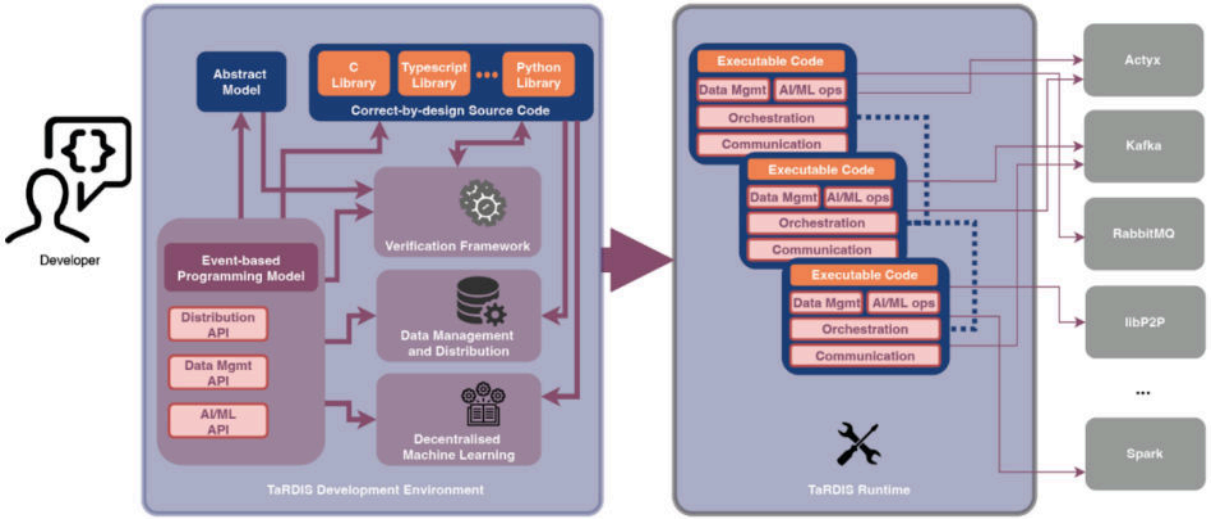
Notes:

- To ease readability, the structure of this document closely follows (where possible) the structure of D3.1: i.e., if a section number X.Y appears both here and in D3.1, then section X.Y in this document is usually intended as a status update over the corresponding section X.Y of D3.1.
- Unlike Deliverable D3.1, this document does not include a description of how the TaRDIS toolbox components will be used in the TaRDIS use cases: this information was outlined in D3.1, but it has been since revised and published in Deliverable D7.2

— and it will be further refined throughout WP7 (and reflected in the next iteration of this Deliverable).

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.

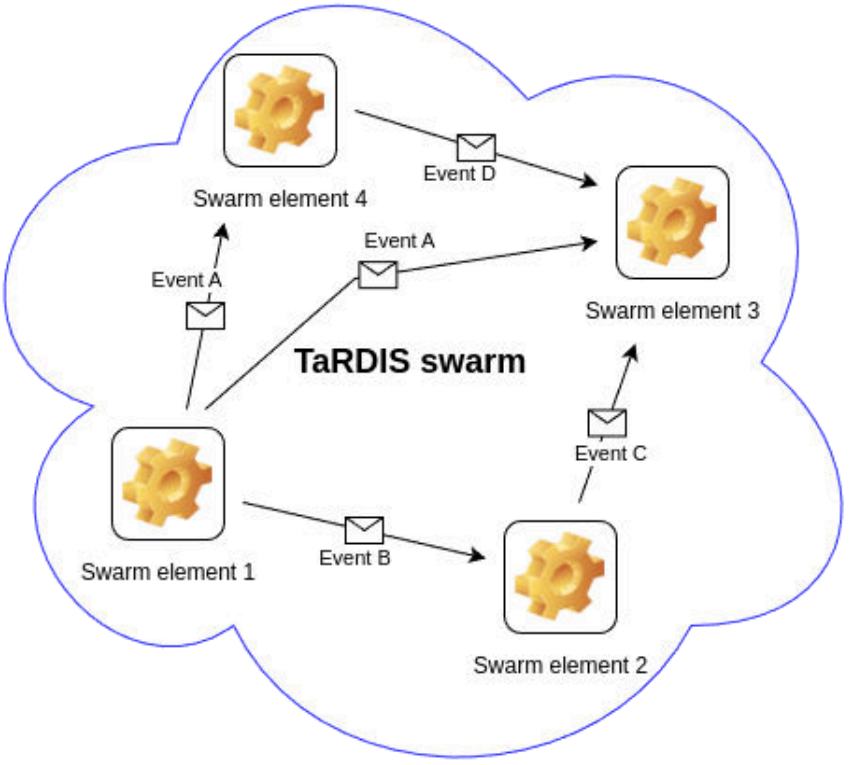
The rest of this section provides an overview of the TaRDIS approach based on managed and free-form swarm elements (Section 2.1), and then illustrates them in more detail (Sections 2.2, 2.3, 2.4).

2.1. The TaRDIS Approach: Managed vs. Free-Form Swarm Elements

This section summarises the contents of Section 2.1 in Deliverable D3.1, and introduces some revised terminology. 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 swarm elements: **managed swarm elements** and **perimeter swarm elements**.[1] The intuition is the following (and is depicted in the figure below):

[1] In D3.1 they were called "internal services" and "perimeter services," respectively. We have revised the terminology for clarity.

- 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 Deliverable D3.1, 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:

- Two complementary approaches to the design of managed swarm applications. The two 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.

- An overview of free-form TaRDIS applications based on **Babel** (Section 2.4).

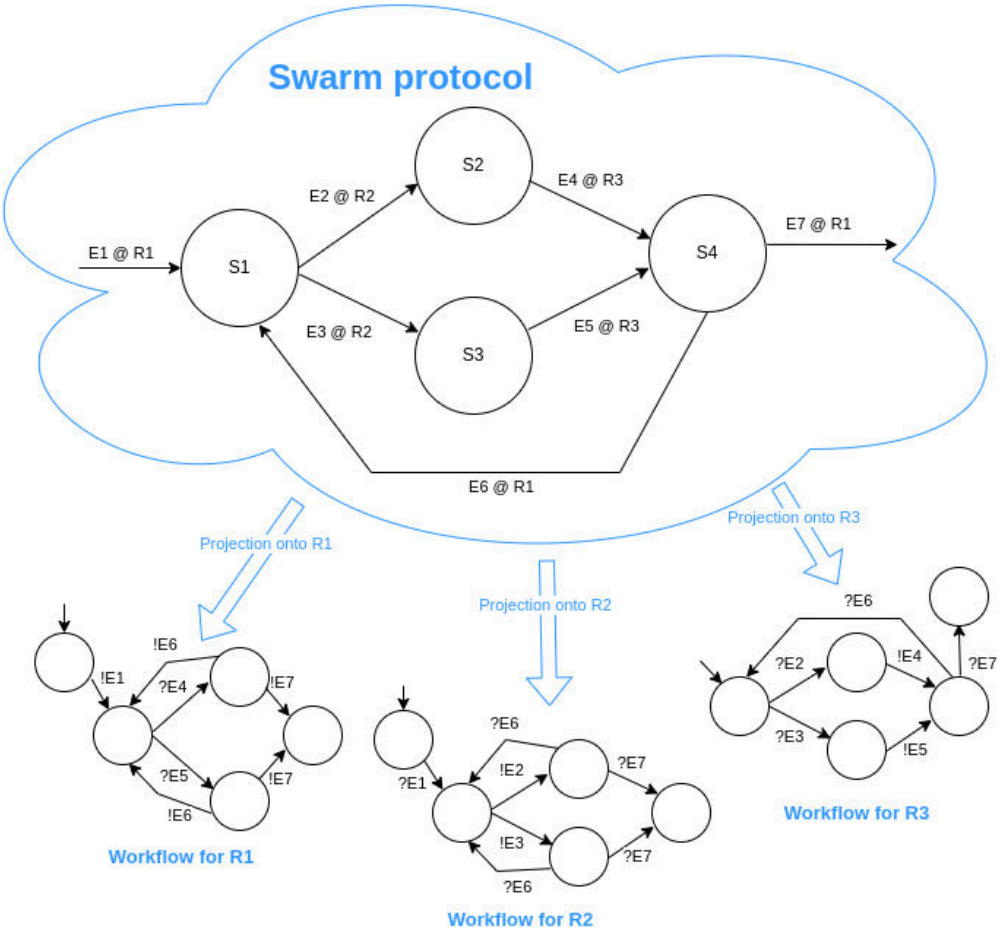
2.2. State-Oriented Managed Swarm Specifications via Swarm Protocols

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. 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:



In the figure above:

- The swarm protocol describes a "global distributed state machines" where 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 Deliverable D4.1).

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 project partner 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 Graphs

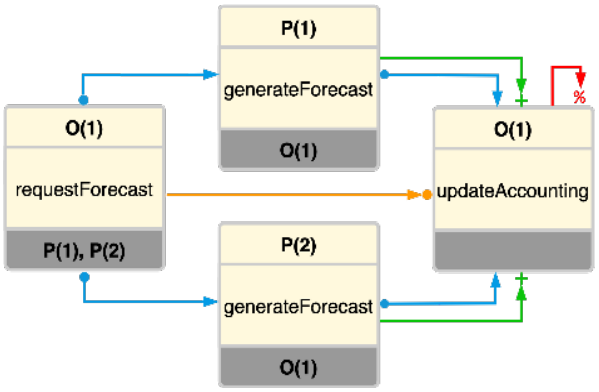
This approach to the specification of managed swarm applications was described in Deliverable 3.1 (Section 2.2.1). This section provides a summary and some revised terminology, reporting on the progress since Deliverable 3.1. Updated documentation, as well as links to additional resources, can be found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

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

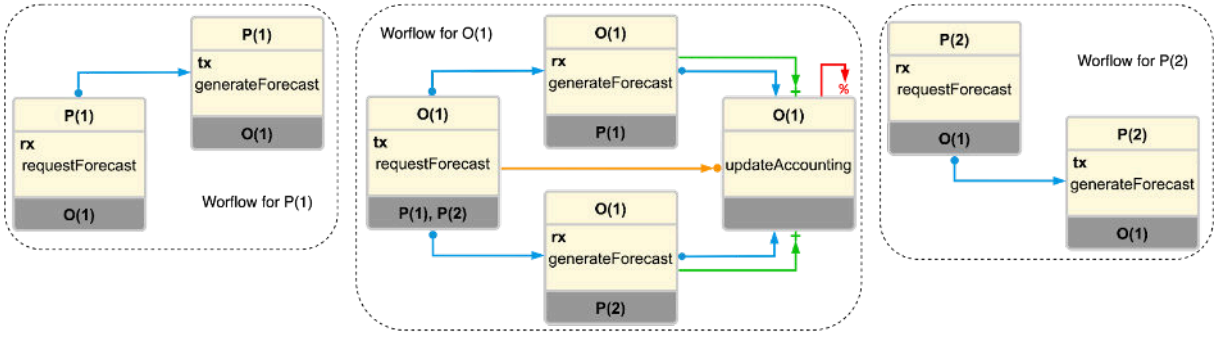


(req: requestForecast) [?] [O(1) -> P(1), P(2)]
(gen1: generateForecast) [? : Number}] [P(1) -> O(1)]
(gen2: generateForecast) [? : Number}] [P(1) -> O(1)]
(updt: updateAccounting) [gen1.value+gen2.value] [O(1)]
;
req *→ gen1
req *→ gen2
req →* updt
gen1 *→ updt
gen1 →+ updt
gen2 *→ updt
gen2 →+ updt
updt →% updt

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 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. Since Deliverable D3.1, the language has been extended to incorporate features of DCR choreographies in the style of Hildebrandt's seminal work[7] [8]. As a result, the language now 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

[7] Thomas T. Hildebrandt, Hugo A. López, Tijs Slaats: Declarative Choreographies with Time and Data. BPM (Forum) 2023: 73-89
[8] Thomas T. Hildebrandt, Tijs Slaats, Hugo A. López, Søren Debois, Marco Carbone: Declarative Choreographies and Liveness. FORTE 2019: 129-147

Babel, enabling decentralized 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.

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.[9] [10]

2.4. Free-Form TaRDIS Swarm Elements and Applications

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 that were produced in the context of WP6. The **Babel framework (T-WP6-04)** was designed (outside the scope of TaRDIS) 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 getting bogged down 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, making it 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 independently executed within the same process, with a dedicated thread for each protocol, evidently to this 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

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

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 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, that 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 6.2.

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 decentralized manner.

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 D3.1: 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 D3.1. The contents will be further refined and improved in the next (and final) iteration of this Deliverable, i.e., D3.5.

The titles of each subsection highlight which WP 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.

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, 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.

The evolution of API documentation has risen significantly, with several modern practices and tools emerging to enhance usability, maintainability and accessibility. The current key trends and best practices in API documentation embrace the use of the OpenAPI[11] specification (OAS) for documenting RESTful APIs using YAML or JSON formats. TaRDIS shall use the Swagger[12] tool to produce interactive documentation. This will foster not only generating documentation from annotations in the tools code, but also allow the users to interact with the APIs directly from the documentation, helping the developers to understand how to use the endpoints effectively.

[11] https://www.openapis.org/what-is-openapi
[12] https://swagger.io

As the APIs are not static and may evolve with time, the TaRDIS team will encompass managing multiple versions of the APIs. By using these tools, the team believes it will be able to create comprehensive and user-friendly API documentation that facilitates easier integration and a better developer experience.

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.[13]

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.[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 3.1):

- 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 D3.1, the support for the development of managed TaRDIS applications has been improved by offering 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)**.[14] [15]

[13] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Core-TaRDIS-APIs
[14] https://developer.actyx.com/
[15] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/The-TaRDIS-programming-model/Actyx-tools

- Event-oriented specification of managed TaRDIS applications via DCR graphs (outlined in Section 2.3), supported by the tool **DCR Editor (T-WP3-03)**.[16] [17]

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.

Scribble is available as open-source software on GitHub[18] 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.[19] Initial documentation for Scribble is available on the TaRDIS wiki.[20]

3.2.1.3. Join Patterns 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 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 D3.1, the JoinActors library has been extended and improved to a more complete prototypal stage, and benchmarked with various different workloads highlighting its strengths and weaknesses. The results have been published in a paper at the ECOOP 2024 conference.[21]

The JoinActors library is available as Open Source software on GitHub,[22] and has some preliminary documentation available on the TaRDIS wiki.[23]

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.

---

[16] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/The-TaRDIS-programming-model
[17] https://codelab.fct.unl.pt/di/research/tardis/wp3/TaRDIS-DCR/-/wikis/home
[18] https://github.com/nuscr/nuscr
[19] https://nuscr.dev/nuscr
[20] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/Scribble
[21] 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
[22] https://github.com/a-y-man/join-actors
[23] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/JoinActors

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**[24] **(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.

Since Deliverable D3.1, JaTyC has added support for subtyping: you may have a class with a protocol that extends another class with another protocol and the tool will ensure that the first protocol is a subtype of the second protocol. One can also create a class with protocol that extends a class without protocol. In the class without protocol, all methods are available to be called and remain so in the subclass. Then in the subclass, one can add new methods and restrict their use by only allowing them in certain states. More information can be found in an article recently published.[25]

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**[26] **(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

[24] https://github.com/jdmota/java-typestate-checker
[25] Lorenzo Bacchiani, Mario Bravetti, Marco Giunti, João Mota, and António Ravara: Behavioural Up/down Casting For Statically Typed Languages. ECOOP 2024. 5:1-5:28. https://doi.org/10.4230/LIPIcs.ECOOP.2024.5
[26] 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

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 and a first release of a public version of the tool is planned until next summer.

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[27] [28].

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.

The Ant approach as been implemented in a tool - REPL[29] 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. A version of the tool expressive enough to cover a significant subset of Java is currently under development.

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,[30] and some preliminary documentation is available on the TaRDIS wiki.[31]

[27] https://doi.org/10.1145/3555776.3577725
[28] https://arxiv.org/abs/2212.14651
[29] http://hdl.handle.net/10362/164248
[30] https://zenodo.org/records/7982416
[31] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/VeriFx

3.2.3. Specifying and Analysing Security Properties - T4.3

The tool **(Sec)ReGraDa (T-WP4-11)** and its module **IFChannel (T-WP4-08)** provide security verification in TaRDIS applications that are defined as a DCR graph. 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. Both tools are work in progress, the foundational work has to a large extent been achieved, and we are working on an implementation of the analysis.

To describe the foundational work for these tools, we need a bit more detail. 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, so the intruder is unable to distinguish them and thus unable to learn anything about the data above security level l. We have to make however a major adaptation to the standard notion of non-interference, since it trivially is 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 do not consider any obfuscation techniques like onion routing to hide the fact that messages are transmitted, as this does not seem feasible in typical swarm protocols. This restriction was already envisioned in D3.1, but we now have at least sufficient conditions that we can check.

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 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 are also considering directly integrating it into the DCR graphs and the generation of implementation used there. The implementation of this tool will commence soon.

Documentation about IFChannel, PSPSP, CryptoChoreo, (Sec)Regrada, and DCR Choreographies may be found in the TaRDIS wiki.[32] [33] [34] [35] [36]

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 relies on the correct orchestration of federated learning algorithms. These are formalised in communicating sequential processes (CSP) and verified in the Process Analysis Toolkit (PAT)[37]. After D3.1, we developed an approach to systematically construct CSP models using Python code that follows a restricted actor-based programming model. This approach should serve as a basis for developing a tool for the automatic translation of certain classes of Python code to CSP models - a work in progress.

- After D3.1, we started work to support a WP6 tool for distributed management of configurations. This tool relies on a model based on namespaces. In T4.4 we worked on this model in two directions: (i) applying the graph transformation theory to achieve accurate resource redistribution throughout namespaces, and (ii) modeling and verifying communication protocols, by applying multiparty session types and the **Scribble tool (T-WP3-02 and T-WP4-05)**.

[32] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/IFChannel
[33] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/PSPSP
[34] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/CryptoChoreo
[35] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/(Sec)Regrada
[36] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/DCR-Choreographies
[37] Ivan Prokic, Silvia Ghilezan, Simona Kasterovic, Miroslav Popovic, Marko Popovic, Ivan Kastelan: Correct Orchestration of Federated Learning Generic Algorithms: Formalisation and Verification in CSP. ECBS 2023: 274-288.

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.[38]

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), the TDM (Time Division Multiplexing) peer data exchange algorithm, 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 asynco 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) have been formalized 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.

More details on the PTB-FLA and MPT-FLA APIs are available in the TaRDIS wiki.[39]

3.3.1.2. Flower-based Federated Learning (FFL) APIs

The Flower-based federated learning APIs within the TaRDIS toolkit 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 rely on the facilities that the open-source Flower framework offers for building custom FL solutions.

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

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

The FFL training models API is meant to enable FL ML model training, by supporting a list of provided AI/ML decentralized solutions. It contains implementations for the following algorithms: FedAvg (federated averaging), pFedMe (Personalized Federated Learning with Moreau Envelopes), pFedMeNew (New Personalized Federated Learning with Moreau Envelopes), Anomaly Detection with noisy labels (in progress, it is relevant for the ACT use case) and Distributionally Robust FL (in progress). The FFL input data preprocessing API is enabling the transformation of raw data into a format that is suitable for the FL ML model training process. It provides an easier and most efficient way to deal with different irregularities in the target data set. The FFL ML inference and evaluation API enables gaining valuable output regarding the relevant data and the trained model.

A Flower-based FL model training tool was recently developed, in order to make the FL techniques more accessible to end users. It is aligned with the mentioned APIs and provides a user-friendly (command-line and GUI-based) interface for eased utilization of FL approaches. More details on the FFL APIs are available in the TaRDIS wiki.[40]

3.3.1.3. Fedra (T-WP5-09): A decentralized 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 utilized 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 decentralized 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) Utilizes 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 decentralized 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.

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

This architectural design ensures that Fedra is not just a tool for federated learning, but a comprehensive ecosystem for decentralized, collaborative AI development. More information related to the operation of Fedra can be found in the TaRDIS wiki[41] and on GitHub[42].

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

FLaaS (FL-as-a-Service), developed by Telefónica, is a practical federated learning 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 (as part of the Telefónica use case) will result in an improved or more modular version of FLaaS.

After this integration, components of FLaaS that are standalone might be included in the TaRDIS toolbox, depending on their level of reusability and maturity. This is ongoing work, and therefore, we do not yet have documentation nor finalized APIs for the FLaaS-related components that might be included in the TaRDIS toolbox. Depending on how this effort will evolve, the potential FLaaS-related TaRDIS components will be documented in future relevant deliverables, including D3.5 (3rd iteration of the TaRDIS models and APIs).

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

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

PeersimGym (T-WP5-11) received multiple upgrades since D3.1. The tool is an environment for task offloading in edge systems. There are two components, a simulation built with the Peersim P2P simulator tool that simulates an Edge Systems with three types of nodes: clients, workers and the cloud. Tasks are generated and must be processed by the different nodes in the network or the cloud. This simulation is wrapped in a python class that implements the PettingZoo environment class in Python providing a Markov Game abstraction to the agents interacting with the environment. The PettingZoo environment manages the simulation using a Spring Boot Server wrapping the simulation and REST requests. To see instructions on how to setup and configure and use the environment see the TaRDIS wiki[43] and the PersimGym repository on GitHub.[44]

3.3.2.1.1 - Updates to the environment

While developing the **FAuNO framework (T-WP5-05)** the team further improved the environment to better replicate the scenarios normally considered when doing Task Offloading in Edge Systems. The WP5 team has updated the simulation and extended the PettingZoo API to support the exchange of federated model updates through the network. Concretely, we have:

- Added a mechanism to permit the regular timestep to be subdivided into multiple sub-timesteps. This allows for events to be more spaced in time and having a finer control over the simulation.
- Added a mechanism to track energy consumption of each node's processing and communicating.
- Added an alternative mechanism to compute the channel delay using a channel bit-rate directly instead of using the Shannon-Hartley theorem.

[41] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs
[42] https://github.com/anaskalt/fedra
[43] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Artificial-Intelligence-and-Machine-Learning-APIs
[44] https://github.com/FredericoMetelo/peersim-environment

- Added a mechanism to exchange FL updates between agents through the simulated network. We utilize the A* algorithm to optimally route communications between all nodes in a static network.

3.3.2.2 - FAuNO: Federated AI Network Orchestrator

The WP5 team has implemented a prototype version of the **FAuNO framework (T-WP5-05)**, a Semi-asynchronous Federated Reinforcement Learning method that is capable of solving the Task Offloading problem in Edge Systems. We have the local agents using a Proximal Policy Optimization (PPO) Actor-Critic algorithm[45], to locally learn how to best interact with the environment. And we federate the critic networks of all participants using a client/server Federated Buffering (FedBuff)[46] technique to obtain a global critic model that the participating agents cooperate to build. The details about FAuNO can be found on the TaRDIS wiki.[47]

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 import as follows:

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

as used as such How to Use:

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:

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)`).

[45] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[46] Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., & Huba, D. (2022, May). Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics* (pp. 3581-3607). PMLR.

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

[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

thresholds	List of floats	Threshold values for early exits, it 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	Optimization 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
- The `all_exits` option will train ONLY the added exits (works well for pretrained networks that we don't want to hurt the performance of the body)
- If an integer is provided as a parameter, it will train the exit specified. 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	List of the training loss.

`test_accuracies`	`List of strings`	the accuracy after training.

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

The Dexit 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.
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) Decentralized 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:

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

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. Its 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 import as follows

```
from knowledge_distilation import knowledge_distillation_train
```

and the input parameters are described in the following table.

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.

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

`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 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:

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

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 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 AI and machine learning. The structure of the following subsections is based on the TaRDIS WP6 project tasks (matching the structure of Section 3.4 of Deliverable 3.1):

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

Funded by the European Union

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 will be provided as part of the TaRDIS toolbox. In the following, we discuss the properties and specific APIs that are currently under development and implemented by the TaRDIS team.

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. However, preliminary testing in the development of the example TaRDIS messaging application (to be detailed in Deliverable 6.2 as an example of the use of TaRDIS technology) have shown that this functionality was confusing for developers, and that effectively, the selection of one, or more, overlay networks to support the operation of swarm applications was misleading when using the proposed approach. Due to this, we have abandoned this tool and instead focused our efforts in building different membership abstractions (materialized by overlay networks) that share a common API, that is enriched when that overlay provides additional functionality to the programmer (e.g., a distributed hash table allows the programmer to use decentralized application-level routing). Membership APIs are outlined in Section 3.4.1.2.

3.4.1.2. Membership Abstractions and APIs (T-WP6-02)

TaRDIS has developed several different decentralized 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

[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.

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).

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 optimization function (in our implementation latency between swarm elements).

Finally, 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 plan to enrich this set of abstractions with additional implementations of overlay networks that take full advantage of the new functionalities of Babel-Swarm to support a wide range of swarm applications. Details about these additional membership abstractions will be provided in the future Deliverable 6.2.

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 following events:

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.

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

[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

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 plan to extend this set of communication primitives with other alternatives, taking advantage of the functionalities in Babel-Swarm, including publish-subscribe primitives.

The primitives discussed above rely on a common API also defined in Babel Protocol Commons that include the following events:

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.
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

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

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 (T-WP6-06)

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 accountings 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.

Clients communicate with the nearest PotionDB replica to ensure low latency. A client's transactions are locally executed in that PotionDB's replica. Updates are propagated asynchronously to other locations. Although we assume it to be uncommon, if a client's transaction accesses objects not locally available, locations where those objects are replicated are contacted and involved in the transaction execution.

The API exposed natively by PotionDB has the following operations:

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

3.4.2.2 Arboreal (T-WP6-05)

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,

[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.

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.

Arboreal achieves this by implementing a dynamic data replication protocol that ensures global causal+ consistency — a robust consistency model that maintains a coherent order of data operations across nodes without requiring direct cloud involvement. It organizes nodes into a hierarchical, tree-like structure, where each edge node can directly synchronize with nearby nodes, which optimizes data propagation, recovery from failures, and minimizes the metadata overhead. Furthermore, the system adapts dynamically to changing access patterns and supports mobile clients seamlessly, adjusting data replication to follow users as they move between locations. These properties make Arboreal particularly effective in large-scale, latency-sensitive edge scenarios where data availability, consistency, and resilience are critical.

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

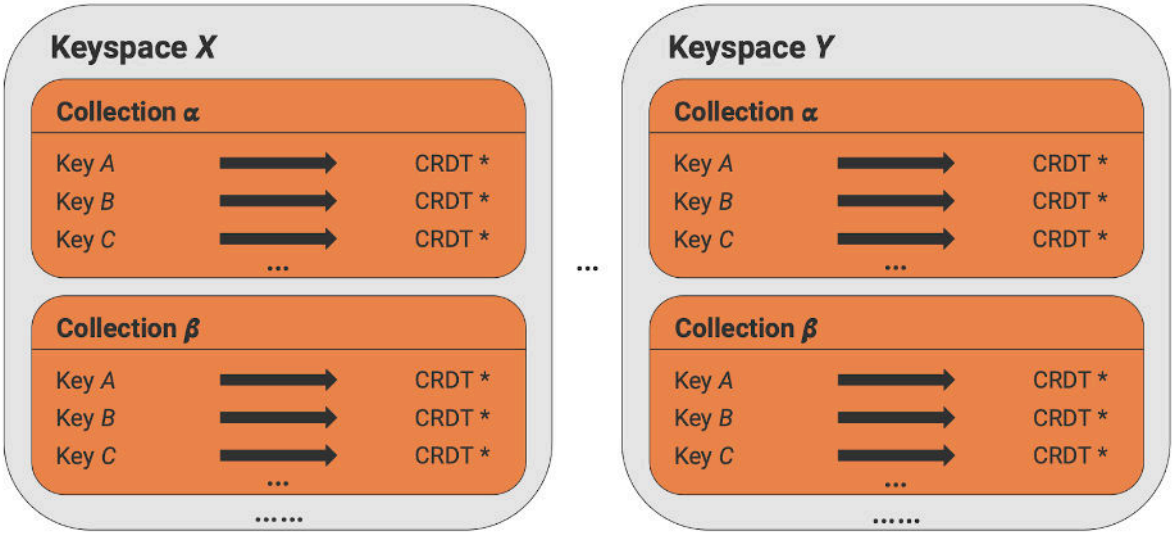
	the previous operation has completed.

3.4.2.3 Nimbus (T-WP6-07)

Nimbus, briefly mentioned in D7.2 as "T-WP6-07 Integrated storage," was not fully detailed as essential features were still being finalized; it is now stable, though development is ongoing. Designed as a fully decentralized 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 decentralized 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.

Nimbus provides strong eventual consistency, a consistency model that guarantees that all nodes reach the same state after receiving all updates, regardless of the order in which updates were applied. This is accomplished by using CRDTs, *Conflict-Free Replicated Data Types,* to synchronize the node's state with the use of epidemic dissemination.

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:



Nimbus client API follows Babel Protocol Commons,[59] a set of common APIs developed in TaRDIS for interacting with distributed systems and their protocols:

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

Request: CreateKeySpaceRequest Reply: CreateKeySpaceReply	A create keySpace request is emitted to a node with the given keySpace identifier as well a set of properties for that keySpace. (i.e., permissions). The corresponding reply is sent back with the status of the operation and an optional message.
Request: CreateCollectionRequest Reply: CreateCollectionReply	A create collection request is emitted to a node with the given collection identifier as well a set of properties for that collection. (i.e., permissions). The corresponding reply is sent back with the status of the operation and an optional message.
Request: DeleteKeySpaceRequest Reply: DeleteKeySpaceReply	This operation is issued by a node when it wishes to delete a keySpace from the system. The reply returns the status of the operation and an optional message.
Request: DeleteCollectionRequest Reply: DeleteCollectionReply	This operation is issued by a node when it wishes to delete a collection from the system. The reply returns the status of the operation and an optional message.
Request: ExecuteRequest Reply: ExecuteReply	An execute request is issued by a node when it pretends to interact with one of the objects of the data store. This request encompasses the type of operation (e.g., READ, WRITE, DELETE etc.), the identifier of the object and the corresponding keySpace and collection, as well as the value in the case of being a write operation. The reply contains the status of the operation, as well as an optional value in case of being a read operation.
Notification: JSONDataNotification	In order to notify the client of new updates brought up to the node by the background synchronization mechanism of Nimbus, the system issues a notification (as a JSON object) of the new updates on the objects that the node replicates.

More details of the API offered by Nimbus can be found in Babel Protocol Commons[60], under the *storage* package, as well as on the TaRDIS wiki.[61]
An under-development implementation can be found on the Nimbus Git repository.[62]

[60] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/babel-protocolcommons
[61] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Data-Management-and-Distribution-Primitives/Decentralised%20Data%20Management%20and%20Replication%20APIs
[62] https://codelab.fct.unl.pt/di/research/tardis/wp6/internal-tools/nimbus

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.

At the time of writing this document, the following third party storage solutions are offered by the TaRDIS toolbox.

- ***Hyperledger Fabric (Blockchain)***

  This adapter implements Hyperledger Fabric client-gateway, the component in charge of invoking transactions on smart contracts deployed in the fabric blockchain network. Hyperledger Fabric[63] is a platform for distributed ledger solutions underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility, and scalability.

- ***C3***

  C3 is designed to extend existing storage systems by integrating the designed replication schema in order to enforce causal+ consistency. At the moment this adapter implements C3 integration with Cassandra (see below).

- ***Cassandra***

  This adapter implements the client-side of Cassandra,[64] a highly performant distributed database, providing high availability and proven fault-tolerance.

- ***Engage***

  This adapter implements the client-side of Engage,[65] a storage system that offers efficient support for session guarantees in a partially replicated edge setting.

Some of these tools are still under testing and further functionalities will be added to support the reconfiguration of the storage solutions at runtime. Further details are given in the draft implementation,[66] as well as the TaRDIS wiki.[67]

[63] https://github.com/hyperledger/fabric-gateway/tree/main
[64] https://codelab.fct.unl.pt/di/research/tardis/wp6/external-tools/cassandra
[65] https://codelab.fct.unl.pt/di/research/tardis/wp6/external-tools/engage
[66] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel/babel-datareplication-adapters
[67] https://codelab.fct.unl.pt/di/research/tardis/toolkit/Documentation/-/wikis/TaRDIS-APIs/Data-Management-and-Distribution-Primitives/Decentralised%20Data%20Management%20and%20Replication%20APIs

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 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.

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.[68] 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.

These monitoring and reconfiguration APIs are still under development and are not yet documented. Some of the already-completed APIs include:

- 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
dissiminateConfig(labels, configList, percentage) → status
getConfigTimeline(config) → timeline
```

[68] https://openmetrics.io/

4 CHALLENGES AND PLANNED WORK

This section discusses two challenges in the ongoing specification and development of the TaRDIS toolbox programming models and APIs: cross-language interoperability (Section 4.1) support for device capabilities (Section 4.2), and toolbox cohesiveness (Section 4.3).

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 at this stage, the APIs and facilities provided by WP4, WP5, and WP6 are being prototyped and developed using the most suitable (given their applications) programming languages.

The plan (already outlined in Deliverable D3.1) is to make such APIs available to other languages, too; however, achieving general cross-language interoperability involves a considerable effort — and for this reason, we will give a higher priority to the programming languages used by the TaRDIS use case applications. To achieve this, we will proceed in two phases:

- First, we plan to leverage industry-standard cross-language translation layers to make an API developed with programming language A available to programs written in programming language B ≠ A. To this end, we may e.g. expose the APIs via gRPC, OpenAPI, or WebSockets/JSON schema. The aim is to achieve quick-to-develop (albeit inefficient) cross-language translation layers when needed. This part of the plan has not been followed yet, because the need for such cross-language interoperability layers has not yet concretely emerged for the implementation of the TaRDIS use cases.
- Then, we will assess the possibility of developing more dedicated and optimised cross-language interoperability layers, for specific pairs of source and target language. This second part of the plan is also on hold, because this need has not yet concretely emerged for the implementation of the TaRDIS use cases.

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 were 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 has not yet been addressed in the TaRDIS programming models and APIs.

We plan to study and evaluate how to improve the support for device capabilities during the rest of the TaRDIS project. This will happen in cooperation with WP7, based on the detailed technical requirements that will emerge as the industry partners’ use cases are redeveloped with the TaRDIS toolbox.

Funded by the European Union

4.3. Ensuring the Cohesiveness of the TaRDIS Toolbox

Until now, 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.

The work on ensuring the TaRDIS toolbox cohesiveness is beginning now, as the toolbox architecture has been stabilised in Deliverable D7.2, and the documentation is being collected on the TaRDIS wiki.

We plan to ensure toolbox cohesiveness as part of the WP3 work towards defining the *TaRDIS development approach*. We plan to write TaRDIS usage tutorials based on the experience gained in developing the TaRDIS use cases, and then organise surveys where programmers are asked to follow the tutorials, and provide their feedback; then, we will use this feedback to improve the TaRDIS toolbox and its documentation (including the tutorials themselves). To prepare these surveys, we will leverage the expertise of the software engineering researchers at NOVA. 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., D3.1) and complementing D7.2 (preliminary evaluation of the TaRDIS toolbox components and use cases architectural specification).

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). As the TaRDIS project activity progresses, the TaRDIS programming models and APIs will undergo further consolidation and alignment with the use cases, towards the final iteration of this deliverable (i.e., D3.5). The TaRDIS project tasks T3.1 (models) and T3.2 (APIs) will keep coordinating the improvement of the documentation of the models and APIs.

Funded by the European Union