SustainML

Application Aware, Life-Cycle Oriented Model-Hardware Co-Design Framework for Sustainable, Energy Efficient ML Systems

Framework architecture, back-end, front-end design and release - Y1

Deliverable D5.1

WP5 - Framework Architecture and Implementation

This project has received funding from the European Union's Horizon Europe research and innovation programme (HORIZON-CL4-2021-HUMAN-01) under grant agreement No 101070408

eProsima
The Middleware Experts

DFKI

Inria

up mem

RPTU Rheinland-Pfälzische Technische Universität Kaiserslautern Landau

IBM

Project

Title:	SustainML: Application Aware, Life-Cycle Oriented Model-Hardware Co-Design Framework for Sustainable, Energy Efficient ML Systems
Acronym:	SustainML
Coordinator:	eProsima
Grant agreement ID:	101070408
Call:	HORIZON-CL4-2021-HUMAN-01
Program:	Horizon Europe
Start:	01 October 2022
Duration:	36 months
Website:	https://sustainml.eu
E-mail:	sustainml@eprosima.com
Consortium:	**eProsima (EPROS)**, Spain **DFKI**, Germany **Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau (RPTU)**, Germany **University of Copenhagen (KU)**, Denmark **National Institute for Research in Digital Science and Technology (INRIA)**, France **IBM Research GmbH**, Switzerland **UPMEM**, France

Deliverable

Number:	**D5.1**
Title:	**Framework architecture, back-end, front-end design and release - Y1**
Month:	12
Work Package:	WP5 - Framework Architecture and Implementation
Work Package leader:	eProsima (EPROS)
Deliverable leader:	eProsima (EPROS)
Deliverable type:	Report (R)
Dissemination level:	Public (PU)
Date of submission:	31/09/2023
Version:	v1.3
Status:	Finished

Version history

Version	Date	Responsible	Author/Reviewer	Comments
v1.0	31/03/2323	eProsima	Mario Domínguez López	D5.1 - First Draft
v1.1	10/09/2323	eProsima	Jesus Poderoso	D5.1 - Review
v1.2	13/09/2323	eProsima	Mario Domínguez López	D5.1 - Apply review suggestions
v1.3	23/09/2323	eProsima	Raúl Sánchez-Mateos Lizano	D5.1 - Final review

© 2023 SustainML | HORIZON-CL4-2021-HUMAN-01 | 101070408

Executive summary

SustainML project aims to develop a design framework and an associated toolkit, so-called SustainML, that will foster energy efficiency throughout the whole life-cycle of Machine Learning (ML) applications: from the design and exploration phase that includes exploratory iterations of training, testing and optimizing different system versions through the final training of the production systems (which often involves huge amounts of data, computation and epochs) and (where appropriate) continuous online re-training during deployment for the inference process. The framework will optimize the ML solutions based on the application tasks, across levels from hardware to model architecture. It will also collect both previously scattered efficiency-oriented research, as well as novel Green-AI methods. Artificial Intelligence (AI) developers from all experience levels can make use of the framework through its emphasis on human-centric interactive transparent design and functional knowledge cores, instead of the common blackbox and fully automated optimization approaches.

This report corresponds to *Deliverable D5.1 - Framework architecture, back-end, front-end design and release - Y1* of the SustainML project. This deliverable establishes the software requirements and covers the detailed architecture and design for the different components comprised within the *SustainML Design Framework*, as well as the release procedure. This document will be updated to reflect the most recent state of the project as it progresses.

Contents

Executive Summary 3

Contents 4

Acronyms 5

1 Introduction 6
1.1 Work Packages and partners . . . 6
1.2 Purpose of this document . . . 7
1.3 Structure of this document . . . 8

2 Background context 9
2.1 Enabling technologies . . . 9
2.1.1 Data Distribution Service (DDS) . . . 9
2.1.2 eProsima Fast DDS . . . 10
2.1.3 Python Bindings . . . 12
2.2 Software requirements . . . 12
2.2.1 General requirements for all the software modules . . . 13
2.2.2 WP1 requirements . . . 13
2.2.3 WP2 requirements . . . 13
2.2.4 WP3 requirements . . . 14
2.2.5 WP5 requirements . . . 14

3 Framework architecture 15
3.1 Data Model . . . 15
3.2 API for the Software Modules . . . 18

4 Software design 20
4.1 Design guidelines . . . 20
4.2 Back-End . . . 21
4.2.1 Module Nodes . . . 21
4.2.2 Orchestrator Node . . . 23
4.3 Front-End . . . 24

5 Release procedure 25
5.1 Versioning . . . 26

6 Results 27
6.1 Module Nodes demonstrator . . . 27
6.1.1 Description . . . 27
6.1.2 Technologies used . . . 28
6.1.3 Experiments . . . 28

7 Conclusions 30
7.1 Future work . . . 30

A API usage examples for the software modules 31

B Screenshots of the Module Nodes demonstrator 35

References 36

Acronyms

AI	Artificial Intelligence.
API	Application Programming Interface.
CFN	Carbon Footprint Node.
DDS	Data Distribution Service.
GUI	Graphical User Interface.
HWN	Hardware Resources Node.
IDL	Interactive Data Language.
ML	Machine Learning.
MLN	Machine Learning Model Node.
QoS	Quality of Service.
RC	Release Candidate.
TEN	Task Encoder Node.
UML	Unified Modeling Language.
WP	Work Package.

1 Introduction

The *SustainML* project aims to develop a design framework and associated toolkit to improve energy efficiency throughout the entire life cycle of machine learning applications: from the design and exploration phase to initial iterations of training, testing and optimizing various system versions until final training in the production systems (often requiring large amounts of data, computations, and epochs) and (where appropriate) continuous online retraining of the inference process during deployment.

The developed framework (*SustainML Design Framework*) aimed to be developed in this project, will provide an energy optimized hardware solution and the corresponding ML model for solving a desired machine learning problem formulated by the user.

The following sections describe in detail the SustainML project workforce and fundamentals.

1.1 Work Packages and partners

The *SustainML Project* is divided into a series of Work Package (WP) that are briefly described below:

WP1 *Energy Consumption Oriented ML Task Modeling* - Leader: DFKI. The aim of this WP is to develop means for parameterizing ML tasks (including associated user defined boundary conditions and optimization criteria) in abstract quantifiable descriptions that will allow the algorithms from WP4 to search for most appropriate implementation, based on the models of the ML methods (WP3) and associated hardware (WP2).

WP2 *Hardware Architectures for Low Power ML* - Leader: RPTU. This work package will explore and refine hardware architectures for machine learning accelerations with the focus on energy efficient implementations. Results and inputs from WP1 will be used to search for the most appropriate implementation, while considering the models of the ML methods (WP3) as well. The overall integration will be done in WP5 together with EPROS.

WP3 *Energy Consumption Optimized ML Toolkit and Methods* - Leader: KU. The aim of this work package is to develop a toolkit of ML methods and optimization approaches that comprehensively considers the resource footprint of different components in the ML pipeline. The toolkit will also include ML cores to handle specific tasks for knowledge recycling.

WP4 *Interaction and User Studies* - Leader: INRIA. The goal of this work package is to create new interaction and visualization approaches that will allow users to interactively explore the trade-offs of competing ML models together with intelligent agents. Exploring ML model alternatives during the development process, before the models enter the training cycles, requires users to express potentially ambiguous project objectives and to understand the trade-offs of ML model alternatives, e.g. time, computing hardware, or estimated CO2 footprint for a particular task.

WP5 *Framework Architecture and Implementation* - Leader: EPROSIMA. This Work Package will implement the overall algorithm-hardware co-designed framework in this work package. This work package connects all WP1-WP4 together and provides the engine directly underneath the design interface.

WP6 *Application Oriented Validation* - Leader: IBM. The aim of this work package is to demonstrate in a quantitative way the improvement of the innovative technology developed in the project. Specifically, we aim to validate the framework developed in WP5 in the context of real-world use cases in two application domains: visual inspection and structure health monitoring in the civil engineering domain, and efficient model-transfer for classification in natural language processing.

WP7 *Dissemination, Exploitation and Management* - Leader: EPROSIMA. The main objectives of this package are listed below:

1. To ensure smooth project execution and community building, dissemination and communication.
2. To monitor project progress and achievements as well as risks and contingencies.
3. To maintain Grant Agreement and Consortium Agreement.
4. To promote communication and information exchange among consortium participants and to the target groups.
5. To produce timely reporting to the Commission.
6. To effectively manage all administrative and financial aspects.

1.2 Purpose of this document

The current report focuses on the software design of the *SustainML Design Framework*. Several partners will be contributing to the project's mainline code across the different *Work Packages*. Figure Figure 1 depicts the interactions among the software-related *Work Packages*.

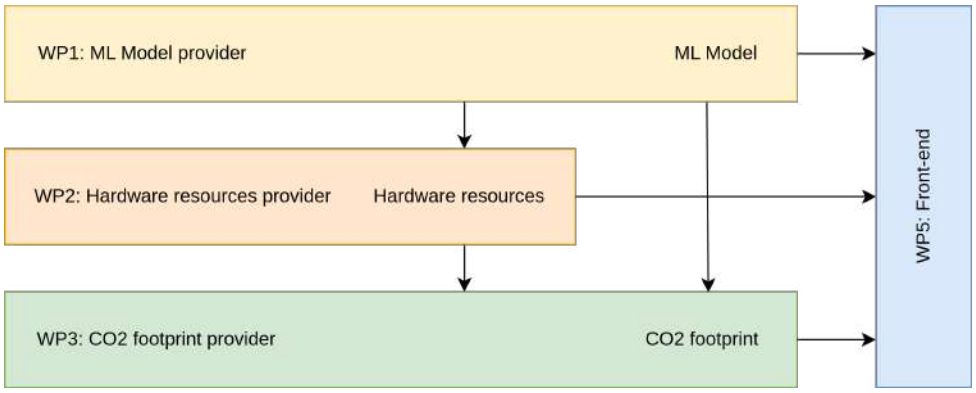

Figure 1: Software based WP interactions

The *SustainML Design Framework* is composed of different *Software Modules*, each one related to a WP, which are specialised in solving the different parts of the ML problem architecture definition, starting from the user's problem description. These steps are basically:

1. Task encoding
2. Generate the Machine Learning model
3. Select an Optimized Hardware for running the proposed model
4. Predict the Carbon footprint
5. Present the results to the user and iterate if required

The main data flows, expected inputs and outputs for each package are showed in figure Figure 2.

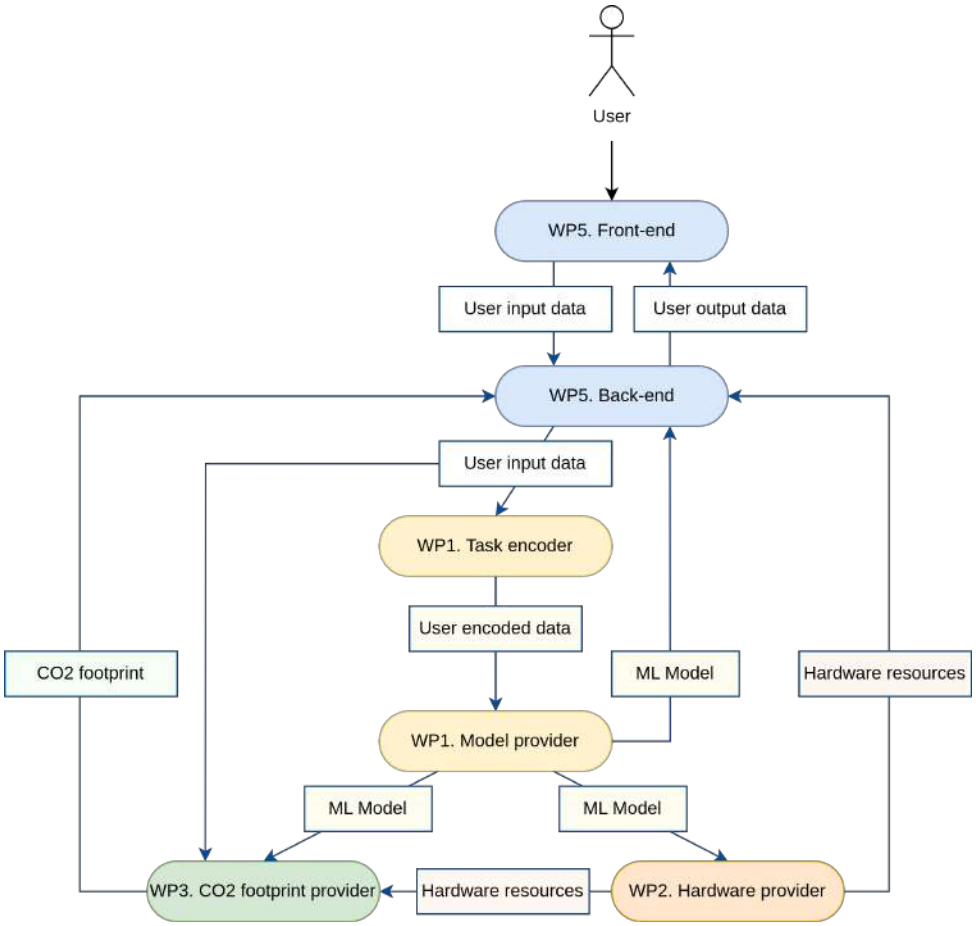

Figure 2: SustainML Pipeline Overview

1.3 Structure of this document

The present document is divided into the following sections:

Background context This section discusses the context and technologies to be used in the project and presents the general and WP-specific software requirements for the *SustainML Design Framework*.

Framework architecture This section describes in depth the main components, data model and API for the software modules (developed by the different partners) that make up the *SustainML Design Framework*.

The **Software design** section details the software diagrams and proper design of the main components that comprise the framework.

Release procedure This sections relates how the software is going to be released, the versioning system and routines to be performed prior to a new software release.

The **Results** section presents several proof of concepts and demonstrators that represent the development stage of the project.

Future work This section analyses the current development stage and proposes the next steps to be addressed.

2 Background context

2.1 Enabling technologies

This section details the different technologies that support the development of the *SustainML Design Framework*. All of them focus on the communication between software modules, the protocols used to support such communication and the tools used to handle the different types of data to be transmitted.

2.1.1 Data Distribution Service (DDS)

Given the distributed nature of the SustainML ecosystem, the use of Data Distribution Service (DDS) is proposed to enable real-time communication between the different SustainML software modules (nodes). As explained in the eProsima Fast DDS documentation [1]:

> *The Data Distribution Service (DDS) is a data-centric communication protocol used for distributed software application communications. It describes the communications Application Programming Interfaces (APIs) and Communication Semantics that enable communication between data providers and data consumers.*
>
> *Since it is a Data-Centric Publish Subscribe (DCPS) model, three key application entities are defined in its implementation: publication entities, which define the information-generating objects and their properties; subscription entities, which define the information-consuming objects and their properties; and configuration entities that define the types of information that are transmitted as topics, and create the publisher and subscriber with its Quality of Service (Quality of Service (QoS)) properties, ensuring the correct performance of the above entities.*
>
> *DDS uses QoS to define the behavioral characteristics of DDS Entities. QoS are comprised of individual QoS policies (objects of type deriving from QoSPolicy).*

The different entities involved in the DDS protocol are also defined in [1].

> *In the DCPS model, four basic elements are defined for the development of a system of communicating applications.*
>
> - *Publisher. It is the DCPS entity in charge of the creation and configuration of the DataWriters it implements. The DataWriter is the entity in charge of the actual publication of the messages. Each one will have an assigned Topic under which the messages are published.*
> - *Subscriber. It is the DCPS Entity in charge of receiving the data published under the topics to which it subscribes. It serves one or more DataReader objects, which are responsible for communicating the availability of new data to the application.*
> - *Topic. It is the entity that binds publications and subscriptions. It is unique within a DDS domain. Through the TopicDescription, it allows the uniformity of data types of publications and subscriptions.*
> - *Domain. This is the concept used to link all publishers and subscribers, belonging to one or more applications, which exchange data under different topics. These individual applications that participate in a domain are called DomainParticipant. The DDS Domain is identified by a domain ID. The DomainParticipant defines the domain ID to specify the DDS domain to which it belongs. Two DomainParticipants with different IDs are not aware of each other's presence in the network. Hence, several communication channels can be created. This is applied in scenarios where several DDS applications are*

> *involved, with their respective DomainParticipants communicating with each other, but these applications must not interfere. The DomainParticipant acts as a container for other DCPS Entities, acts as a factory for Publisher, Subscriber and Topic Entities, and provides administrative services in the domain. See Domain for further details.*

The dynamic addition of new entities to the network is also covered by the the DDS protocol, allowing the dynamic discovery of new instantiated DDS entities. The Discovery Protocol is the mechanism that DDS provides for the discovery of new entities.

2.1.2 eProsima Fast DDS

As described in the *Fast DDS* documentation [2]:

> *eProsima Fast DDS is a C++ implementation of the DDS (Data Distribution Service) Specification, a protocol defined by the Object Management Group (OMG). The eProsima Fast DDS library provides both an Application Programming Interface (API) and a communication protocol that deploy a Data-Centric Publisher-Subscriber (DCPS) model, with the purpose of establishing efficient and reliable information distribution among Real-Time Systems. eProsima Fast DDS is predictable, scalable, flexible, and efficient in resource handling. For meeting these requirements, it makes use of typed interfaces and hinges on a many-to-many distributed network paradigm that neatly allows separation of the publisher and subscriber sides of the communication.*

Among the many features of *Fast DDS*, there are two key functionalities for the development of the *SustainML Design Framework*: *a*) the wide variety of Transports, and *b*) the highly configurable Quality Of Service policies.

Fast DDS provides multiple transport protocols and communication mechanisms, ranging from zero-copy mechanisms to the TCP transport protocol targetting data transmission between DDS entities in WAN networks.

The purpose of the *SustainML Design Framework* is to implement a decoupled and distributed AI design framework able to adapt to local or distributed scenarios optimally.

Moreover, according to [3]:

> *The transport layer provides communication services between DDS entities, being responsible of actually sending and receiving messages over a physical transport. The DDS layer uses this service for both user data and discovery traffic communication. However, the DDS layer itself is transport independent, it defines a transport API and can run over any transport plugin that implements this API. This way, it is not restricted to a specific transport, and applications can choose the one that best suits their requirements, or create their own.*
>
> *eProsima Fast DDS comes with five transports already implemented:*
>
> - *UDPv4: UDP Datagram communication over IPv4. This transport is created by default on a new DomainParticipant if no specific transport configuration is given.*
> - *UDPv6: UDP Datagram communication over IPv6.*
> - *TCPv4: TCP communication over IPv4.*
> - *TCPv6: TCP communication over IPv6.*
> - *SHM: Shared memory communication among entities running on the same host. This transport is created by default on a new DomainParticipant if no specific transport configuration is given (see Shared Memory Transport).*

The *SustainML Design Framework* will exploit the Quality Of Service parametrization described in [4] as follows:

> *QoS (Quality of Service) is a general concept that is used to specify the behavior of a service. Programming service behavior by means of QoS settings offers the advantage that the application developer only indicates 'what' is wanted rather than 'how' this QoS should be achieved. Generally speaking, QoS is comprised of several QoS policies. Each QoS policy is then an independent description that associates a name with a value. Describing QoS by means of a list of independent QoS policies gives rise to more flexibility.*

The *Fast DDS* Quality of Service policies to accommodate the middleware to the different application scenarios, fit particular requirements or restrictions in hardware or network, for instance.

There are some key Quality of Service policies that the *SustainML Design Framework* will take advantage of to provide a distributed reliable communications and optimum resource utilization: *DurabilityQoSPolicy*, *ReliabilityQoSPolicy* and *HistoryQoSPolicy*. The following can be gleaned from the *Fast DDS* online documentation [5]:

- *DurabilityQosPolicy: A DataWriter can send messages throughout a Topic even if there are no DataReaders on the network. Moreover, a DataReader that joins to the Topic after some data has been written could be interested in accessing that information. There are four possible kind values:*
  - *VOLATILE_DURABILITY_QOS: Past samples are ignored and a joining DataReader receives samples generated after the moment it matches.*
  - *TRANSIENT_LOCAL_DURABILITY_QOS: When a new DataReader joins, its History is filled with past samples.*
  - *TRANSIENT_DURABILITY_QOS: When a new DataReader joins, its History is filled with past samples, which are stored on persistent storage (see Persistence Service).*
  - *PERSISTENT_DURABILITY_QOS: (Not Implemented): All the samples are stored on a permanent storage, so that they can outlive a system session.*

  [...]
- *ReliabilityQosPolicy: This QoS Policy indicates the level of reliability offered and requested by the service. There are two configurable fields: kind and max_blocking_time.*
  - *BEST_EFFORT_RELIABILITY_QOS: It indicates that it is acceptable not to retransmit the missing samples, so the messages are sent without waiting for an arrival confirmation. Presumably new values for the samples are generated often enough that it is not necessary to re-send any sample. However, the data samples sent by the same DataWriter will be stored in the DataReader history in the same order they occur. In other words, even if the DataReader misses some data samples, an older value will never overwrite a newer value.*
  - *RELIABLE_RELIABILITY_QOS: It indicates that the service will attempt to deliver all samples of the DataWriter's history expecting an arrival confirmation from the DataReader. The data samples sent by the same DataWriter cannot be made available to the DataReader if there are previous samples that have not been received yet. The service will retransmit the lost data samples in order to reconstruct a correct snapshot of the DataWriter history before it is accessible by the DataReader.*

This max_blocking_time option may block the write operation, hence the max_blocking_time is set that will unblock it once the time expires. But if the max_blocking_time expires before the data is sent, the write operation will return an error. To maintain the compatibility between ReliabilityQosPolicy in DataReaders and DataWriters, the DataWriter kind must be higher or equal to the DataReader kind. And the order between the different kinds is:

$$|BEST_EFFORT_RELIABILITY| < |RELIABLE_RELIABILITY|$$

[...]

- *HistoryQosPolicy This QoS Policy controls the behavior of the system when the value of an instance changes one or more times before it can be successfully communicated to the existing DataReader entities. List of QoS Policy data members:*
  - *kind: Controls if the service should deliver only the most recent values, all the intermediate values or do something in between. See HistoryQosPolicyKind for further details. There are two possible values:*
    - *KEEP_LAST_HISTORY_QOS: The service will only attempt to keep the most recent values of the instance and discard the older ones. The maximum number of samples to keep and deliver is defined by the depth of the HistoryQosPolicy, which needs to be consistent with the ResourceLimitsQosPolicy settings. If the limit defined by depth is reached, the system will discard the oldest sample to make room for a new one.*
    - *KEEP_ALL_HISTORY_QOS: The service will attempt to keep all the values of the instance until it can be delivered to all the existing Subscribers. If this option is selected, the depth will not have any effect, so the history is only limited by the values set in ResourceLimitsQosPolicy. If the limit is reached, the behavior of the system depends on the ReliabilityQosPolicy, if its kind is BEST_EFFORT the older values will be discarded, but if it is RELIABLE the service blocks the DataWriter until the old values are delivered to all existing Subscribers.*
  - *depth: Establishes the maximum number of samples that must be kept on the history. It only has effect if the kind is set to KEEP_LAST_HISTORY_QOS and it needs to be consistent with the ResourceLimitsQosPolicy, which means that its value must be lower or equal to max_samples_per_instance.*

(...)

2.1.3 Python Bindings

eProsima will deliver a communication library that enables the SustainML abstract all the communications SustainML nodes from within a Python environment by binding the C++ library.

This is a very useful feature for the SustainML project, as most of the code produced for *SustainML Design Framework* is written in Python, and Python will most likely be the chosen language for the main framework with which the end users will interact.

2.2 Software requirements

The *SustainML Design Framework* architecture needs to integrate all the modules developed in each WP, offering a way of connecting them in a distributed approach and presenting the data to the user so that the ML problem statement described by the user can be correctly characterized.

To provide a clear and comprehensive description of what the software is intended to achieve, outlining its functionality, features, and constraints, the following requirements and agreements have been set:

2.2.1 General requirements for all the software modules

Following are the list of general requirements for the *SustainML Design Framework*:

SUSML-FRMW-REQ:01 Ubiquitous language the output from each WP shall be delivered as a Python executable module.

SUSML-FRMW-REQ:02 Distributed Communications the developed software modules shall communicate over the DDS protocol.

SUSML-FRMW-REQ:03 API Compliance the developed Python modules shall comply with the interfaces (API's) defined in the corresponding specific requirements.

SUSML-FRMW-REQ:04 Application Lyfecycle The developed Python modules shall accept a *Status* input so that the execution status can be returned as feedback.

The WP specific requirements are depicted in the following sub-sections:

2.2.2 WP1 requirements

The Work Package 1 is in charge of interpreting the user ML problem definition, giving the optimum, energy-optimized ML model for the user-described problem definition.

WP1-TASKENC-REQ:01 : The WP1 shall define a Task Encoder Node (TEN).

WP1-TASKENC-REQ:02 : The TEN shall accept as input the resulting raw data from the ML application description given by the user.

WP1-TASKENC-REQ:03 : The TEN shall output a sequence of *keywords*.

WP1-MLMODEL-REQ:01 : The WP1 shall define a Machine Learning Model Node (MLN).

WP1-MLMODEL-REQ:02 : The MLN shall input a sequence of keywords.

WP1-MLMODEL-REQ:03 : The MLN shall output a path to the file containing the machine learning model.

WP1-MLMODEL-REQ:04 : The machine learning model provided by the MLN shall be in the *ONNX* format.

WP1-MLMODEL-REQ:05 : The MLN shall output a path to the file containing the machine learning model properties.

WP1-MLMODEL-REQ:06 : The machine learning model properties file provided by the MLN shall be in the *json* format.

2.2.3 WP2 requirements

WP2-HWRES-REQ:01 : The WP2 shall define a Hardware Resources Node (HWN).

WP2-HWRES-REQ:02 : The HWN shall accept as an input, the machine learning model provided by the WP1-MLMODEL-REQ:01 .

WP2-HWRES-REQ:03 : The HWN shall accept as an input, the machine learning model properties provided by the WP1-MLMODEL-REQ:05 .

WP2-HWRES-REQ:04 : The HWN shall output an optimum detailed hardware description.

WP2-HWRES-REQ:05 : The HWN shall provide the power consumption estimate of the proposed hardware.

2.2.4 WP3 requirements

WP3-CO2FOOTPRT-REQ:01 : The WP3 shall define a Carbon Footprint Node (CFN).

WP3-CO2FOOTPRT-REQ:02 : The CFN shall accept as an input, the machine learning model provided by the WP1-MLMODEL-REQ:01 .

WP3-CO2FOOTPRT-REQ:03 : The CFN shall accept as an input, the hardware architectures from WP2-HWRES-REQ:04 and WP2-HWRES-REQ:05 .

WP3-CO2FOOTPRT-REQ:04 : The CFN shall accept as an input, the geographic location provided by the user. In addition, it can optionally query the local power grid carbon intensity for the region.

2.2.5 WP5 requirements

WP5-ARCH-REQ:01 : The WP5 shall develop a Python library communication layer, which, using Fast DDS, shall abstract the communications. This library will be later imported and used by the rest of the modules.

WP5-ARCH-REQ:02 : The WP5 shall develop a backend software that collects all the data from the rest of the Work Packages.

WP5-ARCH-REQ:03 : The WP5 shall develop a Graphical User Interface (GUI) based frontend software for presenting the data and guiding the user through the SustainML interactive-design framework.

WP5-ARCH-REQ:04 : The GUI application shall support a multi-view layout so that the user can request multiple Machine Learning problems simultaneously.

3 Framework architecture

This section describes the main components that make up the *SustainML Design Framework* and presents the data model for the different types of messages that are exchanged among the software modules. Finally, the API to be used by the other partners to develop their software modules is detailed.

The entrypoint of the *SustainML Design Framework* architecture is a multi-view GUI application (WP5-ARCH-REQ:03) with which the user interacts by requesting the energy-optimized solution for new ML problems (tasks).[1]

The Back-End is the collection of modules in charge of collecting all the required data for the *Front-End*. To manage it in a decoupled approach, and complying with the SUSML-FRMW-REQ:01 Ubiquitous language, an orchestrator node exchanges information over DDS with each of the developed Python modules. Each of the developed Python modules conforms a *Node*.

The resultant architecture overview is showed in the figure below:

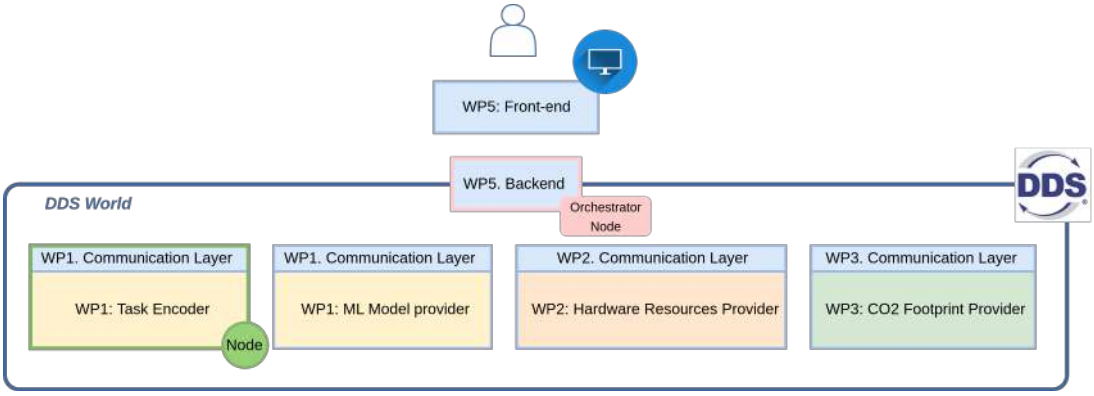

Figure 3: SustainML Framework Architecture

All the modules shall import its corresponding Python library developed by EPROS (WP5-ARCH-REQ:04), so that each module can be abstracted from the DDS communications [2].

3.1 Data Model

Following are the Data Structures definitions using the Interactive Data Language (IDL). Every node in the Framework needs to provide a continuous feedback status to the Orchestrator Node. This is modelled with the *NodeStatus* data structure.

```
enum Status
{
        INACTIVE,
        INITIALIZING,
        IDLE,
```

[1] This visual GUI is mainly developed within the Front-End. Each of the views (independent Machine Learning problem) comprising the layout is uniquely identified by a *task_id*.

[2] In this initial stage, the ML problem description will be provided as a form. This form shall be filled by the user and could include sections like "Provide the Problem description", "At which location is the algorithm going to be run?".

```
        RUNNING,
        ERROR,
        TERMINATING
};

enum TaskStatus
{
        WAITING,
        RUNNING,
        ERROR,
        SUCCEEDED
};

enum ErrorCode
{
        NO_ERROR,
        INTERNAL_ERROR
};

struct NodeStatus
{
    Status node_status;
    TaskStatus task_status;
    ErrorCode error_code;
    @Key long task_id;
    string error_description;
    @Key string node_name;
}
```

Listing 1: Node Status Data Type IDL description.

In order to correctly manage the lifecycle of nodes and tasks, a *NodeControl* data structure is defined. This data structure is internally used in the communication library.

```
enum CmdNode
{
    NO_CMD,
    CMD_START_NODE,
    CMD_STOP_NODE,
    CMD_RESET_NODE,
    CMD_TERMINATE_NODE
};

enum CmdTask
{
    NO_CMD,
    CMD_STOP_TASK,
    CMD_RESET_TASK,
    CMD_PREEMPT_TASK,
    CMD_TERMINATE_TASK
};

```

```
struct NodeControl
{

    CmdNode cmd_node;
    CmdTask cmd_task;
    string target_node;
    long task_id;
    @Key string source_node;
};
```

Listing 2: Node Control Data Type IDL description.

The *UserInput* data structure carries information about the input from the user, when describing a new task (Machine Learning problem). It is comprised of the following fields:

- A string with the raw input problem description.
- The geo-location in which the ML problem is going to take place.
- The task_id identifying the id of the ML problem to solve.

```
struct GeoLocation
{
    string continent;
    string region;
};

struct UserInput
{
    string problem_description;
    GeoLocation geo_location;
    @Key long task_id;
};
```

Listing 3: User Input Data Type IDL description.

The *EncodedTask* data structure represents the output from the TEN. It is composed by:

- A sequence of strings identifying the *keywords* from the user input problem description.
- The task_id identifying the corresponding ML problem.

```
struct EncodedTask
{
    sequence<string> keywords;
    @Key long task_id;
};
```

Listing 4: Encoded Task Data Type IDL description.

The *MLModel* data structure represents the output from the MLN. It is divided in the following fields:

- A string containing the path to the ML model.
- A string with the raw model.
- A string containing the path to the properties of the model.

- A string with the model properties.
- The task_id the ML problem it refers to.

The *model* and *model_properties* can be optionally filled. The reasoning for include them is to overcome situations in which the model is generated into a remote machine.

```
struct MLModel
{
    string model_path;
    string model; // Remote cases
    string model_properties_path;
    string model_properties;
    @Key long task_id;
};
```

Listing 5: Machine Learning Model Data Type IDL description.

The HWN selects a best-suited energy-optimized hardware according to the ML model. To represent that information, the *HWResource* data structure is defined containing the following fields:

- A string with the detailed hardware description.
- The power consumption (W).
- The task_id of the task it refers to.

```
struct HWResource
{
    string hw_description;
    double power_consumption;
    @Key long task_id;
};
```

Listing 6: Hardware Resource Data Type IDL description.

Finally, in order to model the output from the CFN, the *CO2Footprint* data structure consisting in the following fields:

```
struct CO2Footprint
{
    double co2_footprint;
    double energy_consumption;
    double carbon_intensity;
    @Key long task_id;
};
```

Listing 7: CO2 Footprint Data Type IDL description.

3.2 API for the Software Modules

To comply with the SUSML-FRMW-REQ:01 Ubiquitous language for each Work Package, a common API and operation principle is defined for the Python communication library.

For each module, the first step would be importing the corresponding node and data types. As stated in the Framework architecture, each of the working units of the Framework is named as a *Node*. Consequently, each module shall instantiate its *Node*.

Every node will implement its functionality into an appropriate callback, receiving the corresponding inputs in each case, plus the status object and its output data structure. In order for the node to be able to invoke the callback when the required inputs are received for a particular task_id, it is needed to register it in the node. During the callback, it is advised to perform the corresponding calls to the *update()* method of the *NodeStatus*[3] object to track the status of the order identified by the task_id.

Before finishing the callback, if there were no errors and the execution was fine, the corresponding output data structure shall be filled.

Finally, each module should call the *spin()* method of the *Node* so that the node can start running.

All the template script examples can be found in the API usage examples for the software modules appendix.

[3]The corresponding *enum* names describing the different statuses can be used for the purpose.

4 Software design

In this section, the *SustainML Design Framework* software design is presented and discussed. It is important to note that this does not intend to be the final product design, but a dynamic design that represent the state of the actual project requirements and tackles possible future demands.

4.1 Design guidelines

Software design is a critical phase in the project's development lifecycle that lays the foundation for creating an efficient, maintainable, and robust framework. It encompasses the process of transforming requirements and specifications (see Software requirements) into a comprehensive plan that outlines the architecture, components, interfaces, and interactions of the software system.

Effective software design is essential to ensure that the final *SustainML Design Framework* meets user needs, is adaptable to future changes, and exhibits high performance by carefully considering factors such as modularity, scalability, security, and user experience.

Several key aspects underpin successful software design.

- Modularity, the practice of breaking down a complex system into smaller, manageable components, promotes reusability and ease of maintenance.
- Encapsulation emphasizes encapsulating data and behavior within classes or modules, enhancing data integrity and promoting a clear separation of concerns.
- Inheritance and polymorphism enable designers to create hierarchies of classes that facilitate code reuse and extensibility.
- Design patterns, established reusable solutions to common design problems, provide blueprints for addressing specific challenges while ensuring best practices.

Furthermore, iterative and incremental design approaches allow for the continuous refinement of design decisions as the project evolves, accommodating changing requirements and fostering adaptability.

In particular, the *SustainML Design Framework* adheres to the *SOLID Software Design Principles* and makes use of the *Unified Modeling Language* to create and represent the software design.

SOLID design principles

In order to create codebases that are more maintainable, flexible, and resilient to changes. These principles encourage modular design, clear separation of concerns, and a focus on the long-term evolution of the project's requirements.

The SOLID [6] principles consists of five design principles in object-oriented programming in which each letter in the acronym represents a specific principle:

Single Responsibility Principle (SRP) : This principle states that a class should have only one reason to change, meaning it should have a single responsibility or purpose. A class should encapsulate a single piece of functionality or behavior. This promotes modularity, making it easier to understand, maintain, and modify individual components without affecting the entire system.

Open/Closed Principle (OCP) : The Open/Closed Principle suggests that software entities, such as classes, modules, or functions, should be open for extension but closed for modification. In other words, you should be able to add new functionality to a system without altering existing code. This is typically achieved through the use of inheritance, interfaces, and abstract classes, allowing you to extend behavior without changing the existing codebase.

Liskov Substitution Principle (LSP) : The Liskov Substitution Principle emphasizes that objects of a derived class should be able to replace objects of the base class without affecting the correctness of the program. In simpler terms, if a class is a subclass of another class, it should be able to be used interchangeably with its parent class without causing unexpected behavior. This principle helps maintain consistency and predictability in object-oriented systems.

Interface Segregation Principle (ISP) : The Interface Segregation Principle states that clients should not be forced to depend on interfaces they do not use. In other words, a class should not be obligated to implement methods that it doesn't need. This principle encourages the creation of smaller, focused interfaces rather than large, monolithic ones. By doing so, it prevents unnecessary dependencies and minimizes the impact of changes.

Dependency Inversion Principle (DIP) : The Dependency Inversion Principle suggests that high-level modules should not depend on low-level modules; both should depend on abstractions. Furthermore, abstractions should not depend on details; details should depend on abstractions. This principle promotes the use of interfaces or abstract classes to define the interactions between components, enabling easier substitution of implementations and reducing tight coupling between different parts of the system.

Unified Modeling Language

The Unified Modeling Language (UML) [7] is a standardized visual modeling language used in software engineering to describe, visualize, and document the structure and behavior of software systems and processes. UML provides a set of graphical notations and diagrams that allow software developers, analysts, and designers to communicate and capture different aspects of a system's design and functionality. It was developed to address the need for a common language that can bridge the gap between various stakeholders involved in software development.

UML diagrams cover a wide range of perspectives, including structural aspects (like classes, objects, components), behavioral aspects (such as interactions, state machines), and even architectural views. Some common types of UML diagrams include class, use case, sequence, activity, state machine, component or deployment diagrams.

4.2 Back-End

4.2.1 Module Nodes

Within the *SustainML Design Framework*, a node refers to each one of the software modules that performs a particular task, conforms a single executable unit and can be locally or remotely deployed, as described in the SustainML Framework Architecture.

In essence, the node library provides an abstraction layer over the DDS communications and data flow management so that each one of the partners can focus on its corresponding ML task in the project (see Introduction).

Class diagram

The class diagram in Figure 4 depicts the main classes and responsibilities, consistent with SOLID design principles:

- The Node class aggregates all the DDS-related entities in order to publish and subscribe to data. It also contains the dispatcher.
- The Dispatcher carries information from the number of samples received in the different inputs. Each node expects a sample from all its inputs with the same *task_id* in order to perform its task.

It is important to note that the *task_id* must be unique among samples of the same Type. As a consequence, it is invalid to receive two samples with the same task_id in the same input queue.

- The NodeListener class implements the subscription callbacks. When a new sample is received from any of the inputs, the NodeListener stores the sample in its SamplePool and notifies the dispatcher. The SamplePool is an utility class that uses a fixed block of memory that guarantees contiguity, allowing for a better performance.
- The Node class provides the basic methods and interfaces so that it can be specialized by each of the final node classes (TaskEncoder, MachineLearningModel, HardwareResources and CarbonFootprint). Each specialization must provide the implementation for the *publish_to_user()* method.

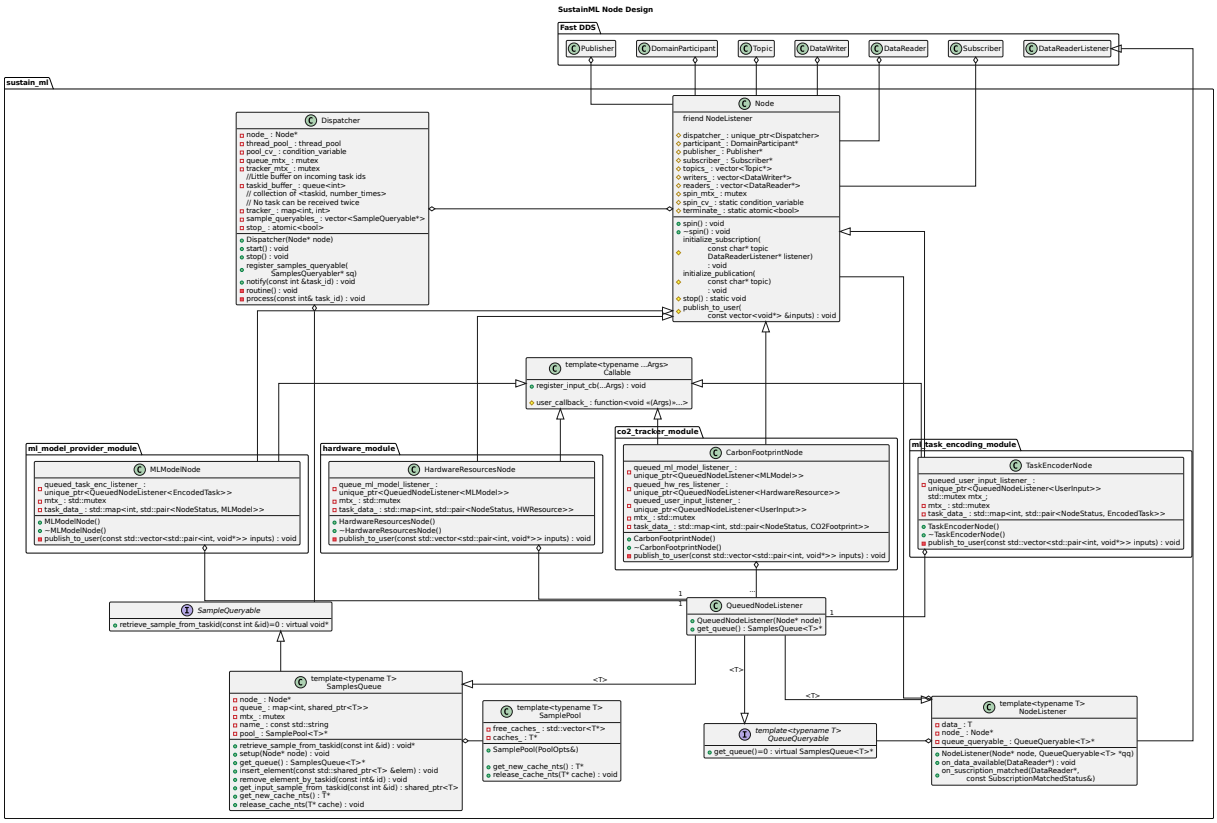

Figure 4: Node UML Design

Flow Diagrams

Following (Figure 5 and Figure 6) are the flow diagrams depicting the initialization of a node and retrieval of a new sample. The different flows represent the interaction among the main entities included in the Node UML Design.

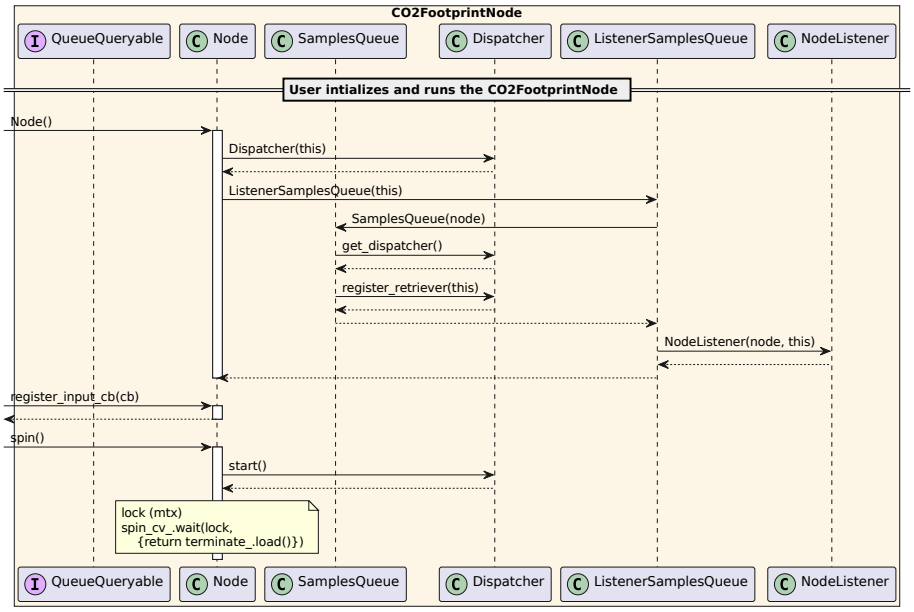

Figure 5: Initialization Flow Chart

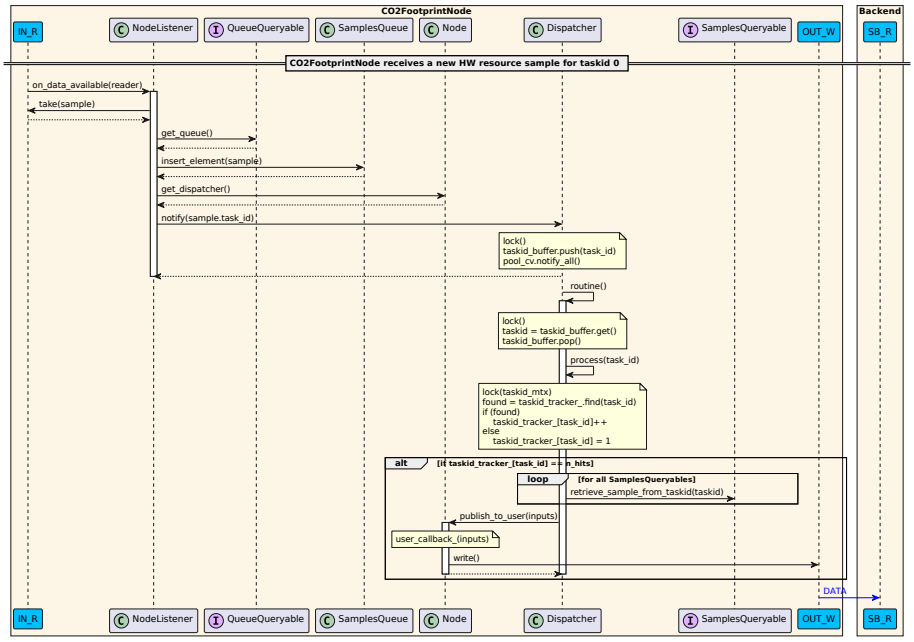

Figure 6: New Sample Flow Chart

4.2.2 Orchestrator Node

The orchestrator node, is the entity that manages the orders received from the front end (actions derived from the user's interaction with the GUI) to the different nodes, and provide feedback of the status of a particular task to the front-end, in order to be displayed.

The orchestrator subscribes to each nodes's intermediate output (DDS topic) to provide traceability of a particular task and track its status. In addition, the orchestrator will act as the lifecycle manager of the different nodes, being capable of starting, pausing, resetting or shutting down nodes when convenient (see the reserved event names defined in Data Model).

As the project its in it initial development stage, the orchestrator is currently is under design stage and development. Hence, this section will be updated in upcoming versions of this document.

4.3 Front-End

The front-end component within the software architecture of the *SustainML Design Framework*, is the user-facing software component. It's the part of the application that users interact with directly. The front end is responsible for displaying information, receiving user input for defining a new task, and providing a means for users to interact with the underlying functionality of the *SustainML Design Framework*.

The front-end includes all the visual elements conforming the user interface, such as windows, dialogs, buttons, text fields, menus, icons, and other graphical elements. The UI of the *SustainML Design Framework* will be designed to be intuitive and user-friendly to exploit its dissemination strategies and reach all kind of end-users: from novel to experts in AI. Besides, the front-end will handle user input, such as mouse clicks, keyboard input and event handling, it will process these inputs, and initiate actions or update the display, synchronised with the back-end.

Creating a positive user experience for the wide range of the target audience in SustainML is a primary goal for the front-end. This involves considering factors like usability, accessibility, performance, and user satisfaction. The overall design for the UI, UX (User experience) and layout of the front-end will take into account the results and feedback from the WP4 (subsection 1.1).

© 2023 SustainML | HORIZON-CL4-2021-HUMAN-01 | 101070408

5 Release procedure

With the aim of establishing an standardized way of proceeding throughout software releases and versions, the SustainML project sets a series of guidelines to correctly address the evolution and traceability of the developed software.

Beta and alpha releases will be important in early stages of the software development lifecycle, each serving distinct purposes that contribute to the overall quality and success of a software product.

Alpha releases will the first versions of the *SustainML Design Framework* product made available for the community. These versions are not feature-complete and may contain significant bugs and incomplete functionality. They are used primarily for testing. They help identify and rectify critical issues, bugs, and performance bottlenecks before the software reaches a wider audience. Subsequently, it allows developers to catch and address issues at an early stage, reducing the likelihood of critical problems in later stages of development.

Following alpha releases, the SustainML project will make use of beta release versions that will allow for a wider testing (stress, compatibility) with the core functionality developed and will be assessed in different scenarios by the community and AI audience of different levels of expertise.

In addition, prior to each release, a new Release Candidate (RC) and release testing must be generated and addressed. Those, are an essential part of the *SustainML Design Framework* software development release procedure for several reasons:

Quality Assurance: RCs represent a near-final version of the software. They are thoroughly tested to ensure that they are stable and meet the quality standards expected in the final release. This extensive testing helps identify and fix any lingering bugs or issues, ensuring a more reliable product.

Stress Testing: Release candidates are subjected to stress and performance testing to assess how the software performs under heavy loads and in various scenarios. This helps identify potential bottlenecks and scalability issues.

Compatibility Testing: RCs are tested across a range of hardware configurations, operating systems, and software environments to ensure compatibility. This is crucial for ensuring that the software works seamlessly for a broad user base.

Feature Freeze: RCs typically represent a feature freeze, meaning that no new features are added at this stage. This stability allows for a final review of all features and functionality, ensuring they are complete and meet the requirements.

Documentation and Training: RCs are used to finalize documentation, user manuals, and training materials. This ensures that users have access to accurate information when the final release is launched.

Preparation for Deployment: The RC phase provides an opportunity to conduct dry runs of the deployment process, ensuring that all necessary procedures and resources are in place for a smooth release.

Risk Mitigation: Identifying and addressing any issues in the RC stage reduces the risk of critical defects or problems occurring in the final release. This helps prevent post-release hotfixes and patches, which can be disruptive to users.

The SustainML partnership will agree and elaborate the release calendar of the software releases from which to define the freeze and release testing dates for the different versions.

5.1 Versioning

The versioning [8] rationale within *SustainML Design Framework* will consist of three segments: *Major*, *minor*, and *patch* in the format *X.Y.Z*. Each segment having a specific purpose:

- Major Version: This number represents a significant, often backward-incompatible changes to the software. A major version change usually indicates substantial updates, a public Application Programming Interface (API) backward-compatibility break or the introduction of major new features.
- Minor Version: The minor version number refers to smaller, backward-compatible feature additions or enhancements. It indicates substantial but non-disruptive changes.
- Patch Version: The patch version number is increased for minor, backward-compatible bug fixes, security updates, or other minor improvements that don't introduce new features.

For instance, if the current version is *1.5.0* and a patch is released to address a few bugs, the new version might be labeled as *1.5.1*. This signifies that it's a backward-compatible update with minor changes aimed at improving the stability and reliability of the software.

Typically, the alpha release versions will correspond to 0.0.X and beta releases will take the range 0.1.X.

6 Results

This section presents several proof of concepts and demonstrators that represent the development stage of the *SustainML Design Framework* project. This section is meant to be continuously updated.

6.1 Module Nodes demonstrator

The *SustainML Design Framework* is composed of several software modules (Figure 3). One of the core concepts and components of the system are the different *module nodes* that solve a particular task within the project: extracting the key words from the user ml problem definition to conform an encoded task, choosing an optimal machine learning model that fits the problem, selecting an energy-optimized hardware and estimating the carbon footprint.

To achieve that, each of the module nodes integrates a communications library (a.k.a Node library) responsible for all the DDS communications. The current demonstrator exemplifies the distributed communications through DDS.

6.1.1 Description

This is a CLI (Command Line Interface) terminal-based demonstrator in which four nodes (Task Encoder, ML model, Hardware Resources and Carbon Footprint) have been created and its implementation has been mocked to wait for some time in order to simulate the *processing time*. In addition, there is a first User Input node that acts as an entrypoint i.e representing the input ML problem of the user. All the messages exchanged among the nodes complies with the Data Model.

The diagram in Figure 7 depicts the demonstrator schema according to Figure 2.

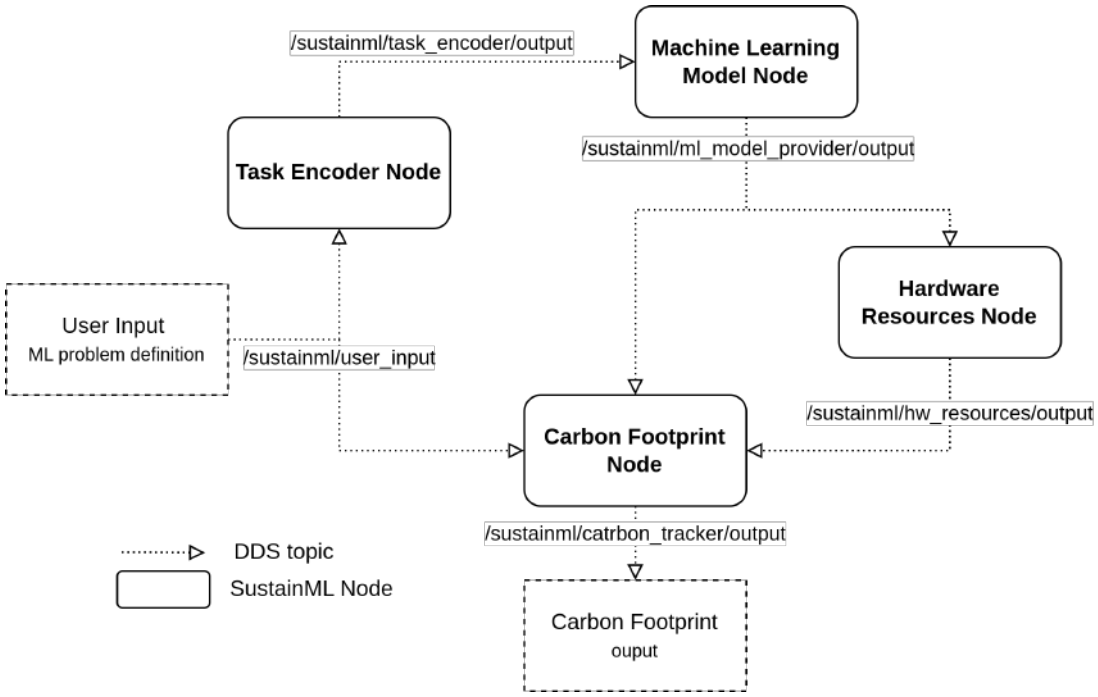

Figure 7: Module Nodes and Topics

The user inputs a new task with the keyboard in the User Input node, which publishes the message to the TEN and CFN. After processing the message, the TEN outputs the encoded tasks to the MLN node that generates a sample MLModel message and publishes it for the HWN and CFN. After the HWN has finished, it publishes it output to the CFN which, having all the three required inputs, processes and generates a new CO2 footprint estimation.

6.1.2 Technologies used

In this initial stage of the project, C++ has been used to produce the demonstrator, but the node library will later be provided in Python so that each partner can develop its Python module, complying with the General requirements for all the software modules.

The eProsima Fast DDS library has also been used for the demonstrator.

6.1.3 Experiments

The Module Nodes demonstrator comprises two scenarios: single and multiple processes, that are detailed below:

Single process scenario In this first scenario all the nodes are executed in the same process, as showed in Figure 8.

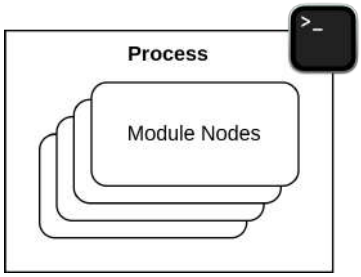

Figure 8: Module Nodes Demonstrator in Single Process scenario

The real capture screenshots can be found in Screenshots of the Module Nodes demonstrator appendix section.

Inter-process scenario In this other scenario all the nodes are executed within different processes, as showed in Figure 9. In this case, if launched in the same host, the Shared Memory Transport (SHM) is used to communicate data between the different processes, taking advantage of the Fast DDS capabilities.

In particular, this scenario exemplifies the native distributed approach that can be achieved in the project, allowing for a loosely coupled component design and better scalability and re-usability.

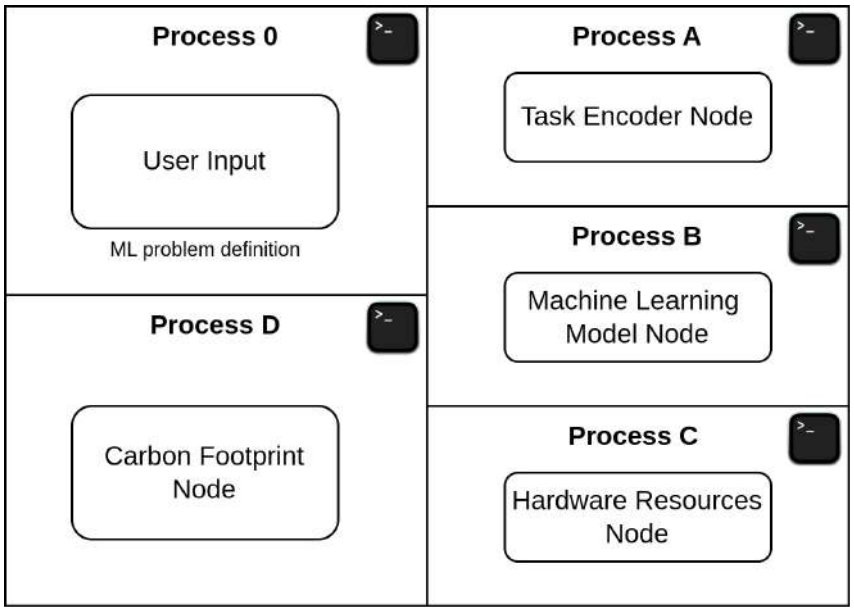

Figure 9: Module Nodes Demonstrator in Inter Process scenario

The corresponding screenshots can be found in Screenshots of the Module Nodes demonstrator.

7 Conclusions

This document presented the software design of the *SustainML Design Framework*. It is not intended to be the final design but an insight on how it is going to be built, and it could change as the project advances depending on the requirements that may arise or the limitations encountered during the development process. However, it is intended to be a complete, robust and functional design fulfilling all the SustainML project Software requirements.

The *SustainML Design Framework* software design has the following strengths:

Decoupled modular design The modular design nature of the *SustainML Design Framework* exploits the distributed capabilities of the underlying DDS protocol and makes easier the integration of the different software modules developed by each of the projects' partners.

Scalability The software design internals of the *SustainML Design Framework* are meant to be for long-term, highly scalable and flexible software product. The design principles applied ensure robustness and a quick response to quick changes.

Adaptability The *SustainML Design Framework* is endowed with a highly configurable network Quality Of Service policies (Enabling technologies) to adapt to a wide variety of network scenarios, capable of being optimally adjusted to each of them i.e loosy networks, restricted bandwidth scenarios, WAN (Wide Area Network) or LAN (Local Area Network), intra-process/inter-process, or a mix between all of them.

In addition, and complying with SustainML project objectives, the *SustainML Design Framework* can be carefully customized to fit low power consumption hardware devices with limited resources. This kind of hardware will be frequently preferred, as it represents a more sustainable, energy-optimized AI solution with a lower CO2 footprint.

Security The security of communications when working across networks is crucial, so it is a matter to be taken into account in every framework including network communications. *SustainML Design Framework* internal nodes, exchange data to provide an accurate, green AI solution to the problem described by the user.

The possibility of adding user authentication, access permissions and data encryption over the network is given by *Fast DDS* communications middleware, avoiding the need for extra security features. This, and other secure mechanisms to join a network of SustainML nodes will be considered for future versions.

7.1 Future work

The software design of the *SustainML Design Framework* software design comprises two core modules: the Front-End the Back-End. This latter one has been the most developed at this initial stage. In turn, Back-End can be divided in two components: the Module Nodes and the Orchestrator Node (Figure 2). A series of experiments for the Module nodes (Figure 7) were presented whereas the Orchestrator Node design and implementation is currently under development. Further work regarding the Front-End is to be addressed with the feedback of WP4 (Work Packages and partners) with the aim of creating a positive user experience.

A API usage examples for the software modules

Python implementation template for the TEN:

```
from SustainML import TaskEncoderNode
from SustainML import UserInput
from SustainML import NodeStatus
from SustainML import EncodedTask

def user_input_cb(user_input, module_status, output):
    ...                                 # set up the job
    status.update(Status.RUNNING)       # notify to the backend that job has
        started
    ...                                 # perform the job
    status.update(Status.SUCCEEDED)     # notify to the backend that job has
        finished
    output.keywords = ...               # record the output in the structure

if __name__ == "__main__":
    node = TaskEncoderNode()                    # register node
    node.register_input_cb(user_input_cb)       # select callback
    node.spin()                                 # start the job
```

Listing 8: Implementation template for the TEN

Python implementation template for the MLN:

```
from SustainML import MLModelNode
from SustainML import EncodedTask
from SustainML import NodeStatus
from SustainML import MLModel

def encoded_task_cb(encoded_task, status, output):
    ...                                 # set up the job
    status.update(Status.RUNNING)       # notify to the backend that job has
        started
    ...                                 # perform the job
    status.update(Status.SUCCEEDED) # notify to the backend that job has
        finished
    output.model_path = ...             # record the output in the structure
    output.model_properties = ...

if __name__ == "__main__":
    node = MLModelNode()                        # register node
    node.register_input_cb(encoded_task_cb) # select callback
    node.spin()                                 # start the job
```

Listing 9: Implementation template for the TEN

Python implementation template for the HWN:

```
from SustainML import HardwareResourcesNode
from SustainML import MLModel
from SustainML import NodeStatus
from SustainML import HardwareResource

def ml_model_cb(ml_model, status, output):
    ...                                 # set up the job
    status.update(Status.RUNNING)       # notify to the backend that job has
        started
    ...                                 # perform the job
    status.update(Status.SUCCEEDED) # notify to the backend that job has
        finished
    output.hw_description = ...         # record the output in the structure
    output.power_consumption = ...

if __name__ == "__main__":
    node = HardwareResourcesNode()      # register node
    node.register_input_cb(ml_model_cb) # select callback
    node.spin()                         # start the job
```

Listing 10: Implementation template for the HWN

Python implementation template for the CFN:

```
from SustainML import CarbonFootprintNode
from SustainML import HWResource
from SustainML import UserInput
from SustainML import MLModel
from SustainML import NodeStatus
from SustainML import HardwareResource

def input_cb(hw_resource, user_input, ml_model, status, output):
    ...                                 # set up the job
    status.update(Status.RUNNING)   # notify to the backend that job has
        started
    ...                                 # perform the job
    status.update(Status.SUCCEEDED) # notify to the backend that job has
        finished
    output.co2_footprint = ...          # record the output in the structure
    output.energy_consumption = ...
    output.carbon_intensity = ...

if __name__ == "__main__":
    node = CarbonFootprintNode()      # register node
    node.register_input_cb(input_cb) # select callback
    node.spin()                         # start the job
```

Listing 11: Implementation template for the CFN

B Screenshots of the Module Nodes demonstrator

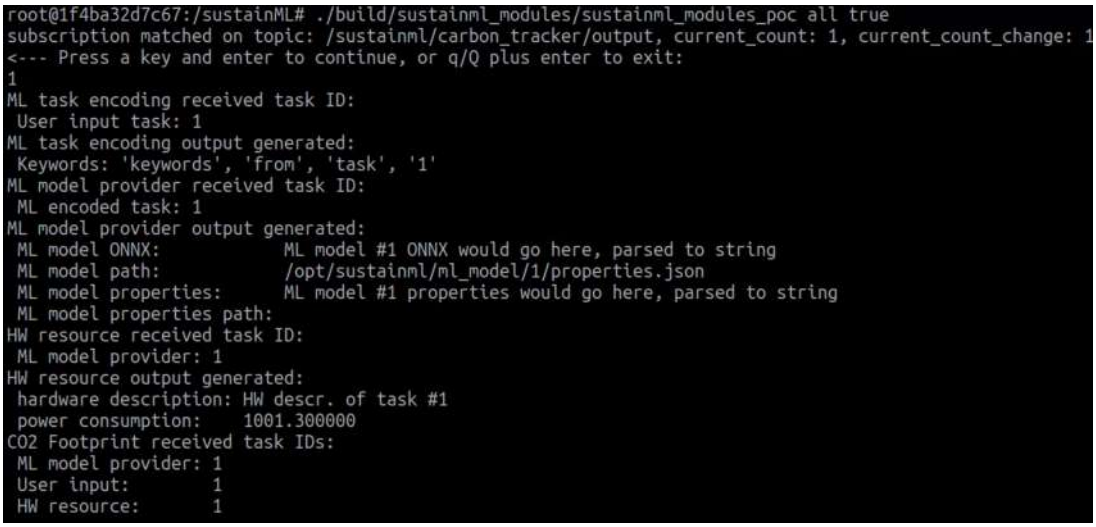

Figure 10: Module Nodes demonstrator in single process scenario

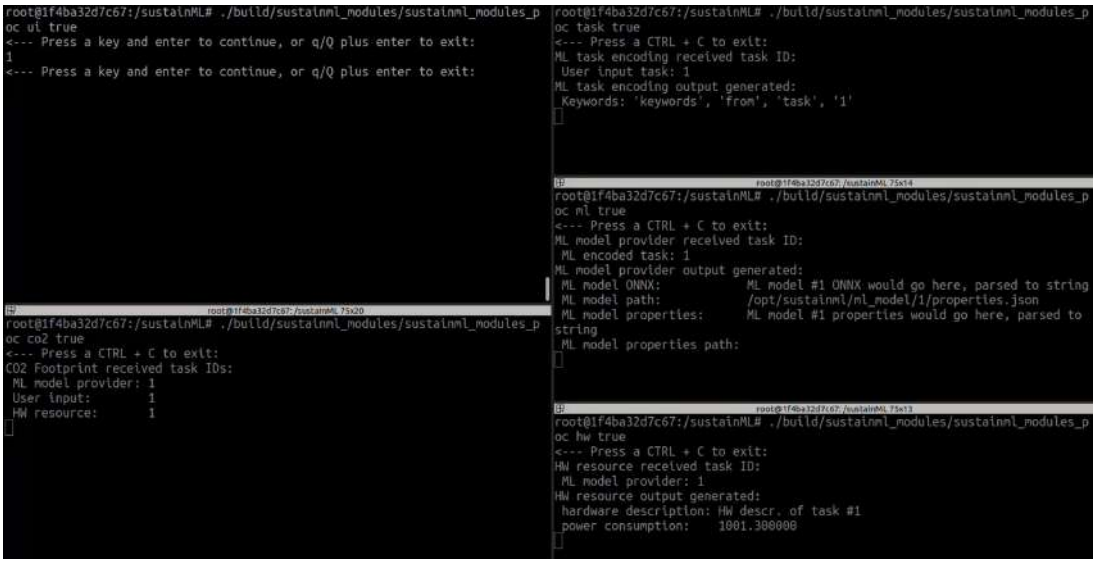

Figure 11: Module Nodes demonstrator in inter-process scenario

References

[1] *eProsima Fast DDS Documentation - What is DDS?* Version 2.11.2. Proyectos y Sistemas de Mantenimiento SL (eProsima), 2023. URL: https://fast-dds.docs.eprosima.com/en/v2.11.2/fastdds/getting_started/definitions.html.

[2] *eProsima Fast DDS Documentation*. Version 2.11.2. eProsima, 2023. URL: http://fast-dds.docs.eprosima.com/en/v2.11.2/.

[3] *eProsima Fast DDS Documentation - Transport Layer*. Version 2.11.2. Proyectos y Sistemas de Mantenimiento SL (eProsima), 2023. URL: https://fast-dds.docs.eprosima.com/en/v2.11.2/fastdds/transport/transport.html.

[4] *Data Distribution Service (DDS)*. Object Management Group, 2015. URL: http://www.omg.org/spec/DDS/1.4/About-DDS/.

[5] *eProsima Fast DDS Documentation - Quality Of Service*. Version 2.11.2. Proyectos y Sistemas de Mantenimiento SL (eProsima), 2023. URL: https://fast-dds.docs.eprosima.com/en/v2.11.2/fastdds/dds_layer/core/policy/standardQosPolicies.html.

[6] *Design Principles and Design Patterns*. Robert C. Martin, 2000.

[7] *Unified Modelling Language*. Version 2.5.1. Object Management Group, 2017. URL: https://www.omg.org/spec/UML/.

[8] Raúl Sánchez-Mateos Lizano. *Data Management Plan (M6)*. Version 1.1. SustainML: Application Aware, Life-Cycle Oriented Model-Hardware Co-Design Framework for Sustainable, Energy Efficient ML Systems. eProsima, 2023.