D2.4 Methodological Framework (b)

Deliverable Id:	**D2.4**
Deliverable name:	**Methodological Framework (b)**
Status:	**Final**
Dissemination level:	**Public**
Due date of deliverable:	**2023-02-28 (M30)**
Actual submission date:	**2023-02-20**
Work package:	**WP2 "Specifications & Methodology"**
Organization name of lead contractor for this deliverable:	Thales Research & Technology
Authors:	Thales: Jérôme Quévremont Modis Consulting: Daniela Angela Parletta University of Oulu: Lauri Lovén, Teemu Leppänen, Huong Nguyen University of Siegen: Daniel Onwuchekwa, Namby Rakotojaona BSC: Sergi Alcaide, Jaume Abella HALTIAN: Matti Vakkuri Offcode: Antti Takaluoma, Janne Rosberg Solver Machine Learning: Leticia Pascual Siemens: Bekim Chilku UPV: Carles Hernández AVL: Christina Schwarz, Milan Zivadinovic, Tomislav Bukić University of Genoa: Igor Bisio, Andrea Sciarrone, Chiara Garibotto ETH Zürich: Frank K. Gürkaynak Rulex Innovation Labs: Enrico Ferrari Università degli Studi di Modena e Reggio Emilia: Gianluca Brilli LKS Next: Iñaki Paz Università degli Studi dell'Aquila: Luigi Pomante PLC2: Ernst Wehlage, Alexander Flick QUALIGON: Juan Manuel Garcia

	AITEK: Stefano Delucchi Ikerlan: Ana Patricia Bautista, Christian Martin
Reviewers:	PROINTEC: David Fernández-Cano INDRA: Martín Rivas AVL: Bernhard Peischl

Abstract:
D2.4 "Methodologic Framework (b)" introduces a methodological framework specification. It is presented as a compositional workflow, which introduces the interactions of FRACTAL building blocks towards the integration of the FRACTAL computing node and the use cases. Following this global picture, the deliverable focuses on methodologies for several important topics for the project: AI and safe autonomous decisions, safety design, integration of FRACTAL platforms and security analysis.

Contents

Acknowledgement

This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 877056. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Italy, Austria, Germany, Finland and Switzerland.

1 History

Version	Date	Modification reason	Modified by
0.0	2022-10-21	Agreed template, starting from D2.2 v1.0	Thales
0.1	2023-01-20	Complete version	Authors
0.2	2023-02-01	Reviewed version	Reviewers
0.3	2023-02-20	Reviewers’ remarks solved	Authors
1.0	2023-02-20	Final clean-up, delivered version	Thales

Table 1 – Document history

To cope with the high number of contributors, this document has been edited online. The Microsoft Sharepoint solution has been selected to keep information under EU legislation. This solution offers a reduced feature set compared to a “regular” Word editor. For instance, we have not been able to build a table of references and have instead used footnotes.

2 Summary

The task T2.2 is described in FRACTAL DoA as "For the integration of the FRACTAL platform in an industrial environment, an important aspect is to describe (1) how it should be used and (2) how this usage helps to qualifications and certification of products developed using it, including safety-critical products."

Accordingly, D2.4 "Methodologic Framework (b)" introduces a methodological framework specification. It is presented as a compositional workflow, which introduces the interactions of FRACTAL building blocks towards the integration of the FRACTAL computing node and the use cases.

Following this global picture, the deliverable focuses on methodologies for several important topics for the project:

- AI and safe autonomous decisions
- Safety design
- Integration of FRACTAL platforms
- Security analysis

A list of abbreviations is available at the end of the document.

3 Introduction

This deliverable provides a methodological framework specification. The aim of this framework is to identify the key enabling technologies with their supporting methodology and tools.

Development of the FRACTAL node product and its sub-products (or sub-components) are supported by this workflow.

This document is structured into 5 main chapters. First the overall workflow of the project is presented in chapter 4. Then sub-workflows/methodologies for "AI and Safe Autonomous Decisions" are presented in chapter 5, for the "Safety Design" in chapter 6, for the "integration of FRACTAL platforms" in chapter 7. The development of the risk analysis in WP4 is presented in chapter 8.

Different WPs provide or use components for composition. This requires prior alignment of what is provided and what is expected: functional boundaries, interfaces, and other necessary information. Collaboration within the workflow includes handover of these artifacts between stakeholders and workflow steps while ensuring, managing, and maintaining composability during the workflow.

4 Workflow of the project

The **FRACTAL** platform specifications (see deliverables D2.1 and D2.3 for more details)define what the framework should provide (i.e. components, tools, methodology and workflow) to be key enabling technologies (KETs) for the FRACTAL node. To limit the scope and better target the domains considered in the project, the use cases specifications are used as inputs. The final specifications should also consider the extensibility and usage of the framework in related domains of application.

4.1 Capturing Requirements

Figure 1 shows the overall workflow used to capture requirements during the **FRACTAL** project. First, the different demonstrators have been specified (i.e. scenarios, features, and functional and non-functional requirements) in "Integration and verification" (WP7) and "Case Studies" (WP8). The demos' requirements have been then analysed to get a unified list of requirements and key enabling technologies that are being developed during the project. These have been identified into "Specifications & Methodology" (WP2). Third, the identified key technologies have been characterized and decomposed into the technical work packages: the "node architecture & building blocks" (WP3), "Safety, security and low power techniques" (WP4), "AI & safe autonomous decisions" (WP5), and "CPS communications framework" (WP6). At the same time WP4, WP5 and WP6 have decomposed requirements received from WP7 and WP8 and have realized what they needed to further extend the list of requirements for WP3.

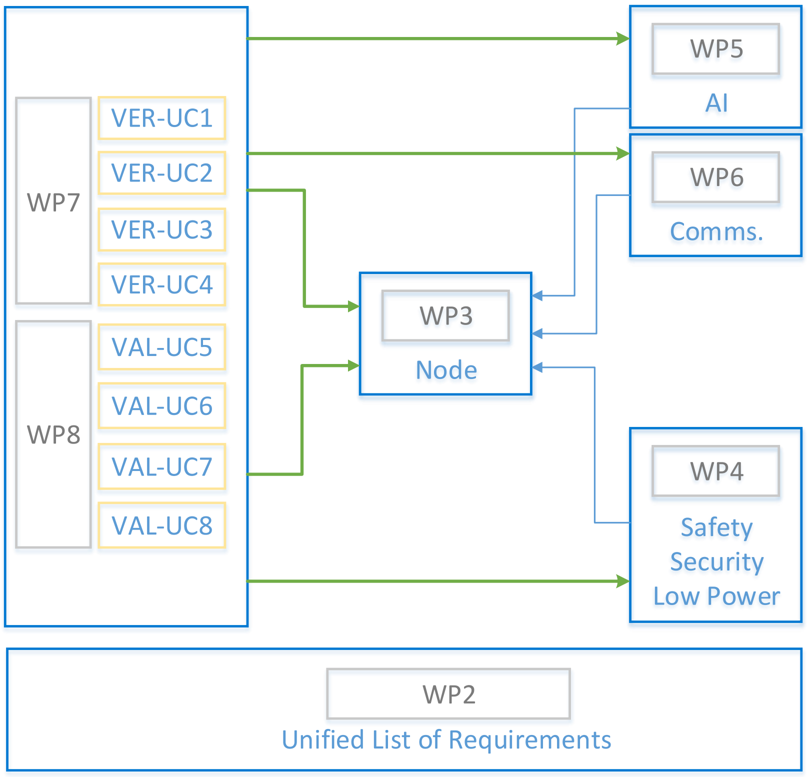

Figure 1 – Requirements workflow in FRACTAL project

Upon reception of external requirements, each Work Package works its internal requirement list and develop its "products".

4.2 FRACTAL Features

There are many definitions of Feature, coming from the Software Product Line (SPL) community. Using the most notable definition from FODA[1]: ***A "feature" is defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a system".*** SPLs refer to engineering techniques for creating a portfolio of similar systems from a shared set of assets (components) using a **common means of production**. This portfolio of similar systems is called a **Family of Products**.

Using the analogy of a car, the family of that car model is composed of the different versions and variations of the given car model (style, colour, engine, etc.) produced in the car product line. When a user wants to buy a car, (s)he has to select among the distinct features available. For instance, he may choose the version, the engine, the colour, the air conditioning system, the transmission, etc. Sometimes some selections force the selection of another feature. For instance, a given car style can

[1] Kang, K.C. and Cohen, S.G. and Hess, J.A. and Novak, W.E. and Peterson, A.S., "Feature-oriented domain analysis (FODA) feasibility study", Technical Report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University, November 1990

force a given engine and restrict the number of other options available. That is, features can have distinct values and also have relationships/dependencies. Even, some features can be optional (e.g. sunroof). Here, the user must be able to understand the features in a model to be able to configure a valid product. The car seller helps the user to understand the given options to configure a valid car to be produced.

In the context of FRACTAL, a **FRACTAL Feature is a distinguishing characteristic of FRACTAL, visible to users that will configure FRACTAL for their use cases**. FRACTAL Features can describe functional characteristics (hardware, software), non-functional characteristics (performance or other criteria) or even other parameters (cost, max weight). A **Feature Model** is a ***hierarchical diagram*** that visually depicts the features of a solution in ***groups of increasing levels of detail***. Feature Models are compact representations of all the products in a Family in terms of features and provide a great way to summarize the features that will be included in a solution and how they are related in a simple visual manner.

Summarizing, the **FRACTAL Feature Model** describes the variability in **FRACTAL's family of products** in terms of **FRACTAL Features**. Use Cases represent variants of FRACTAL products, this is, specific product realizations.

Based on the concepts described in FRACTAL Document of Agreement and the needs and requirements captured from the Use Cases, a FRACTAL Feature Model has been built (and is maintained alive as those needs evolve). This FRACTAL Feature Model has been introduced in D2.3 as a tree representation with increasing levels of detail. On the first level, FRACTAL high level features are defined (see figure 2).

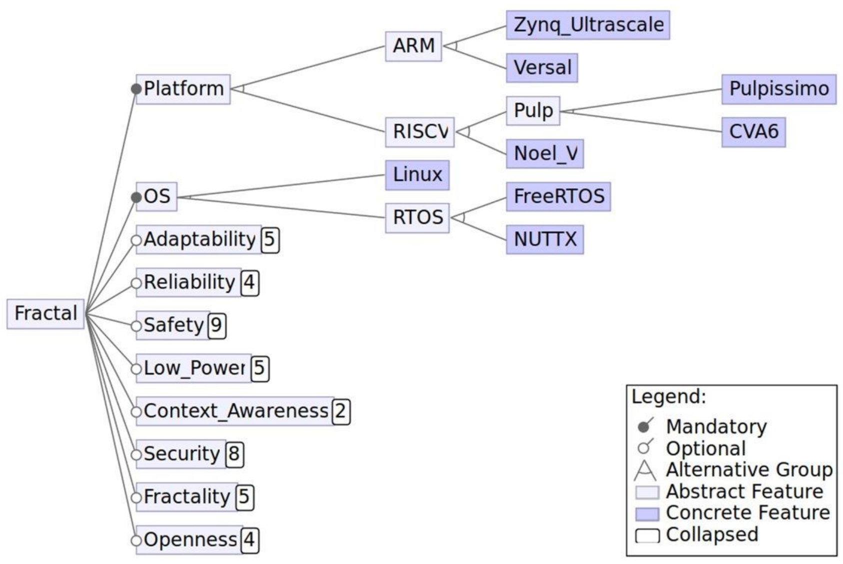

Figure 2 – FRACTAL High Level Features

A short definition of these features follows:

- Adaptability: Ability to adapt system behaviour to surrounding scenario.
- Context-awareness: Ability to detect and react to context and surroundings.
- Openness: FRACTAL Core should be Open to be used, and mainly based on open-source licenses.
- Safety: Be resilient to internal failures.
- Security: Be resilient to external attacks and menaces.
- Reliability: The system should behave consistently well.
- Low Power: Ability to support low power scenarios.
- Fractality: Ability to support fractal configurations, which can be seen as the organization of nodes in distinct layers and the communication/connectivity involved.

FRACTAL Use Case requirements (see Use Case requirements defined in D2.3) have been used to guide the feature model construction. For instance, based on the requirement:

REQ_UC4_06 - the edge node shall provide dedicated HW accelerator to process the CNN layers (Yolo) of AI inference.

The following features have been defined:

- Adaptability -> AI -> HW -> AI ML Accelerator
- Adaptability -> AI -> SW -> Inference -> Location -> Edge Node
- Adaptability -> AI -> SW -> Learning/Training -> Algorithm -> Yolo
- Adaptability -> AI -> SW -> Learning/Training -> Algorithm -> CNN

Using another example from other UC, based on the requirement:

REQ_UC6_14 - The edge node shall acquire images from at least one HD camera.

The following features have been defined:

- Context Awareness -> Sensors -> HD Camera

The lower the level of the feature in the tree, the higher the detail. For instance, figure 3 partially presents the Adaptability feature subtree:

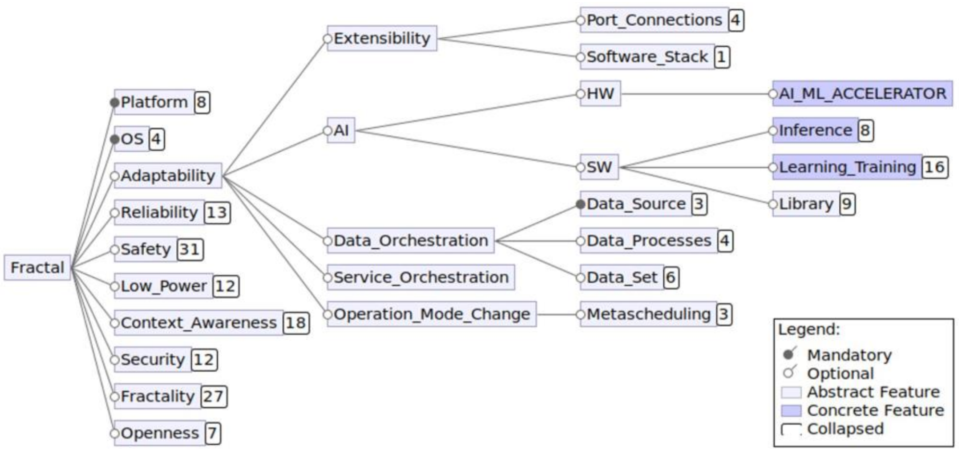

Figure 3 – FRACTAL Adaptability partial subtree

Adaptability is considered as the ability to adapt system behaviour. System behaviour can be adapted by (1) extending the system through port connections or extending the software stack; (2) introducing AI that deals with adaptation; (3) orchestrating the data on which adaptation is based; (4) orchestrating the services executed on FRACTAL nodes or (5) changing the operation mode of the system based on a certain detected condition.

4.3 FRACTAL Product Composition

Technical Work Packages (WP3-6) implement the shared set of components (building blocks) required to produce the Use Case variants. WP3 provides the node definition and platform. WP4, WP5 and WP6 add their products on top of the results of WP3 (see figure 4). In particular, WP3 provides the hardware and software components (aka *primitives*) and platform nodes that allow building complex services and properties. On top of this in WP4, WP5 and WP6 aim on smartly combining them in accordance with specific UC goals. The first global integration attempt is performed in the verification phase WP7 that helps fine tune all products and then everything should be ready to the final validation in WP8.

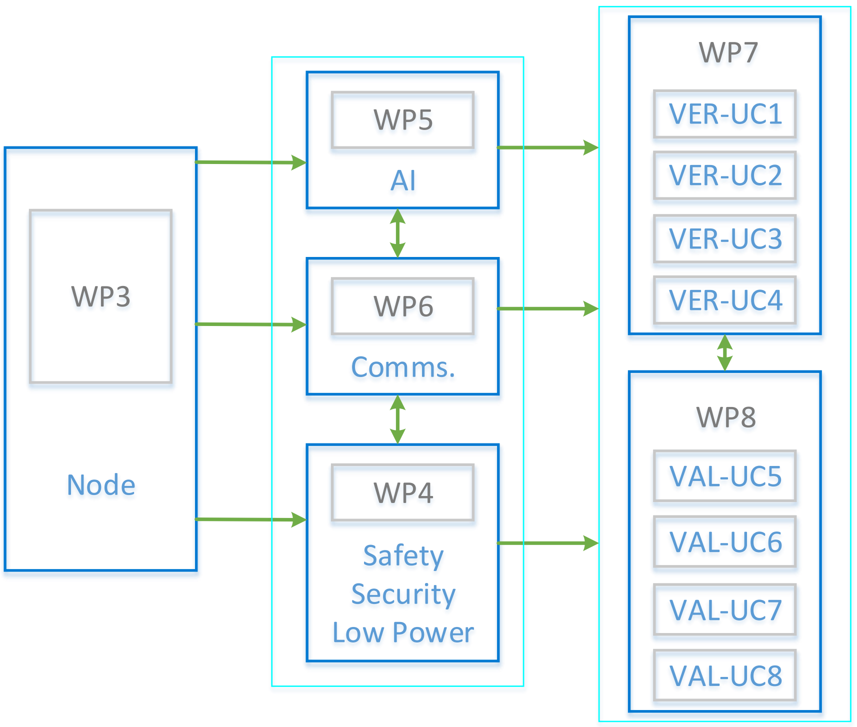

Figure 4 – FRACTAL products composition workflow

To ease FRACTAL Product composition, the components developed in the technical WPs have been associated with the corresponding features it implements (defined in D2.3) and in which architectural layer it operates. In FRACTAL, three main architectural layers have been identified:

- Node Layer: References node base infrastructure layer, be it a device (low-level SW & HW) or the cloud itself.
- Orchestration Layer: Executed over the node layer, provides a standardized execution environment to FRACTAL Nodes as containers and defines the FRACTAL Distributed System (how nodes and concrete containerized services are added/removed/configured). Containers provide a base standard for the development and distribution of application logic to FRACTAL Nodes.
- Application Layer: Specific application logic is built as services in containers and deployed over nodes. Services make use of the specific Node Layer offerings if needed (e.g., HW Acceleration, Diverse redundancy libraries, etc.). This layer includes data ingestion, federated data collection, data pre-processing, edge ML API, etc.

Use Cases have selected the specific features they are interested in from the FRACTAL Feature model. The specific valid selection of features by a Use Case is called **Bill of Features**.

By doing so, through the component relationship, Use Cases select the list of FRACTAL components they are interested in. Finally, a FRACTAL Production plan will be in place, which given a Platform (selected on the specific bill of features) and the set of selected components, defines how to build the FRACTAL product (see figure 5).

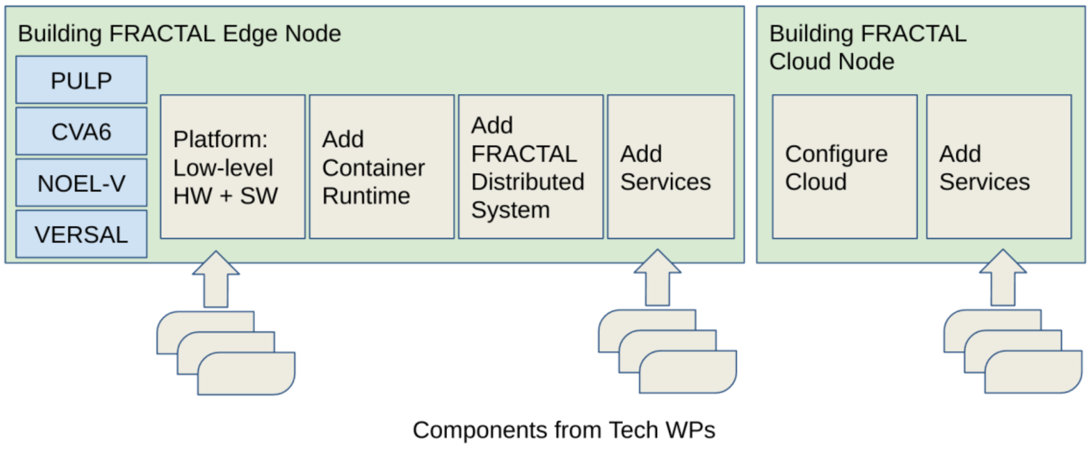
Building FRACTAL Edge Node
PULP
CVA6
NOEL-V
VERSAL
Platform: Low-level HW + SW
Add Container Runtime
Add FRACTAL Distributed System
Add Services
Building FRACTAL Cloud Node
Configure Cloud
Add Services
Components from Tech WPs

Figure 5 – FRACTAL production plan

Section 7.1 "Operational integration" presents this composition workflow exemplified on a given Use Case.

5 Methodology and workflow for AI and safe autonomous decision

This chapter looks at the AI building blocks in WP5, focusing especially on the FRACTAL autonomous decision framework (section 5.1), the FRACTAL service architecture and middleware (section 5.2), and a common framework and format for FRACTAL inference models (section 5.3). Further, the chapter looks at AI functionality required by the use cases, concentrating in particular on UC2 (Automotive air path control) (section 5.4) and UC6 (Intelligent Totem) (section 5.5).

5.1 AI and autonomous decision framework

The FRACTAL AI framework aims at integrating the AI functionalities to allow advanced prediction capabilities in the FRACTAL node. The framework has been developed keeping in mind the requirements from the use cases but ensuring the correct degree of generality and flexibility. The FRACTAL node must be able to take decisions in an autonomous way. This means that while data are measured from the field, the FRACTAL node must be able to analyse the data and make all the inferences needed to make the right decision in real time, without the need of communicating with external server. Notice that, when talking about AI, we need always to distinguish between the training phase and the inferencing phase. In the first one, historical data are used to train the AI models while in the second one the already built models are used to make predictions and decisions about the new incoming data.

Since in the studied use cases the system's behaviour is not changing with high frequency, it is not necessary to perform the training phase very often and, above all, to have a response quickly. Usually, the training phase is done once or is updated on a regular basis if the system is supposed to undergo some changes in the behaviour. So, two different scenarios are considered:

- For the first tests, pretrained models are deployed on the edge node. These models are either already available in literature or obtained training the data coming from the first tests.
- The training is done periodically. In this case (see figure 6) data are automatically transferred to a cloud service (via 5G or Wi-Fi connection) where they are stored in a database and used for building a model. Task 5.2 and the related deliverables developed the cloud framework for treating data and performing AI processing. The model is then sent back to the device where it is used for subsequent elaborations. Notice that in this case the network is not a bottleneck since FRACTAL node can continue its processing even without the generation and the transferal of new model. So, if for some reason the connection is not guaranteed the system can go on working.

Since it could be unfeasible (due to huge amount of produced data) or inappropriate (for example for security or privacy reasons) to send all the data in FRACTAL also the possibility of using Federated Learning approaches has been studied. According to Federated Learning also the training phase is done on edge nodes. Each node uses the available data to build a local model. All the models can be further processed by a supervisor to improve their quality, but, nonetheless, each node is independent and can also use the model generated on its own data.

In any case the inferencing phase is performed on the FRACTAL node since the decisions should be taken very quickly, which is not in line with sending data over the internet. Moreover, the system should work also without connection, so the node must be autonomous in taking decisions.

So, the FRACTAL node is equipped with a layer able to perform some basic pre-processing operations on the data and an AI module able to use already built models to make decisions about new incoming data. The elaboration must be very quick because in some use cases, decisions are expected to take place within 100 ms. Moreover, the module is able to do some aggregation on the data; in this way, data could be sent in an easier way to a cloud service where the AI models could be updated.

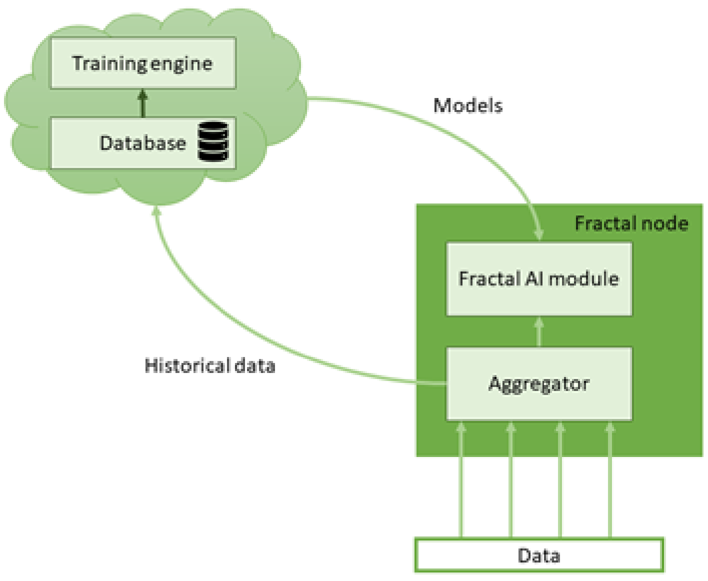

Figure 6 – Functioning of the online training for the FRACTAL node

The AI model that is implemented in the FRACTAL node according to the needs of the use cases:

- Video analysis
- Audio stream analysis

- Supervised and unsupervised learning
- Time series analysis

5.1.1 Video analysis

Video Content Analysis deals with the extraction of information from images and video. Such information can be used for further processing done in "high-level" applications that collect and correlate data from heterogeneous sensors. Modern Video Content Analysis (VCA) systems are based on AI approaches: after a proper training phase, they can understand how to analyse and detect relevant information inside images and video streams.

The first step needed to build an AI-based VCA system is to define exactly the kind of information to be recognized and detected. There are different possible tasks:

- Classification: assignment of images to different classes or groups, according to their content.
- Tagging (or labelling): it is a classification task more complex with respect to the previous one; multiple labels can be associated to an image as the VCA is able to recognize multiple "scenarios" or "concepts" in it. A practical example can be useful to explain differences: classification can distinguish between images collected indoor or outdoor while tagging can add multiple labels to the same images like outdoor, city, road intersection or indoor, house, bedroom for example.
- Detection/Segmentation: both previous tasks are focused on detecting the presence of one or more reference target (object, person, a scenario etc.) in an image or a video. Detection and segmentation can also infer the location of such target(s) inside the image. In particular:
    - Detection process generates as output a bounding box that surround each target detected,
    - Segmentation process detects the shape of each target as it performs a pixel-based decision (i.e., each pixel can be assigned to background or to a specific target).

During FRACTAL, and in particular in Use Case 6, detection and/or segmentation tasks play a crucial role, as they are extremely useful to the FRACTAL node to understand its surrounding environment. As a matter of fact, targets detection and localization enable further processing like for example counting the number of persons and or reference objects within a specific area. Also target tracking is a quite relevant function in the scope of FRACTAL, in particular for those applications related to safety and security control.

VCA, as any other AI system, is based on a training phase in which the system learns to detect and recognize a target. In general, the training can be supervised or unsupervised. For the VCA, training is supervised, meaning that the system learns how to perform the detection through a set of labelled (annotated) images. Basically, such annotated dataset (e.g., a large number of images with a person inside) is the input for the training phase; after the training phase, the output is a model that, at runtime, can be used to perform the task (in this case people detection).

Such training phase has a relevant impact on VCA performance, that is dramatically affected by the training set characteristics. In particular, this dataset must include an adequate number of images, it must be accurate, meaning that images used for training should be similar to those analysed at runtime, but at the same time, it must be representative of all the different alternatives that are possible at runtime.

Overfitting and underfitting are two typical problems of AI-based VCA system: the model achieves poor classification/detection performance after training. In particular, overfitting means that there are too many parameters in the model and a high variability of the classification. Therefore, the model is too complex and sensitive to training dataset (high variance). On contrary, underfitting means that there are few parameters in the model and a high classification discrepancy (high bias). In other words, underfitting can be explained as the model is too simple and therefore unable to provide good results during prediction; overfitting is when the model is too good to be true, as it performs very well analysing training data but it is completely unable to be generalized and therefore achieve very poor results on runtime prediction.

As previously said, the training phase and therefore the training set play a crucial role in the VCA system performance. A limited amount of data for training, in this case annotated images, is one of the worst scenarios concerning an AI-based VCA system. Data augmentation is a useful technique to overcomes this problem. It consists in a manipulation of available images in order to increase artificially the dimension of the dataset. For example, some pictures in the database can be transformed by rotating, flipping them or by modify colour, contrast or brightness. Recently has been emerging a new approach for data augmentation based on the use of Generative Adversarial Network (GAN). Such particular type of Neural Network is here mentioned as it is quite relevant in the AI framework for the VCA. Moreover, it is taken in deep consideration during the next phases of FRACTAL, in particular in WP5. Several details are included in the deliverable D5.1 and D5.3.

Most methods for object detection are essentially based on CNN. Convolutional Neural Network is a specialized kind of neural network for processing data with a known grid-like topology, like images or time series. For our purpose we focus particularly on images. These are the most common methods:

Regional Proposals: These methods are organised in two phases: in the first phase, a convolutional NN is used to identify the regions where a certain target object could be found; in the second phase, a more precise NN is trained to confirm or not the presence of the object. Relatively heavy approach, but one that can achieve very high accuracies. The first network does a coarse skimming while the second can be very precise in identifying real targets. The most used NN architectures are: R-CNN , Fast R-CNN, Faster R-CNN, and cascade R-CNN.

Single Shot Detector (SSD): It presents an object detection model using a single deep neural network combining regional proposals and feature extraction. A set of default boxes over different aspect ratios and scales is used and applied to the feature maps. As these feature maps are computed by passing an image through an image classification network, the feature extraction for the bounding boxes can be extracted in a single step. Scores are generated for each object category in every of the default bounding boxes. In order to better fit the ground truth boxes adjustment offsets are calculated for each box.

You Only Look Once (YOLO): It is a fast real-time multi-object detection algorithm that utilizes a single convolutional network for object detection. Unlike other object detection algorithms that sweep the image bit by bit, the algorithm takes the whole image and reframes the object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. In more details, it takes an image input, splits it up on a SxS grid, passes it through a neural network to create bounding boxes and class predictions to determine the final detection output. It is first trained multiple instances over an entire dataset before being tested on real-life images and video. The advantage of these networks is that they are (relatively) fast, work on a single frame and can recognise objects of very different scales.

Typically CNN is a sequence of different types of layers. The input to the CNN is a pixel array of an input image. The intermediate results obtained are a set of feature maps. The outputs obtained are in the form of conditional probability for a given set of inputs. Highest probability for a choice depicts the confidence of the network in that output.

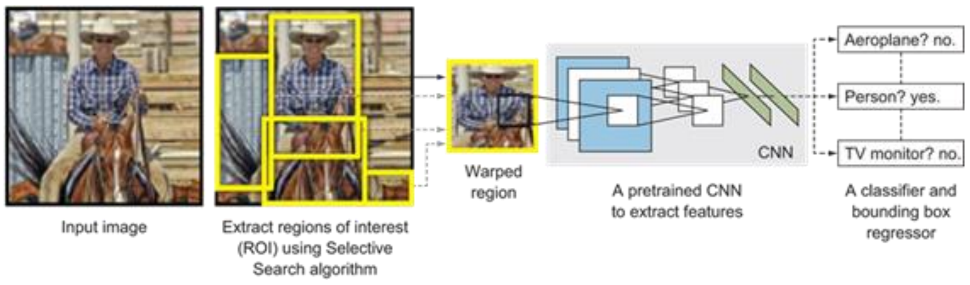
Input image
Extract regions of interest (ROI) using Selective Search algorithm
Warped region
CNN
A pretrained CNN to extract features
Aeroplane? no.
Person? yes.
TV monitor? no.
A classifier and bounding box regressor

Figure 7 – Object detection example using CNN approach

The CNN-based object detection requires a high volume of parameters and calculations to extract features in the image and make predictions about the objects. To meet this requirement, many researchers adopt high-performance devices, such as graphics processing units (GPUs), central processing units (CPUs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs), to build onboard real-time systems. It is difficult for CPUs to take full advantage of parallel computing to meet real-time processing. They are rarely used as CNN implementation platforms. While the computing performance of GPUs is fantastic, the high-power consumption hinders their usage in the onboard system with limited resources and power budgets. FPGAs and ASICs have the advantages of high performance and energy efficiency. Thus, taking these low-power devices as hardware implementation platforms for CNNs has become a research hotspot. However, ASICs require a long development period and high costs to be designed. Therefore, owing to the advantages of the short development period, energy efficiency, and reconfigurability, FPGAs are the ideal implementation platforms for CNNs. Therefore, if the FPGA can be used to implement the target detection algorithm, the hardware volume can be greatly reduced, and the implementation speed is fast, the flexibility is high, and the power consumption is low.

5.1.2 Supervised and unsupervised learning

Besides the analysis of video and audio streams, other AI tasks could be implemented in the FRACTAL AI framework. In this subsection a short overview of more traditional machine learning applications that could be used to analyse data deriving from use cases. The methods introduced here deal with structured data, i.e. that can be organized in tables. Usually, a distinction is done between supervised and unsupervised methods. Supervised methods assume that data are somehow labelled either in a natural way or because some human has labelled them manually. This label is the target of supervised methods since they aim at building a model able to predict the value of the target starting from a set of inputs. According to the type of target, classification or regression problems could be defined. Multilayer Perceptron (MLP), Support Vector Machine (SVM), Logistic Regression (LR), Decision Trees (DT) are methods for supervised learning.

On the other hand, in unsupervised problems, no target variable is available and the goal is to find information within the data. For example, some unsupervised approaches are:

- Clustering, aimed at organizing data in homogeneous groups.
- Outlier detection, that are devoted at finding configurations that deviate from standard behaviour.
- One-class classification, whose goal is finding a classification model when only data of one class are available. For example, data about failures could not be yet available in historical data in a predictive maintenance application.
- Sequence Analysis and Anomaly Detection, aimed at analysing time sequences to detect frequent or uncommon patterns that could be related to regular or anomalous behaviours.

In general, these approaches could be used in association with the analysis of audio and video streams. As a matter of fact, the analysis of video and audio could generate features that can be used as an input of supervised or unsupervised tasks.

Moreover, it is worth noting that some techniques belonging to this class also allow the generation of intelligible models, according to the Explainable AI (XAI) paradigm, enabling applications where the understandability is a key feature.

5.2 Distributed AI and the AI services in the middleware

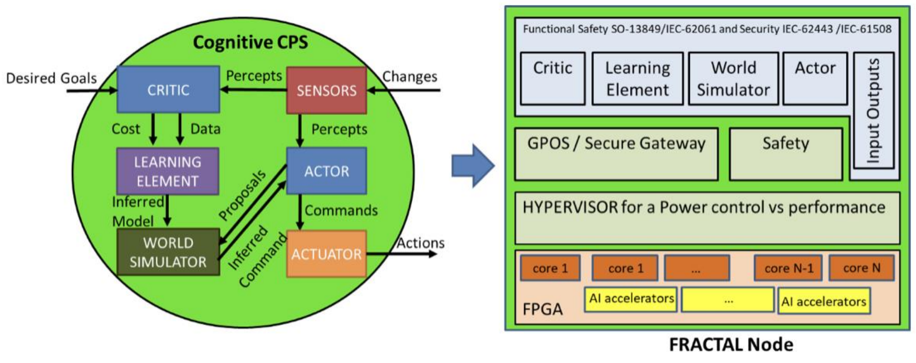

Figure 8 – FRACTAL cognitive agent and the corresponding node architecture.

Building upon the architecture described in section 5.1, the FRACTAL framework supports common AI workflows related to distributed learning and decision-making. In more detail, a FRACTAL cognitive agent (figure 8) interacts with other system components, devices and data sources through the services provided by the platform middleware. In its operation, it uses and exports both internal and external interfaces for connecting to its sensors and actuators and to external services and data sources, for operational control (e.g., setting the agent goals), and for sharing its results, knowledge and data. The agent architecture is internally composed of software components with varying roles, such as modules for interactions, decision making, implementing and evaluating the selected actions and interactions.

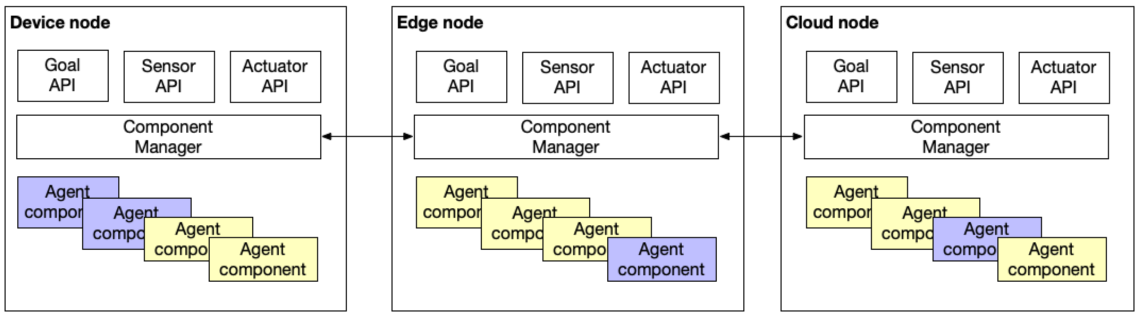

Figure 9 – Agent components and APIs. Active component shown in blue.

To facilitate orchestration of the operations towards optimal Quality of Service and performance, the run-time deployment of each software component must be decided. To this end, the agent architecture must consider the application requirements. For example, some processing-heavy components (e.g., the *learning element*) could be run on a nearby edge node or on the cloud, while others may run on-device (see figure 9). As a result, the edge-cloud framework provides component online deployment (including offloading and migration), management and monitoring functionalities.

On the one hand, the framework enhances the adaptivity of the individual nodes in response to the dynamics of the environment, the state of the platform, and application requirements, by allowing control of its communication-computation trade-off (e.g. latency vs. data transmission vs. computational load). On the other hand, such a framework increases operational complexity significantly and introduces a need for (partially) autonomous decision-making by the components. Figure 10 further illustrates the related horizontal offloads and vertical migrations.

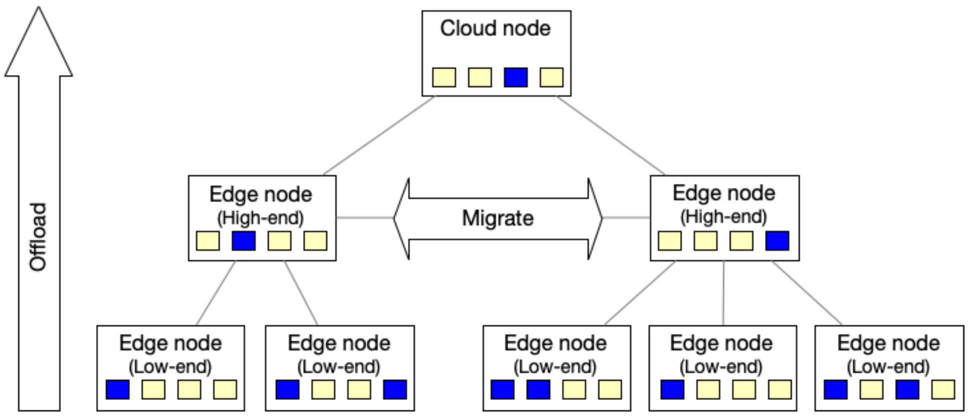

Figure 10 – Hierarchical component transfer framework.

For those of the agent components, which encompass learning and decision-making elements, the framework requires data and knowledge sharing and collaboration across the platform. For example, some components may employ a *federated learning* schema (see figure 11), where a number of edge nodes, coordinated by a cloud node, collaboratively build a shared understanding (e.g., a model). The architecture, and largely the framework, must thus allow the online sharing of data, results and knowledge, all the while monitoring and evaluating the operation and environments of each component taking part in the learning.

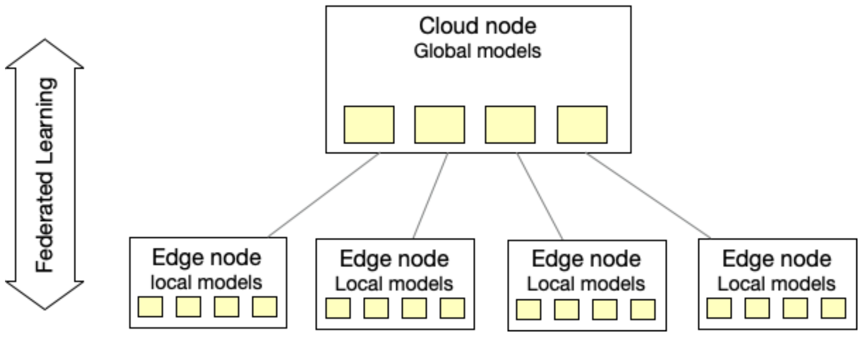

Figure 11 – Federated leaning in a FRACTAL system.

Further, while an agent makes partially autonomous decisions, a number of use cases also call for collaboration and co-operation (e.g. through swarm intelligence) as distributed decision-making, targeting operational efficiency and effectiveness of the system. To facilitate such a multi-agent system, the agent must support sharing of both data and knowledge, leading to complex interactions between the agents and other system components (figure 12). Such an integrated architecture is also required to support the top-down/bottom-up control in the operational framework, further specified in T5.4.

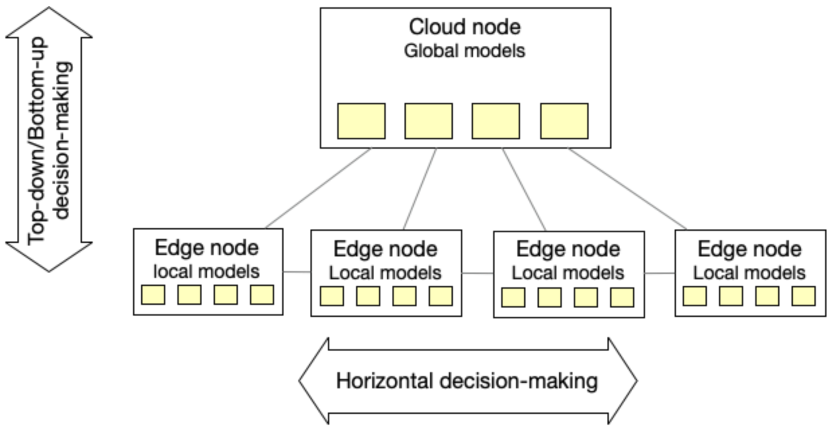

Figure 12 – Distributed decision-making in a FRACTAL system.

In addition to placing and allocating the application components, the distributed AI framework must also cater for the flow of data used to build those models. Indeed, the middleware has to provide interfaces for application components to publish data sets or streams as well as to subscribe to them, while keeping track of the placement of both: if a component is offloaded or migrated, its pub/sub endpoint has to move accordingly.

Moreover, taking into consideration the dynamic nature of the edge-cloud computing continuum, at times the streams have to be routed anew due to link capacity changes between components. Finally, the data streams themselves can also change in volume and velocity, requiring a reconsideration of the stream routing.

Optimally, the framework thus considers simultaneously the communication, computation as well as data-related resources while orchestrating the distribution of AI models in the computing continuum.

5.3 Use of LEDEL

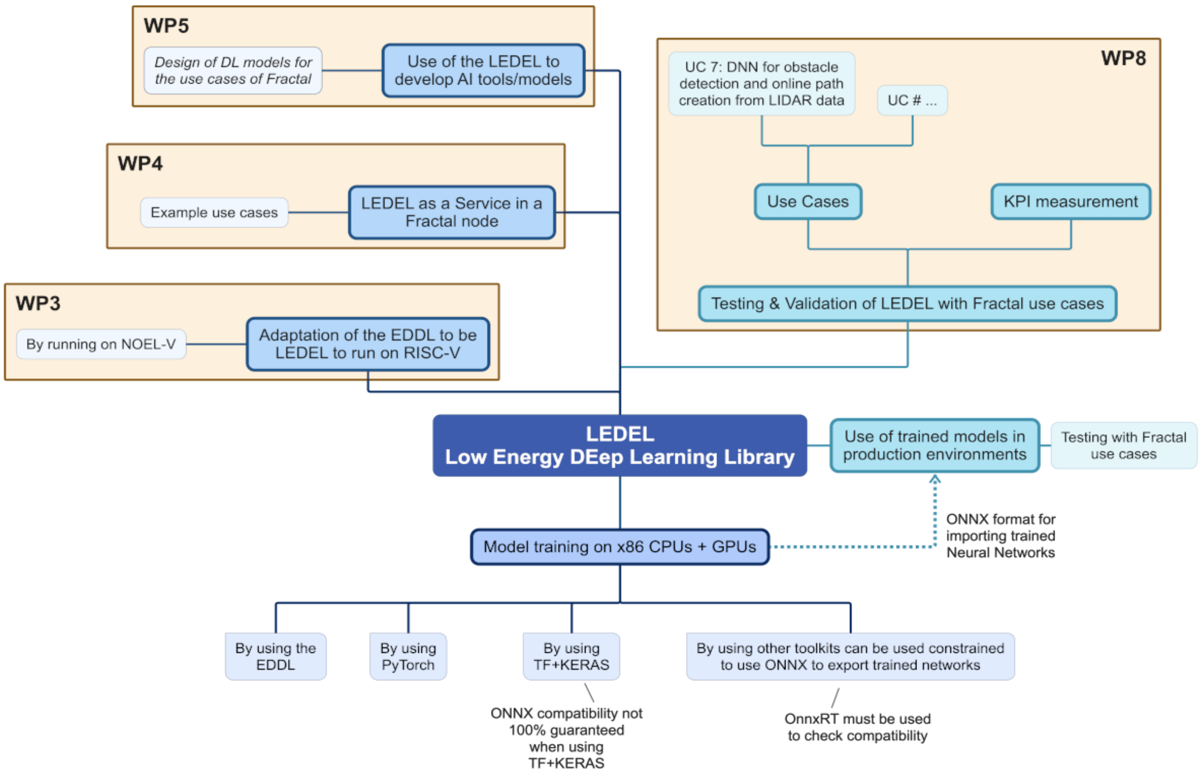
WP5
Design of DL models for the use cases of Fractal
Use of the LEDEL to develop AI tools/models
WP4
Example use cases
LEDEL as a Service in a Fractal node
WP3
By running on NOEL-V
Adaptation of the EDDL to be LEDEL to run on RISC-V
WP8
UC 7: DNN for obstacle detection and online path creation from LIDAR data
UC # ...
Use Cases
KPI measurement
Testing & Validation of LEDEL with Fractal use cases
LEDEL
Low Energy DEep Learning Library
Use of trained models in production environments
Testing with Fractal use cases
ONNX format for importing trained Neural Networks
Model training on x86 CPUs + GPUs
By using the EDDL
By using PyTorch
By using TF+KERAS
By using other toolkits can be used constrained to use ONNX to export trained networks
ONNX compatibility not 100% guaranteed when using TF+KERAS
OnnxRT must be used to check compatibility

Figure 13 – Development stages of LEDEL in FRACTAL

EDDL (https://github.com/deephealthproject/eddl, European Distributed Deep Learning Library) is a Deep Learning (DL) toolkit designed and developed to provide support to design and train Deep Neural Networks (DNNs) on single computer nodes and on hybrid HPC + Big Data computing architectures. EDDL is ready to leverage hardware accelerators, such as GPUs and many-core CPUs. It also uses the ONNX standard format (https://onnx.ai) to import/export DNNs. Thus, trained DNNs can be used on production environments to infer/predict. LEDEL (Low Energy EDDL) is the adaptation of the EDDL to run on Low Energy hardware. Trained DNNs using the EDDL are easily employed to infer/predict in production environments working with the LEDEL. In figure 13 we can observe different tools provided by EDDL.

LEDEL is about to be ready to run on Edge computing hardware that has been developed in FRACTAL. This hardware has limited computing capabilities but is more powerful than existing low energy hardware so far. Thus, thanks to LEDEL, not only is it possible to make decisions based on simple conditions, but more complex decisions based on indicators provided by more sophisticated algorithms running on the edge are feasible to be made as well. As an example, we are able to evolve from simple presence detectors to decide whether to switch on lights, to much more complex scenarios where the number of people in a room and the distance between individuals is computed every few seconds in order to adapt room conditions such as light intensity, cooling and heating, only switching on the required lights, etc.

The tasks involving the LEDEL in the FRACTAL project are shown in figure 13 and defined as follows.

- EDDL adaptation to run on RISC-V based hardware in the NOEL-V processor model proposed (WP3, T3.5).
- In order to check the correct execution of the LEDEL as a software service in a FRACTAL node, an example of a use case from the DeepHealth project has been adapted and tested for its correct behaviour (WP4, T4.1). To emulate the FRACTAL node, a simulated environment has been built using an/the Isar of Siemens [https://github.com/siemens/isar-riscv]. The model has been trained outside the node, imported into an ONNX file and transferred into the emulated node. Finally, the net from the ONNX has been loaded into a program implemented using LEDEL and checked for its proper behaviour (check D4.2 for further detail)
- The work carried out in WP5 corresponds to the investigation of different models that could be used in a FRACTAL node, considering its dependencies, like OpenCV to execute TinyYolo, or different techniques and implementation approaches, such as network configuration, and layers and changing among different learning techniques like reinforcement learning.
- Finally, SML is providing support to check the integration of the LEDEL in the FRACTAL platform and guarantying a good performance and a proper behaviour in UC7.

In conclusion, LEDEL is almost ready, as a service offered in a FRACTAL node, to be able to perform more complex calculations (i.e., to run more sophisticated algorithms). The goal is to develop an API that provides deep learning functionalities that are devoted to face the deployment on low energy computing infrastructures. LEDEL is already accessible in the FRACTAL project repository (https://github.com/project-fractal/WP3/tree/main/Components/WP3T35-03%20LEDEL) and a very complete tutorial with examples and docker files are presented in order to make it easy for FRACTAL partners to use it in their use cases.

5.4 Advanced control strategies in the automotive domain

Existing automotive control strategies are fully reliant on model-based control strategies. These techniques imply a high calibration effort and the ability to perform self-learning through observations is very limited. This use case therefore contributes to integrate the environmental influences and changes as a fundamental part of the system, among other benefits, like potentially increased product quality and increased efficiency for the development of customized controllers. The FRACTAL nodes are crucial to the implementation of this use case.

The FRACTAL framework shall demonstrate the following objectives:

- Inference of data-driven models aimed at improved energy efficiency and reduction of environmental pollutants on the FRACTAL node
- Online self-adaptation algorithms of the initial state model, to react to variations regarding the environment and different driver behaviours
- Freeze frame data collection and connection to the cloud for re-training purposes

In figure 14 an overview of the planned (implementation) interactive environment schema can be seen. Three different model operations can be differentiated. Firstly, the model inference of the initial state AI-based model, as a result from the model development process, which is implemented to replace the conventional control strategy. Secondly, the model adaption during the vehicle in-use phase, to cover the model blind spots coming from the limitations of the input data used for model training and to have the possibility to adjust to vehicle specific parameters. The adaptation algorithm would compare the output of the model inference with the measured data and in case of a deviation, learn and preserve the additional information. Thirdly, a cloud connection is established to utilize the potentials of crowdsourcing and the access to extensive information from other drivers/vehicles, with the overall target to optimize the control strategy from many different aspects (e.g., changes in environmental conditions, variability, coverage of different operation modes, etc.). Since this task comprises the use of big data, a computationally heavy training infrastructure is needed and would therefore require cloud computing for the execution.

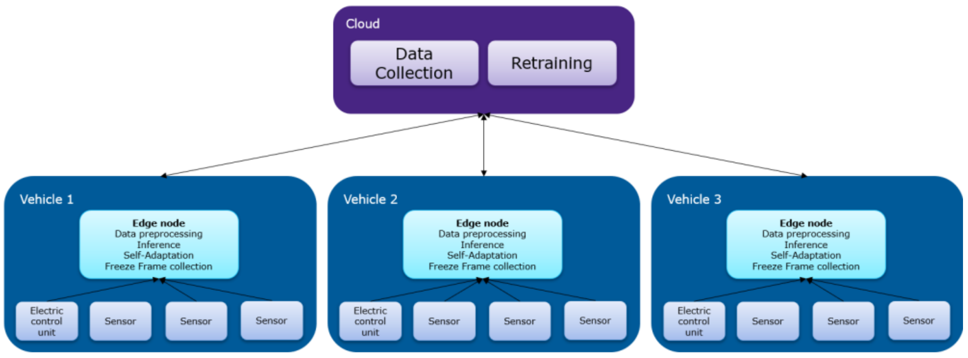

Figure 14 – Overview of FRACTAL framework needed for Automotive implementation

5.5 Image classification

Modern deep learning architectures achieves state-of-the-art performances on vision tasks such as image classification, tracking and segmentation; nevertheless, the adoption of these approaches into critical domains is still limited. Two key aspects limiting the diffusion of these approaches are the limited interpretability of the learned models and the lack of specific guidelines to efficiently and effectively perform the optimization of their hyper-parameters.

Within the scope of the FRACTAL project the methodological approach taken for this task is twofold. From one hand, it pursues the adoption of state-of-the-art explainers, specific algorithms that provide human-interpretable explanations about the model predictions. One of such explainers that has gained much attention for vision tasks is LIME (Local Interpretable Model-Agnostic Explanations). Roughly speaking, LIME takes as input the learned deep network and the image to classify; as output, it provides the classification of the image along with the image enriched with patches: the patches highlight the areas that most contributed to the classification of the image. Since the patches are visual, they are, usually, very-well interpretable by human users.

On the other hand, the idea of incorporating the optimization of important pre-processing hyper-parameters into the training and the test phases is promoted. Both these aspects are demonstrated in the context of a clinical application, diabetic retinopathy detection.

More specifically, suppose that the processing pipeline use some pre-processing algorithm **A** to perform a given step of pre-processing. Suppose also that **A** requires as input some parameter *p* besides the image to process; and suppose also that *p* must be specified on a per-image basis. A typical approach to this problem, is to perform some expensive grid-search procedure, image-per-image, to find each time the best value for *p* according to some mathematical criterion. This results in an extremely time-consuming approach that is not even necessary correlated with good predictive performances. Indeed, the existing optimality criteria for **A** might be not connected with its role in the prediction pipeline. To overcome these limitations, it is possible to *learn* the best parameters to use at training time. In particular, the predictive model is trained jointly with another deep learning architecture – in an end-to-end fashion – whose sole role is, given an image **x**, to predict the best value *p(***x***)* for the parameter of **A** to use. The image is then pre-processed using **A** parametrized by p(**x**). This approach has two advantages, first at test time, given an image, predicting the parameter to use for **A** requires only a feedforward step which is typically faster than performing a grid search. Second, the parameter of **A** is optimized to obtains the best predictive accuracy, as one would expect. In order to learn both networks, the models are connected together through a differentiable layer (e.g., the Gumbel soft-max activation) which allows for the backpropagation of the gradients. In this way, both models are optimized to reach the best predictive accuracy.

Diabetic retinopathy (DR) is a dangerous disease that can lead ultimately to blindness. A timely diagnosis of this pathology is fundamental to limit its consequence on the patient. Recent research studies have shown the potential benefits of adopting deep learning methods to support the specialist in the analysis process. In particular, those methods have achieved nearly human performances in the task of classifying certain levels of retinopathy.

However, such algorithms mostly act as black boxes taking as input a high-resolution image of the eye fundus, and producing as outputs a prediction on the level (if present) of retinopathy affecting the patient. The black-box nature of these methods limits their usability as support tools from the specialist that has no clues to understand the factors that caused the model to output a certain prediction. Moreover, updating and tuning these algorithms for the specific cameras is a timely consuming and poorly defined process due to the lack of well-defined mathematical objectives. An important example of this issue is given by the image equalization algorithms, that turned out to be fundamental to obtains good performances in this task. These algorithms, most prominently CLAHE, require the user to provide one or more parameters that control their behaviour. Unfortunately, it is not clear how one should choose these parameters, and current solutions are often based on expensive greed search procedures. The proposed methodology is used in the context of a methodological study described in D5.6. In figure 15, a DR detector enriched with the LIME explainer is shown. An image is given an input to the processing pipeline and, and explanation about its prediction is built according by LIME. In figure 16 instead the end-to-end approach to the image equalization problem is shown. Here the image is given in input to the first network, that predict the parameter of CLAHE (the image equalization algorithm); then the image is equalized by CLAHE with the predicted parameter, and finally is given in input to the second network for the DR detection.

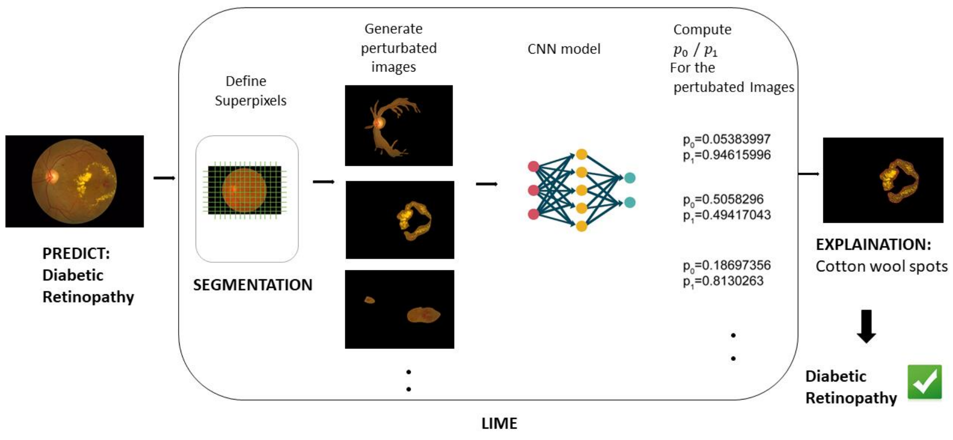
PREDICT: Diabetic Retinopathy
Define Superpixels
SEGMENTATION
Generate perturbated images
CNN model
Compute p_0 / p_1 For the pertubated Images
p_0=0.05383997 p_1=0.94615996
p_0=0.5058296 p_1=0.49417043
p_0=0.18697356 p_1=0.8130263
LIME
EXPLAINATION: Cotton wool spots
Diabetic Retinopathy

Figure 15 – Image classification with explanations, an example in the context of DR detection

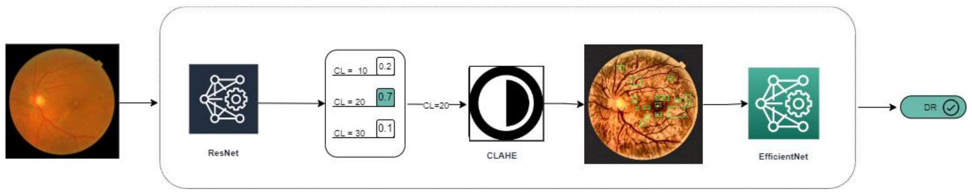

Figure 16 – End-to-end learning for hyper-parameters optimization, and example in the context of DR detection.

6 Measures and strategies for safety design

To build the safety concept for a FRACTAL system, a number of safety-related properties need to be built bottom up. Generally, these properties relate to the ability of the system and its components to detect, manage and/or tolerate faults and or behaviour beyond the intended functionality. This chapter covers those aspects for the different components in the form of safety measures and strategies. In particular, approaches are presented to deal with faults for the computation of an application (diverse redundancy), timing interference across applications, communication across nodes, AI-related processes and time-triggered NoC.

Note for the reader: the D2.2 sections "6.1 Guidance to define a methodology for a safety-related development (ISO61508) tailored to an R&D cooperative project" and "6.3 Certification of VERSAL platform and related applications" have moved to D2.5 "Safety-critical applications regulations compliance handbook".

6.1 Diverse redundancy

Fault detection in a FRACTAL system architecture can be efficiently provided bottom-up. This implies that appropriate safety measures need to be deployed along with computation means. Safety measures for the highest criticality functionalities (e.g. ASIL C and D) include some form of diverse redundancy so that a single fault, despite affecting all redundant elements, cannot lead to exactly the same error, which might escape detection.

Common Cause Failures (CCFs), in automotive terminology, are those failures caused by a single fault that makes safety measures, such as redundancy, ineffective. For instance, two identical cores executing the same task redundantly fully synchronized have the same state, and upon a common fault (e.g. a voltage drop) could experience the same error. To avoid CCFs, safety-related systems implement redundancy with some form of diversity so that the risk of experiencing identical errors in redundant elements is residual. In the case of storage, this is usually achieved using Error Detection Codes, in the case of communications using Cyclic Redundancy Check codes, and in the case of computation, using some form of lock-stepped execution where two or more identical cores execute identical software but with some staggering (i.e. time shift) so that cores' state is sufficiently diverse.

Therefore, platforms intended to run functionalities with high integrity requirements need some form of lock-stepping support. This can be achieved with tight lock-stepping, where only one redundant core is visible at software level and the others can only work in lock-step mode, as done, for instance, by the Infineon AURIX microcontrollers for the automotive domain.

Tight lock-stepping at core level can be implemented with different flavours. For instance, one could compare the outcome of each instruction or even each pipeline stage every cycle[2]. However, the most effective solution has been shown to compare only off-core activity (e.g. requests visible in the interconnect) to reduce the overheads while avoiding any visible impact due to errors.

While this solution is highly effective to attain diverse redundancy, it is inflexible since it does not allow using the cores independently to run different tasks. An alternative is using light lock-stepping, where redundancy is created and managed at software level, and independent cores are used enforcing staggering with SW-only means, as illustrated in figure 17.

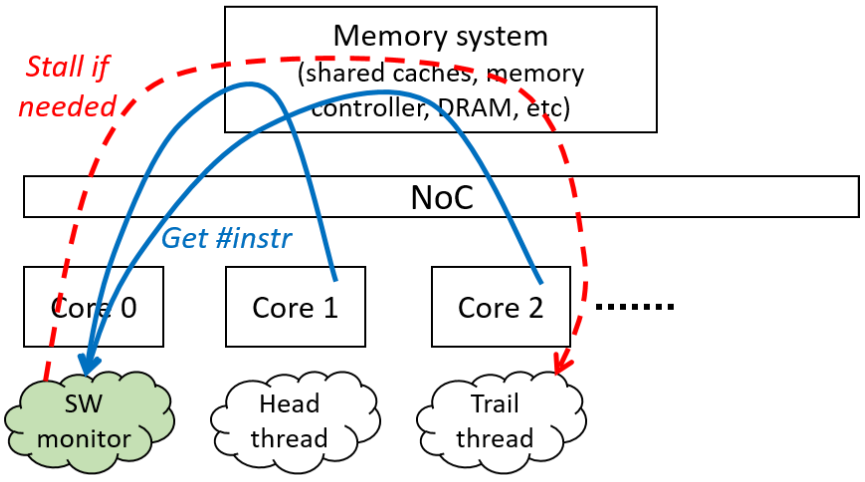

Figure 17 – Schematic of the SW-only lightweight lock-step.

SW-only diverse redundancy builds upon the creation of redundant processes at software level, so that both of them receive the same – redundant – input data and return their results for comparison in a *safe* CPU. Without loss of generality, this discussion focuses on dual-core lock-step, which is the common approach in many domains, including automotive.

The methodology builds on the use of three threads (see figure 17):

- **Monitor**. The monitor is the one spawning the redundant computation threads (head and trail), and monitoring and enforcing staggering between them.
- **Head thread**. The head thread executes the functionality without any specific control for the sake of achieving diverse redundancy.
- **Trail thread**. The trail thread executes the functionality redundantly with some staggering (delay) with respect to the head thread. Therefore, if the staggering at any point is too short, it is stalled for a while.

[2] A more detailed analysis of the different alternatives can be found here: C. Hernandez and J. Abella, "Timely Error Detection for Effective Recovery in Light-Lock-step Automotive Systems," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 11, pp. 1718-1729, Nov. 2015, doi: 10.1109/TCAD.2015.2434958.

Since the monitor lacks any form of redundancy, it needs to run in a native lock-stepped core, which may be in a separate microcontroller, or may also be in the same microcontroller. The monitor spawns the head and trail threads into two other cores, which do not implement tight lock-stepping support. Then, the monitor performs the following steps periodically every **`Tcheck`** cycles:

1. Collects the instructions executed count (**`IC`**) from both threads, so **`IChead`** and **`ICtrail`**.
2. Computes the difference between both counters and compares it against a threshold **`Istagger`**.
    a. If the head thread is sufficiently ahead from the trail one, then both threads continue the execution. Formally, execution continues normally if (**`IChead - ICtrail) > Istagger`**.
    b. Else, if the trail thread is too close to the head one, then the monitor stalls the trail thread during the next monitoring period (so **`Tcheck`** cycles). Formally, the trail thread is stalled if (**`IChead - ICtrail) <= Istagger`**.
3. Finally, the monitor sleeps until the remaining time until elapsing **`Tcheck`** cycles.

This mechanism guarantees that the trail thread cannot catch up with the head thread as long as the instruction threshold **`Istagger`** is large enough so that, even if the head thread stalls completely and the trail thread executes at the maximum possible speed, the trail thread cannot catch up with the head thread since one monitoring period until the next one. This implies that executing a **`Istagger`** instructions must require at least **`Tcheck`** plus the time to detect that the current staggering is too low and stalling the trail thread.

In the context of automotive systems, as well as in other domains, as long as faults are managed short enough after their occurrence, only single-point faults need being considered. Single-point faults are those faults caused by a single event even if such event can lead to multiple errors. This is explicitly covered in ISO 26262, part 5, clause 7.4.3.3: "*Evidence of the effectiveness of safety mechanisms to avoid single-point faults shall be made available*".

Hence, the following scenarios are possible for single-point faults:

- The fault affects one or the two computing threads (head and/or trail). Since they have intrinsic diversity due to the proposed mechanism, if the fault leads to errors, those will be detected by the monitor and properly managed with the corresponding system level recovery solution implemented (e.g., re-execution).
- The fault affects only the monitor. The monitor runs on a native lock-stepping computing core, so, if the fault leads to an error, it will be detected and managed as needed at system level. In this case, the execution of the threads is correct. Hence, one may simply use the result of the head thread since it

will match the one of the trail thread even if staggering is not preserved since the fault only affected the monitor. Since the fault affecting the monitor will be detected if it leads to an error, we can just compare the result of both threads to validate that they match. Note that the single-fault assumption guarantees that this later comparison is fault free.

- The fault affects all components (monitor, head thread and tail thread). In this case, the monitor detects it failed, and the outcomes of both threads mismatch. Hence, we can proceed as in the previous case but, in this case, the later comparison will detect a mismatch across the outputs of the head and trail thread, so system recovery actions can be taken then to manage the impact of the fault on the monitor as well as on the computing threads.

Overall, if a single-point fault leads to errors in any of the computing components or the monitor, those errors will be properly detected and reported so that system level recovery actions can be taken.

6.2 Management of timing interference

Safety-related systems, as part of their development process, must adhere to specific verification and validation (V&V) processes, and include safety measures to deal with random hardware faults that may occur in the field. This is achieved by means of observability and controllability knobs. Those knobs have already been deployed in mono-core SoCs. However, MPSoCs pose a number of challenges related to performance interference in the hardware shared resources across tasks running in different cores. Such interference must be properly accounted for and mitigated to meet performance-related safety requirements in line with safety standards specifications. In the context of FRACTAL, since such interference emanates at node level, that is the scope where safety measures and V&V means have been deployed for efficiency reasons[3] (e.g., to stop excessive interference immediately rather than awaiting for software layers to react too late when further interferences has occurred).

Commercial safety-related MPSoCs, such as, for instance, the Infineon AURIX processor family, include limited features to monitor and mitigate multicore timing interference. Typically, those MPSoCs include some form of Statistics Unit (SU) capable of tracking access counts, number of instructions executed of different types, as well as aggregated stall cycles in some buffers and queues. Those MPSoCs may also include powerful debug facilities such as Aurora, Lauterbach and CodeWarrior,

[3] J. Cardona, C. Hernandez, J. Abella and F. J. Cazorla, "Maximum-Contention Control Unit (MCCU): Resource Access Count and Contention Time Enforcement," *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy, 2019, pp. 710-715, doi: 10.23919/DATE.2019.8715155.

which allow collecting detailed information about events in the MPSoC. Unfortunately, those SUs and debug facilities have some limitations:

- It is generally hard – if at all possible – discriminating how much interference each core creates on each other core since stall cycles, if available, are provided in an aggregated manner.
- The MPSoC may lack means to exercise control over multicore timing interference.
- Despite debug facilities are powerful, they are normally only usable during the development process given the fact that they require specific equipment and software for their use. Hence, safety measures to be used during operation cannot exploit those facilities.

Specific SUs can be tailored to deal with those challenges and meet the requirements of safety-related systems in terms of multicore timing interference, both during V&V and during operation. In particular, a timing-interference aware SU would allow implementing appropriate safety measures if it provides the following features:

1. Interference monitoring. Monitoring the amount of delay experienced by each task due to interference, and being able to identify the particular task causing such interference is particularly relevant to optimize the system during design and verification, and to diagnose deadline overruns in the field.
2. Interference quota enforcing. Those deadline overruns can be simply avoided if means are deployed to set quotas on how much interference each task can create on each other task.

Those features allow implementing both, reactive mechanisms and diagnostics by monitoring interference, as well as proactive mechanisms based on quotas to avoid effects due to undesired timing behaviour. Overall, those mechanisms are the basis to mitigate interference so that freedom from interference, as requested in some safety standards (e.g., ISO26262), is achieved to a sufficient extent (e.g., upper-bounding potential interference).

Timing interference generally manifests in the on-chip interconnection networks since all data sent to/from the cores needs to traverse those networks. Hence, SUs for timing interference monitoring and control must likely be deployed along with those interconnection networks. Since those interconnects use standard interfaces, such as AMBA AHB/ACE/AXI, SUs may be easier to integrate and reuse if they comply with those protocols.

In the context of FRACTAL, if the MPSoC includes a centralized interconnect managing all on-chip traffic from cores (e.g., a bus), a single SU may suffice. Alternatively, if distributed interconnects are deployed (e.g. NoCs) so that no single location drives all core-related traffic, we may need to set up multiple SUs to be able to monitor and control all on-chip traffic. Those SUs may further require some post-processing of the information collected in a distributed manner to gain global knowledge about how multicore timing interference is occurring. Alternatively, a single SU can be deployed as long as it can monitor multiple NoCs simultaneously.

It is also common experiencing multicore interference in other components apart from the NoC, such as, for instance, shared caches and shared memory controllers. If such interference does not manifest in the form of backpressure in the NoC, just observing the NoC may not be enough to accurately measure all interference. In this case, the SU may need being extended to consider additional events such as core stall signals, and queue occupancy signals in caches and memory controllers. In any case, simple logic monitoring those signals in the SU can determine what core is delaying what other cores so that interference can be monitored and ascribed with sufficiently high precision.

6.3 Reliable communication

FRACTAL systems rely on inter-node and intra-node communications. To satisfy safety requirements, on-chip communication networks or interconnects may rely on state of the practice protection mechanisms such as Error Correction Codes (ECC), replication or monitoring.

ECC is usually required for data links. The choice for the actual ECC implementation, (e.g. SECDED or parity) depends on the implementation costs and the actual safety need. For instance, fail-safe applications or applications with long fault tolerance time intervals can use simple parity bits while more stringent applications requiring fail-operational capabilities may also require redundancy bits with correction capabilities such as SECDED (single error correction double error detection).

Hardware replication. Hardware replication (e.g. duplication) might be also needed in addition to ECC to achieve the highest integrity levels. Implementing redundancy at the core level with software only means is not sufficient to protect the system from random faults. Actually, without a hardware mechanism that prevents cores and other IO to write to arbitrary locations faults affecting a core or I/O can introduce undesired memory corruptions. Thus, the software redundancy mechanism of the NOEL-V FRACTAL node requires having a memory protection unit (MMU) like the one present in the NOEL-V. Unfortunately, accelerators are not covered by the MMU and memory isolation properties are preserved in our platform using two physically different memory spaces for the accelerators and cores.

Hardware monitors. Shall be integrated with the interconnect to allow built-in self-test (BIST) and error reporting capabilities. This is usually a requirement that expands all integrity levels and it is aimed to ensure safe operation during the lifetime of an application. In the context of highly complex SoC platforms (e.g. multicores) these monitors should also have the capability to track software timing information. Existing monitors in the NOEL-V FRACTAL platform ensure that requests targeting a specific device (e.g. memory) have the permission rights to access such device. Different IDs are assigned to specific masters/tasks in the SoC so that safety-relevant and non-safety relevant worlds can be built. Such scheme can also be used to by the AXI and AHB traffic monitors to propagate contention information to the edge SafeSU. Additionally, specific software-based tests are used to test the functionality of the different SoC devices. These tests are currently used for validation purposes but can also be leveraged for runtime verification.

For off-chip communications, applications safety shall be supported only if the underlying off-chip communication is based on a safety-critical infrastructure (e.g. Time-sensitive networking). However, other than such safety-critical communication infrastructure, a fail-safe mechanism is also considered for the FRACTAL communication. That is, the node needs to remain safe (i.e. transition to a safe state) if the communication with other nodes is down. However, data integrity and authenticity conveyed by the FRACTAL communication are secure features that are also important for safety, not to make any decision based on flawed information.

6.4 Artificial intelligence

The utilization of artificial intelligence (AI) techniques in the context of FRACTAL applications with safety requirements are constrained. In general, the utilization of AI does not involve tasks of the system mapped to a certain criticality level. For instance, following ISO26262 nomenclature AI tasks are restricted to QM (quality management) functionalities only. That is, functions without safety relevance and that only require standard Quality Management processes. Otherwise, safety violations in absence of HW or SW failures that are a consequence of limitations in the algorithms, sensors or actuators must be also considered as defined in the ISO/DIS 21448: Safety of Intended Functionality (SOTIF) automotive sector standard.

Even in this constrained scenario, the deployment of AI techniques in platforms that involve critical functionalities (e.g. mixed criticality systems) is challenging. The reason is that sharing the same CPU between critical and non-critical tasks requires ensuring the existence of partitioning mechanisms (provided by Linux or a hypervisor) to allow AI related tasks not to interfere with critical tasks (neither at the functional nor at the temporal level). The NOEL-V FRACTAL node supports the H extension for hypervisors and partitioning hypervisors like Jailhouse or Xtratum have been successfully ported to this platform in the context of the SELENE and DE-RISC H2020 projects. Additionally, since accelerators are memory intensive the execution time of critical tasks co-running in the platform can be significantly affected by such timing interference. To solve this, we have leveraged the end-to-end QoS capabilities of this platform that allow us to establish contention quotas that also account for the contention caused by the accelerators. Additionally, if FRACTAL platforms make use of AI outputs to improve non-safety properties like quality of service (QoS), energy or availability, the AI system shall provide a measure of uncertainty in all the decisions to get an estimate of the QoS that these systems based on AI are able to provide. The robustness of a redundant AI acceleration scheme was analysed in D4.4.

In summary, for FRACTAL systems, the safety of the platform should not depend on the output of the artificial intelligence algorithms. Involving AI in the decisions that may affect the safety of an application is however mandatory to achieve fully autonomous certified applications. Unfortunately, this technology, while being currently a hot research topic, is not expected to be available in the near future for critical industrial domains.

6.5 Time-triggered NoC

Safe-on-chip communication is required when transferring critical messages within an SoC. An NoC is a good candidate for on-chip communication when the number of processors increases within the chip since it provides high bandwidth and scalability and supports parallel communication. For critical safety systems, the on-chip communication needs to provide deterministic communication in order to satisfy the real-time system's behaviour.

Time-Triggered communication is a good candidate for safety-critical multi-core architecture. In Time-Triggered communication, the Network Interface of the NoC has the capability to inject the messages from the core using a predefined schedule that is computed offline. Having this full control of injection time for each message within the NoC reduces the risk of message collision and reduces the jitter within the NoC.

In summary, for FRACTAL on-chip communication, using a Time-Triggered NoC helps the systems to meet the deadlines when message exchange is needed between cores since it provides deterministic communication and low jitter compared to a generic NoC.

6.5.1 Adaptability techniques for Time-Triggered NoC

The static allocation in time-triggered systems offers significant benefits for the safety arguments of dependable systems, such as avoiding resource conflict within the NoC. However, adaptation is a key factor for fault recovery in FRACTAL on-chip communication.

Adaptation using multiple schedules allows the NoC to adapt based on every predefined scenario (for example, when a fault occurs in one particular core, the systems should be able to switch from one schedule to another and remove the faulty core within the new schedule, to avoid the propagation of the fault within the chip, and allow the entire systems to work even if in the presence of fault).

The TTNoC performs three operations before switching from one schedule to another when a context event occurs, as described in the following items:

- Context monitor that monitors the presence of faults reported in each core.
- Context Agreement that propagates the local fault to all cores within the chip, so each core has the state knowledge of all cores.

- Adaptation: After the context agreement, each processing element within the NoC has the state of each core. The state of all the cores in the multicore on a chip is known as a global context event. Based on the global context event, the adaptive scheduler generates a new schedule, and this new schedule is used to reconfigure the NoC in order to maintain the safe operation of the multicore on a chip.

In summary, adaptability in the TTNoC can provide fault tolerance and increased reliability for various workloads and applications running on the chip. The ability to switch between schedules allows for greater flexibility and adaptability to different context events and situations that may arise, regardless of the specific application being executed. Overall, this can lead to more efficient and robust operation of the chip and the applications running on it.

7 Methods for the integration on FRACTAL platforms

This chapter introduces methods that are used to integrate use cases on FRACTAL platforms in WP7 and WP8. Some sections are specific to a use case.

WP7 integrates the FRACTAL building blocks, technologies and methods. The FRACTAL features cognitive awareness, cloud connectivity, adaptability, security and low power guide the integration and verification activities. The verification task assesses the metrics regarding these quality attributes and set up a coordination task to gather and analyses the insights and KPIs of the various FRACTAL nodes.

WP8 aims at homogenizing the requirements from use case providers to provide a unique framework for dealing with all the real-world situations considered in the use cases in term of Fractality. The objective of this work package is to define the needs of the use cases by identifying domain-specific requirements and assess at the end of the project whether the technical objectives of the project have been reached.

Moreover, WP8 ensures that developments in the field of:

- Artificial Intelligence and Autonomous Decisions
- Safety and security insurance
- Low power and high performance on the edge implementation

To do this, a coordination process between use case providers, technology providers and integrators are needed to ensure that the development activities are in the direction of fulfilling the pillars of the project.

7.1 Operational integration

To be able to fulfil FRACTAL’s goals in the market, FRACTAL must be viewed as a minimal viable product and this project must define the characteristics of that product and how it is built and delivered. Chapter 4 has introduced FRACTAL Features and their relationship with FRACTAL components developed by the distinct partners and that UCs use.

On the other side, the distinct use cases provide a focus on the variability that a FRACTAL node must support. By the end of the project, a FRACTAL node should be be constructed to support the FRACTAL Features that each use case (WPs 7 and 8) brings to light. Operational Integration refers to the process that must be undertaken to build the FRACTAL node that an end user wants. This section presents the specific workflow for a given use case as an example (the use case is simplified for a better understanding of the process).

The first thing that a given use case needs to do is to identify the features the use case is interested in. As an example, a given use case pretends to build a device that will be deployed on a machine on the field. The machine is fully powered and thus the device will not have problems with the power supply. The device will control how the machine works and adapt its behaviour based on the current context where the machine is working. The device will receive data from the machine and its sensors, analyse it and if needed adapt the behaviour. The device will be able to communicate to the cloud. The adaptation engine will be AI based and the models will be trained on the cloud and updated in the device when required.

Let's start with feature selection:

- PLATFORM -> VERSAL will be selected as the more powerful platform used in FRACTAL.
- FRACTALITY -> ORCHESTRATION -> DATA as data will be orchestrated from the devices into the cloud.
- ADAPTABILITY -> DATA ORCHESTRATION -> PROCESSES -> INGESTION as data ingestion processes will apply on the cloud.
- ADAPTABILITY -> DATA ORCHESTRATION -> DATA SET -> STORAGE as the data will need to be stored on the cloud.
- ADAPTABILITY -> DATA ORCHESTRATION -> DATA SET -> VERSION CONTROL as the data sets for AI training will be versioned.
- FRACTALITY -> ORCHESTRATION -> MODEL as the built models will be orchestrated and version controlled.
- ADAPTABILITY -> AI -> HW -> AI/ML ACCELERATOR as the acceleration from the Versal board will be used.
- ADAPTABILITY -> AI -> SW -> INFERENCE -> MODEL FORMAT -> VERSAL as the Versal model for deployment will be used.
- FRACTALITY -> ORCHESTRATION -> SERVICES as the analysis engine will be containerized.
- ADAPTABILITY -> DATA ORCHESTRATION -> DATA PROCESSES -> TRANSFORMATION since the data will be analysed and transformed on the device.
- ...

Notice that in this process the selection of a FRACTAL feature can force the selection/removal of another feature. For instance, a feature of Low Power consumption may exclude the Versal Platform (feature dependencies and constraints).

In order to be able to select the features, understanding of the goal of the features is required, thus a FRACTAL-skilled interpreter should be in place (as a seller to describe option when buying a car).

The bill of features corresponds to all the features that have been selected. It may also be represented with a pruned tree (see figure 18).

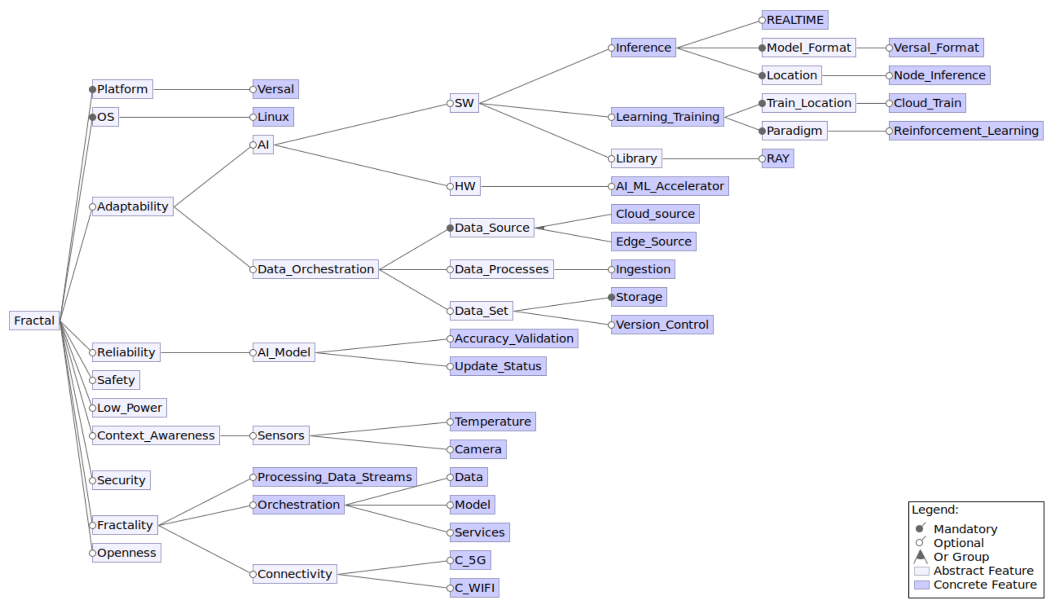

Figure 18 – Sample bill of features

Given a valid "Bill-of-Features", the process to build the FRACTAL node should start. First, a set of components has been selected following the FRACTAL Feature – component map defined in D2.3 and also crossing it with the selected platform. For the given example, the following components have been selected:

Component ID	Component Name	Related Features
WP3T34-03	Versal Model deployment layer	ADAPTABILITY -> AI -> HW -> AI /ML ACCELERATOR
WP5T52-01-01	Cloud Data Ingestion Service	ADAPTABILITY -> DATA ORCHESTRATION -> PROCESSES -> INGESTION
WP5T52-02-01	Cloud Raw data Object storage service	ADAPTABILITY -> DATA ORCHESTRATION -> DATA SET -> STORAGE FRACTALITY -> ORCHESTRATION -> DATA
WP5T52-04-05	Cloud Datasets version control	ADAPTABILITY -> DATA ORCHESTRATION -> DATASET -> VERSION CONTROL FRACTALITY -> ORCHESTRATION -> DATA

Component ID	Component Name	Related Features
WP5T52-04-03	Cloud S3 compatible data storage	ADAPTABILITY -> DATA ORCHESTRATION -> DATASET -> STORAGE FRACTALITY -> ORCHESTRATION -> DATA
WP5T52-04-07	Images repository	FRACTALITY -> ORCHESTRATION -> SERVICES
WP6T61-02	Edge API	FRACTALITY -> ORCHESTRATION -> SERVICES
…		

Table 2 – Selecting Components from FRACTAL Features

Note that these features are not related to use case business needs. Those needs are covered by use case specific components that are integrated in the FRACTAL Platform. Thus, we have FRACTAL Base Components (or building blocks) and use case specific components that deal with the specific business logic for the use case.

In FRACTAL, these components can be integrated in two ways:

- System specific libraries integrated in the OS of the node
- Container based services, which may include system specific libraries to access specific functionalities, that may be executed on the node or on the cloud.

Having this in mind, the following diagram describes FRACTAL product construction.

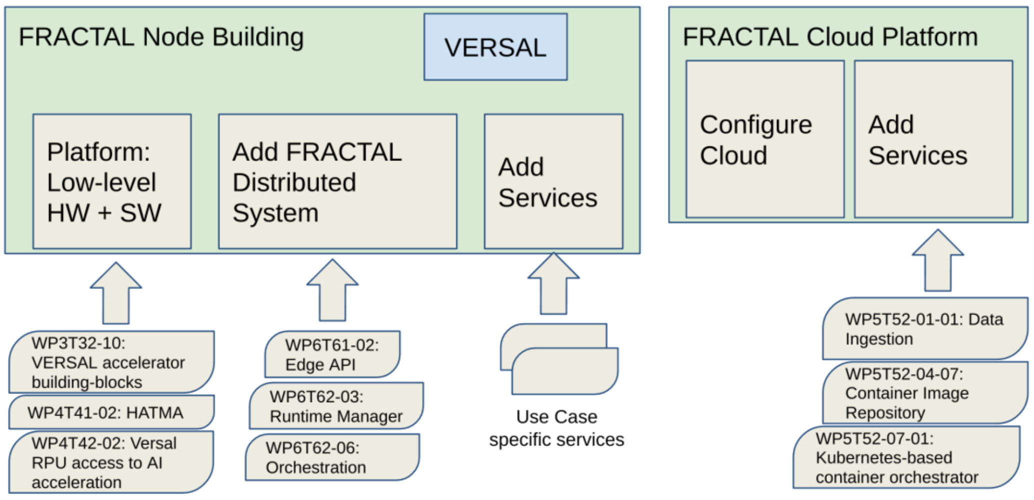
FRACTAL Node Building
VERSAL
Platform: Low-level HW + SW
Add FRACTAL Distributed System
Add Services
FRACTAL Cloud Platform
Configure Cloud
Add Services
WP3T32-10: VERSAL accelerator building-blocks
WP4T41-02: HATMA
WP4T42-02: Versal RPU access to AI acceleration
WP6T61-02: Edge API
WP6T62-03: Runtime Manager
WP6T62-06: Orchestration
Use Case specific services
WP5T52-01-01: Data Ingestion
WP5T52-04-07: Container Image Repository
WP5T52-07-01: Kubernetes-based container orchestrator

Figure 19 – Sample FRACTAL construction process for Versal

Construction process is separated in two parts: FRACTAL Node and FRACTAL Cloud.

For the node, first the node HW platform has to be constructed and its corresponding OS built adding the corresponding selected system libraries (low level components). Then the FRACTAL Distributed System must be added, which includes the container runtime and orchestration capabilities (which may themselves be containers). Finally Use Case specific services are added as containers managed through the FRACTAL Distributed System.

For the cloud, the same implies. First a base cloud configuration is executed over which the cloud services are added as containers (both FRACTAL and Use Case specific cloud services).

Containers play a crucial role on FRACTAL's architecture, being able to produce application logic that can be executed over different hardware platforms or even the cloud, being able to load balance its execution depending on use case needs.

With this process, the industrialization and productization of FRACTAL is achieved, enabling companies to sell products that conform to FRACTAL for any other use case. Following sections detail construction specifics.

7.2 SW integration (PULP, VERSAL), RTOS for PULP

The purpose of this task was to expand the FRACTAL framework to support also the extremely limited nodes. This opens a huge area of new business cases, especially at the consumer sector.

Focus on work at low-end nodes was to introduce and demonstrate APIs, where higher levels of FRACTAL integrate smoothly. This task specially focuses to introduce Posix compatibility on low-end nodes.

The secondary focus was the integration of Asymmetric Multi Processing (AMP), so the Linux capable nodes can benefit RTOS low-energy and real-time features — resulting simpler node electronics and smaller energy budget. Unfortunately, this part was not utilized by any UC. However, while UCs move closer to commercial product the need will arise (after the project).

Major part of the work was utilizing and modifying existing Nuttx background and publishing the results to the Nuttx forum (e.g., over the years the Nuttx Pulpissimo support was deprecated).

Another part was the integration of the Pulpissimo FPGA implementation and other used RISC-V low-end targets (see section 7.2.1).

The AMP implementation task was mainly developing and utilizing Nuttx background. Results were published back to Nuttx. Work in this task, was very hardware dependent. The implementation was demonstrated at yet another RISC-V platform (see section 7.2.2).

7.2.1 Posix compatible RTOS integration to the PULP

Background:

In FRACTAL the general (low risk – fast to deliver) approach is to integrate the middleware layers (by WP4 and WP5) top of the Linux operating system. For use-cases this offers a simple and powerful system to implement the application logic. Also, the acceleration/protection functions that physical HW offers are easy to integrate to the Linux low-level layers. The drawback is that Linux requires adequate amount of processing power and energy. Due high-power nature Versal platform this is not issue as such, but in low-end nature of PULP it limits the options.

On this task a Posix compatible OS (Nuttx) is integrated to PULP. From software point of view, this is limited – less memory, no supervisor abstractions, it lacks high-level languages such as Java and Python. However, unlike most of the Realtime operating systems (RTOS) it has Posix threads and sockets, standard C-APIs and standard dev-tools. Most of the cases source-code developed to the Linux can just be compiled in.

As a result, the Nuttx is compatible with a FRACTAL subset. In its scope, it does not require any special porting for the FRACTAL SW.

As a result, this offers a mechanism to push certain Factual applications to the extreme low-power – low-cost targets.

Development of Pulpissimo requires special tools. Xilinx tools are needed for loading the system binary file from ETH Zürich to the Genesys2 FPGA board. Due instruction extensions the Pulpissimo RISC-V requires a special version of GCC C-compiler.

Integrating the Nuttx RTOS to Pulpissimo requires mainly work on DMA and interrupt processing. Also, the existing binary file supports just one serial port, so assignment of console and logs to that.

Results are published at Nuttx official git-repository.

Due to limited peripheral support and low system clock (10MHz) of Pulpissimo Gensys2 FPGA implementation, the further FRACTAL connectivity development was implemented at ESP32-based RISC-V platform that had more integrated peripherals I.e. Wi-Fi, that Nuttx IP-stack can easily utilize.

Together with UC3 – that was developing Pulpissimo FPGA further, this was too agreed. They demonstrated their system as two. One demonstrating FRACTAL secure AI FPGA implementation based on Pulp and second demonstrating FRACTAL connectivity with ESP32 based RISC-V top of Nuttx RTOS.

Nuttx was not ported to UC3 FPGA, due the FPGA processor peripherals and modem change at next phase when final ASIC is developed.

For similar reasons the AMP (see next subsection) was implemented on Microchip ICICLE platform.

7.2.2 Asymmetric Linux-RTOS multiprocessing to the FRACTAL project

In symmetric multiprocessing (SMP) all processor cores are utilized by same operating system. In asymmetric case (AMP) cores are utilized by different operating systems. There are some benefits of this sort of systems. Here are briefly the two cases:

First: The Real time matter. Linux as such is not a real-time operating system. At certain stage – that is not fixed – Linux begin to lose real time deadlines (Practical tests have demonstrated that response jitter below 1ms is intolerable). One solution to solve this issue, is the Real time version(s) of Linux. There the certain parts of kernel are modified for real time purposes. Drawback on this is that these changes are branches of the main line Linux. While the Linux evolves, the compatibility of this branch weakens and over time the new feature backporting becomes more difficult. This yields to practical problems – typically related to security and compatibility. Another solution is to have a parallel real time operating system (RTOS). This RTOS process the real time tasks, while Linux is the actual computing platform. And, having this RTOS running on same processor chip, eases the system design.

Second: The low power operation. Linux is a complex system that requires complex HW to run – lot of moving parts that consume energy. Driving system to standby and woke up on event is a better solution, however the woke up takes in average more energy than normal operation and often the woke-up events require only little processing or even are false. Again, obvious solution is to introduce a second low-power OS that stays awake, while Linux is standby. This OS can verify the woke-up events and some cases process them. And in cases where "The Force" is needed, RTOS can woke-up Linux OS and delegate the event. Like the above case, the system design is easier, when the low-power OS and application OS are in same physical chip.

Two cases above and combination of them are valuable tools for certain application types in FRACTAL context. In task we seek to run Nuttx (RTOS) in one core and Linux in other cores. The Posix compatibility offers yet another potential compatibility benefit: Due both OSes are in same processor context; they may share same security, data and even binaries.

Challenges on this task are related to the context switching, security and shared resources matters below the OSes.

As in previous task the implementation was done according to the spiral model. At first an adequately good platform is selected, where some version of previous task Nuttx implementation is integrated together with some Linux. The second target is a FRACTAL improved version of that. And so on.

Security features, low-power operation, application requirements need to be collaborated with other FRACTAL partners. This can be also in focus of Versal platform.

AMP communication between Linux and NuttX is possible via Remote Processor Messaging (RPMsg). Both operating systems support RPMsg over VirtIO. Basically, shared memory with proper notification mechanisms is the foundation for this AMP mechanism.

It's even possible to use multiple RPMsg channels. For example, Linux may talk to two different operating systems. However, support for this kind of features is very limited at the moment. A part of this work is available at commit: https://github.com/apache/nuttx/commit/3afc83abc7887b0371a1a6501a056641c5f5ff3 on NuttX operating system.

Moreover, the hardware has likely a fixed set of peripherals, and thus not all the operating systems can use them simultaneously. Especially NuttX has a number of RPMsg drivers: block drivers, network drivers and serial drivers just to name a few. Those drivers may be accessed from a remote processor. For example, NuttX may have complete control over a network driver, but Linux may still use it via the RPMsg based AMP system. RPMsg drivers are less common in Linux than in NuttX.

7.3 Electronic System-Level HW/SW co-design

Systems based on heterogeneous parallel architectures (*Heterogeneous Parallel Systems* - HPSs) have been recently exploited for a wide range of application domains, especially in the *System-on-Chip* (SoC) form factor (e.g., Xilinx ZYNQ/VERSAL and Altera Cyclone V SoC families). Such systems can include several processors, memories, and a set of physical links among them. By definition, the set of processors in the same architecture is heterogeneous. This implies that it is possible to exploit, at the same time, the following processing classes[4]:

- *General-Purpose Processors* (GPP): x86/x64, ARM, MicroBlaze, NiosII, Leon3, etc.
- *Application-Specific Processors* (ASP): *Digital Signal Processor* (DSP), *Graphics Processing Unit* (GPU), *Network Processor* (NP), *Artificial Intelligence Processor* (AIP), etc.
- *Single-Purpose Processors* (SPP): AES encoder/decoder, JPEG encoder/decoder, UART/SPI/I2C controllers, AI engines; in general, every ad-hoc developed digital HW component (a.k.a. co-processors or accelerators).

[4] Frank Vahid and Tony Givargis. 2001. Embedded System Design: A Unified Hardware/Software Introduction. John Wiley & Sons, Inc.

Finally, such processors can be adopted in the form of soft, hard or fuse (i.e., *hardwired*) IP cores or as discrete integrated circuits (IC) mainly depending on the final system form factor (i.e., on-chip, on-FPGA, on-board) and scope (complete product or platform).

HPSs are often used to implement *Dedicated Systems* (DS). DSs are digital electronic systems with an application-specific HW/SW architecture. They are specifically designed to satisfy a priori known application requirements, both functional (F) and not functional (NF). A DS could be then embedded in a more complex system and/or it could be subjected to hard/soft real-time constraints. When DSs are based on HPS they are called *Dedicated Heterogeneous Parallel Systems* (D-HPS).

Apart from possible differences in terminology and composition, for this kind of systems one consideration is always true: they are so complex that the adopted *HW/SW Co-Design Methodology* plays a major role in determining the success of a product. Moreover, in order to cope with such a complexity, the selected methodology should allow the designer to start working at the so-called *Electronic System-Level* (ESL) of abstraction. This means to be able to start the design activities from an executable model of the system behaviour based on a given *Model of Computation* (MoC) that would be unifying for HW and SW, and that could be described by means of a proper specification/modelling language. In fact, in the past years, a remarkable number of research works have focused on the system-level HW/SW co-design of D-HPS. In such works, the most critical issues have been always related to the *System Specification* and *Design Space Exploration* activities. In the first activity, the designer models the behaviour of the desired system (specifying also possible NF requirements), the available basic HW components, and the target HW architecture. The second activity is then related to the approach, automated or not, used to find the best HW/SW partitioning and mapping for the final system implementation. The main differences among the various approaches are related to the different amounts of information and actions that are directly requested to the designer and that are so heavily influenced by his/her experience. In particular, a lot of approaches explicitly require as input the HW architecture to be considered for mapping purposes. Very few others try to fully addresses the problem of both to "automatically suggest an HW/SW partitioning of the system specification" and to "map the partitioned entities onto an automatically defined heterogeneous parallel architecture". In the context of the latest category, during other ECSEL RIA projects (e.g., MegaM@RT2, AQUAS, FITOPTIVIS, COMP4DRONES), it has been defined and improved a model-based ESL HW/SW co-design methodology (and related prototypal toolchain), called HEPSYCODE[5], targeting heterogeneous parallel dedicated systems. HEPSYCODE reference ESL HW/SW co-design flow is briefly described in the following, together with a proposal about possible adaptation/integration opportunities of HEPSYCODE in the context of the FRACTAL project.

[5] http://www.hepsycode.com

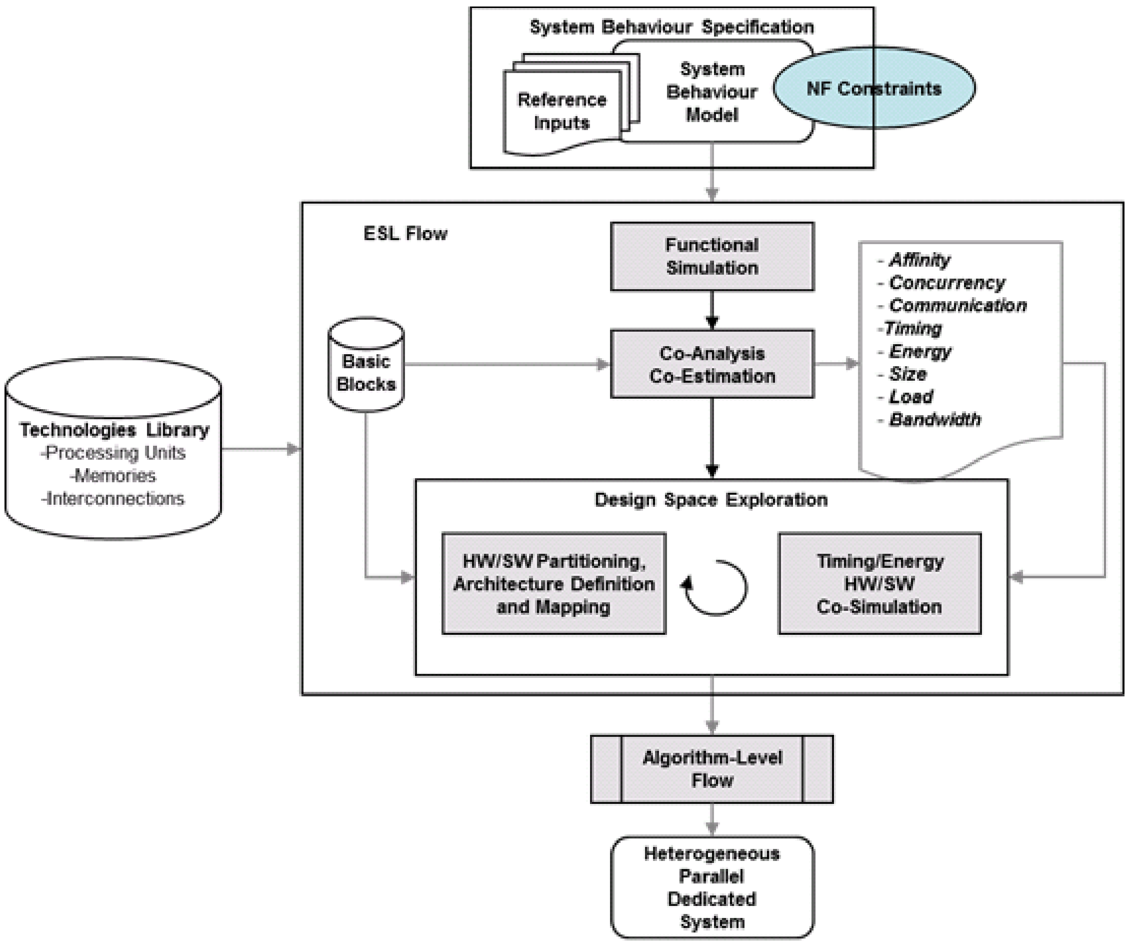

Figure 20 – The reference ESL HW/SW co-design flow

7.3.1 Reference ESL HW/SW Co-Design Flow

The HEPSYCODE ESL HW/SW co-design flow is shown in figure 20: it shows the main co-design steps and the related items that are briefly described in the following paragraphs. More details about the whole methodology can be found in [6,7,8,9].

7.3.1.1 System Behaviour Specification

The entry point of the reference co-design flow is the *System Behaviour Specification* (SBS). It is composed of *System Behaviour Model* (SBM), *Reference Inputs* (RI), and *Non-Functional Constraints* (NFC).

SBM represents the behaviour of the system to be implemented, i.e., the functional requirements. It is based on a CSP-like (*Concurrent Sequential Processes*) MoC[10,11] and described by means of the *SystemC* language[12].

RI is a set of input-output tuple pairs, possibly timed, that represents the expected outputs from the SBM when specific inputs are provided to it. RI is of critical importance since it has to be as much as possible representative of the actual operating conditions of the system (a.k.a. *Golden Inputs*).

NFC represents requirements related to aspects orthogonal to the behaviour. In fact, they specify a set of constraints that have to be satisfied while still following a correct behaviour. They are currently related to one or more of the following issues:

- Timing Performance Constraints
    - *Time-To-Completion constraint (TTC)*
        - Time allowed to complete the processing related to RI

[6] L. Pomante. "System-level design space exploration for dedicated heterogeneous multi-processor systems". IEEE Int. Conf. on Application-specific Systems, Architectures and Processors, 2011.

[7] L. Pomante, D. Sciuto, F. Salice, W. Fornaciari, C. Brandolese. Affinity-Driven System Design Exploration for Heterogeneous Multiprocessor SoC. IEEE Transactions on Computers, vol. 55, no. 5, May 2006.

[8] Pomante, L., Muttillo, V., Santic, M., Serri, P. SystemC-based electronic system-level design space exploration environment for dedicated heterogeneous multi-processor systems. Microprocessors and Microsystems, 72, 2020.

[9] Ciambrone, D., Muttillo, V., Pomante, L., Valente, G. HEPSIM: An ESL HW/SW co-simulator/analysis tool for heterogeneous parallel embedded systems, 2018. 7th Mediterranean Conference on Embedded Computing, MECO 2018 - Including ECYPS 2018, Proceedings.

[10] C.A.R. Hoare. Communicating sequential processes. Communications of the ACM, 21(8):666–676, August 1978.

[11] http://www.usingcsp.com

[12] SystemC, http://www.accellera.org

    - *Time-To-Reaction (TRC)*
        - Time allowed to complete the processing related to specific processes of the SBM
- Energy Consumption Constraints
    - *Energy-To-completion (ETC)*
        - Energy consumption allowed to complete the processing related to RI
- Mixed-Criticality Constraints
    - Isolation
        - Do not map processes with different criticalities on the same processor
    - Isolation with hypervisor technologies (HPV)
        - Do not map processes with different criticalities on the same HPV-based SW partition
- Architectural Constraints
    - Set of available processors and physical links
    - Min and max number of available processors and physical links instances
    - Available area (for PCB/ASIC) or an equivalent metric for FPGA
    - Reference template architecture
        - HW
            - Distributed/shared memory
            - Homo/heterogeneous mono/multi-core processors
        - SW
            - Available (hyper)scheduling policies.

7.3.1.2 Technologies Library

In order to list and describe the basic HW elements available to automatically build the final architecture, a proper *Technologies Library* (TL) provides a characterization of available processors, memories and physical links. Such a library contains information like processing classes (i.e. only GPP, DSP, and SPP), operating frequencies, maximum load (for GPP and DSP classes), capacity (for SPP and memories), max bandwidth (for physical links), relative cost (considering the cost related to obtain a component and/or the effort needed to use it), and so on. Such information is then exploited during the different steps of the co-design flow.

7.3.1.3 Functional Simulation

The first step of the proposed co-design flow is the *Functional Simulation* where SBM is simulated to check its correctness with respect to RI. Such a simulation allows also to consider timed inputs, i.e., there is a concept of simulated time, but it doesn't consider the time needed to execute computation and communication (i.e., 0 simulated time). If SBM is not correct (i.e., wrong outputs or critical conditions such as deadlocks) it should be properly modified and simulated again. The early detection of anomalous behaviours allows the designer to correct the specification avoiding a late discovery of problems that could lead to time-consuming design loops.

7.3.1.4 Co-Analysis and Co-Estimation

This step aims at extracting as much as possible information about the system by analysing the SBM while considering the provided TL. This step is composed of *Co-Analysis* and *Co-Estimation* activities.

During *Co-Analysis*, SBM is analysed to evaluate three metrics: *Affinity*, *Communication* and *Concurrency*. The first one represents how much a process is suitable to be executed on a specific processor class (i.e., GPP, ASP, SPP). The second one is the evaluation of the number of bits that the different processes pairs have exchanged during the simulation. The third is related to how much concurrency has been found during the simulation in the activities of processes and channels.

Co-Estimation provides a set of estimations about *Timing*, *Energy*, *Size*, *Load* and *Bandwidth*. Timing is related to the estimation of the number of clock cycles needed, by each processor in the TL, to execute each single statement composing the SBM processes[13]. Energy is related to the estimation of the Joule consumed, by each processor in the TL, to execute each single statement composing the SBM processes[14]. Size represents the number of ROM/RAM bytes needed for SW implementations and equivalent gates (or similar metrics for FPGA) for HW ones[15]. Finally, by exploiting Timing data and considering the TTC constraint, it is also possible to estimate the Load associated with the execution of the SBM processes when mapped on a single instance of each processor in TL, and the Bandwidth needed to the different processes to communicate while fulfilling the TTC constraint. The extraction of these data from the SBM is an important step that allows, during the following design space exploration, the identification of the number and type of processors and physical links needed to satisfy the NFC.

7.3.1.5 Design Space Exploration

Finally, the reference co-design flow reaches the *Design Space Exploration* (DSE) step that is constituted of two iterative activities: *"HW/SW Partitioning, Architecture Definition and Mapping"*, and *"Timing/Energy HW/SW Co-Simulation"*. All the data (i.e., metrics and estimations) extracted in the previous steps are then used to drive the DSE by considering all the NFC. The "HW/SW Partitioning, Architecture Definition and Mapping" activity is based on a genetic algorithm that allows exploring the design space looking for feasible mapping/architecture items suitable to satisfy imposed

[13] V. Muttillo, G. Valente, L. Pomante, V. Stoico, F. D'Antonio, and F. Salice, "CC4CS: an Off-the-Shelf Unifying Statement-Level Performance Metric for HW/SW Technologies", In Companion of the 2018 ACM/SPEC Int. Conf. on Performance Engineering (ICPE '18).

[14] V. Muttillo, P. Giammatteo, V. Stoico, L. Pomante. An Early-Stage Statement-Level Metric for Energy Characterization of Embedded Processors. Microprocessors and Microsystems, 2020.

[15] Brandolese, C.; Fornaciari, W.; Salice, F. An area estimation methodology for FPGA based designs at systemc-level. Design Automation Conference, 2004. Proceedings. 41st, 2004 Page(s):129 – 132).

constraints. Then, the "Timing/Energy HW/SW Co-Simulation" activity considers suggested mapping/architecture items to actually check for Timing/Energy NFC satisfaction. If the suggested mapping/architecture item does not meet such constraints, the designer should perform again the DSE by changing some exploration parameters, by modifying the starting SBM, by enriching the TL with new elements, or by relaxing some constraints.

7.3.1.6 Algorithm-Level Flow

When the mapping/architecture item proposed by the DSE step is satisfactory, it is possible to implement the system. For this, the SW-mapped processes are typically transformed in C/C++ code, with the support of a possible embedded and/or real-time OS, while the HW-mapped ones are transformed in synthesizable HDL code or implemented by means of existing COTS component depending on the final system form factor. It is worth noting that such transformations are done automatically or manually depending on the language and the coding style adopted to describe the SBM. This step is fully based on existing commercial algorithm-level methodologies and tools.

7.3.2 HEPSYCODE in the FRACTAL project

Given the ever-increasing opportunities provided by the advancement in the HW/SW technologies, there is a strong need for ESL methodologies and tools able to keep as much as possible smaller the design-productivity gap in the field of HW/SW dedicated systems. According to this scenario, the HEPSYCODE methodology, briefly described in the previous sub-section, has been customized to consider some of the platform related to FRACTAL (i.e., ZYNQ Ultrascale+ and VERSAL) and exploited to support UC6 development. As detailed below, the customization and the analysis performed during UC6 development are the most relevant contributions.

In order to early support the FRACTAL designer during the mapping of a set of functionalities on UC6 FRACTAL node, and during the related FRACTAL node customization, the following ESL activities have been performed:

- ESL modelling of the Totem Node behaviour (SBM), by considering as RI the ideal scenario described in D8.2 (sec. 5.2.2). Such a modelling has been done by referring to a CSP-like MoC and by using the SystemC modelling language.
- Analysis of UC6 requirements (see D8.2 – sec. 5.1) to identify the NFCs relevant for the HEPSYCODE methodology. In particular, REQ_UC6_26 has been considered to set TRC = 1 sec as the maximum time that shall elapse from the moment a user is in front of a Totem and the moment the Totem start to be reactive in a tailored way.
- Enrichment of the TL to consider specific ZYNQ Ultrascale+ and VERSAL components: ARM core (GPP), Xilinx DPU (ASP), and FPGA (SPP).
- Timing HW/SW co-estimations, related to the added TL components, in order to annotate the SBM with information about timing performances of different "SBM to components" mappings.

- Timing HW/SW Co-Simulations to verify which mapping/architecture items are suitable to implement SBM while satisfying TRC. It is worth noting that, since the Architectural Constraints impose the use of a predefined SoC, the DSE has been limited to co-simulation issues.

As main results, by considering the Data Flow Diagram (DFD) shown in figure 21 (adapted from the one in D8.2- sec. 5.2.3 and used to create the SBM), it has been possible to early verify that the mapping shown in figure 22 (i.e., the one finally selected for UC6, see D8.2 - sec. 5.2.1) is able to satisfy TRC.

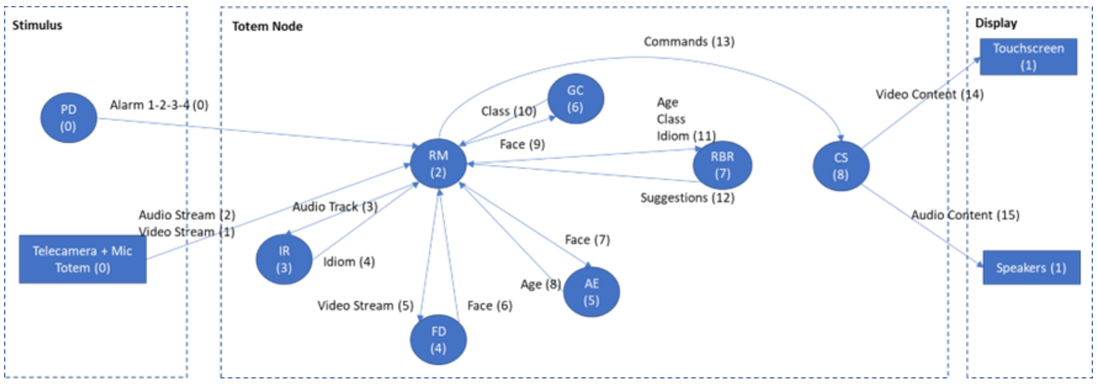

Figure 21 – Totem Node DFD

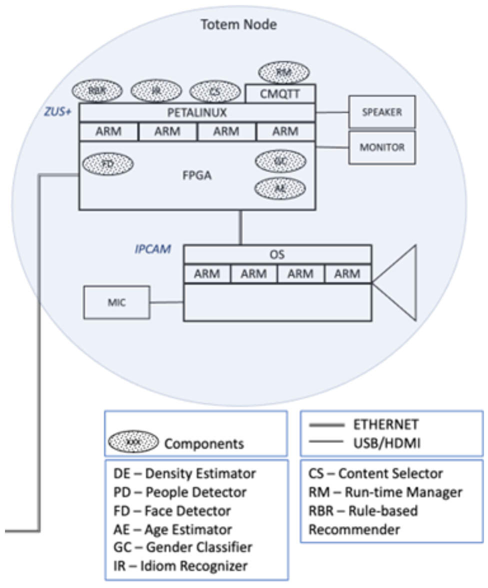

Figure 22 – Totem Node Architecture and Mapping

Moreover, it has been also possible to verify that, by offloading additional instances of GC and AE to available nodes in the surrounding, it would be possible also to manage more complex scenarios (e.g., multiple persons in front of the same camera, multi-touchscreen totem) by still satisfying TRC.

Summarizing, by exploiting an ESL approach, combined with timing estimations related to single components, it has been possible to early verify timing performances of several mappings, finally identifying the most suitable ones without the need to completely perform time consuming integration activities at lower levels of abstraction. Finally, as a more general result, the HEPSYCODE methodology has been customized to consider some of the platforms related to FRACTAL and so it could be potentially exploited to support other UCs and the whole FRACTAL approach.

7.4 Methodologies for VERSAL platform

The heterogenous nature of the Versal ACAP devices resemble one readily available example of a platform to scale the FRACTAL nodes as described in the earlier chapters. The capabilities support the features to implement components way into the AI techniques required for Video analysis as presented in chapter 5. Even the orchestration and separation methods required for the safety and reliability requirements, chapter 6, are available to add to the node design.

To cater consistent integration of components on the different processing elements, the AMD-Xilinx Vitis toolchain is deployed. This is not a monolithic approach to deliver a closed node artifact but instead lends to a modularity that is beneficial for the use cases' requirement-driven selection of actually used features from the catalogue. Part of the integration work within FRACTAL also provides increased visibility into the toolchain applicability and best practices to enable efficient work on the Versal platform.

Versal ACAP based designs are based on this design flow to define the hardware features as well as the software ecosystem for the heterogenous processing system. These tools provide for basic platform creation as well as accelerator-based design and the application on either bare metal or OS support packages.

While the efficient design creation is typically ramped through authorized training partners of AMD-Xilinx, there are substantial tutorials available from https://github.com/Xilinx to get an overview of the basic design steps.

For most part the FRACTAL designs utilize the VCK190 development kit to derive the components or apply required use case integration steps. Along WP3 a base platform component has been derived where a FRACTAL node should be based on. The use case specific features and requirements yield a further selection of components, that require additional integration of hardware and / or software components. The tools are available to cover these integration steps, also along the Application Acceleration Development (UG1393[16]). The typical steps are shown here:

[16] https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration/Building-and-Packaging-the-System

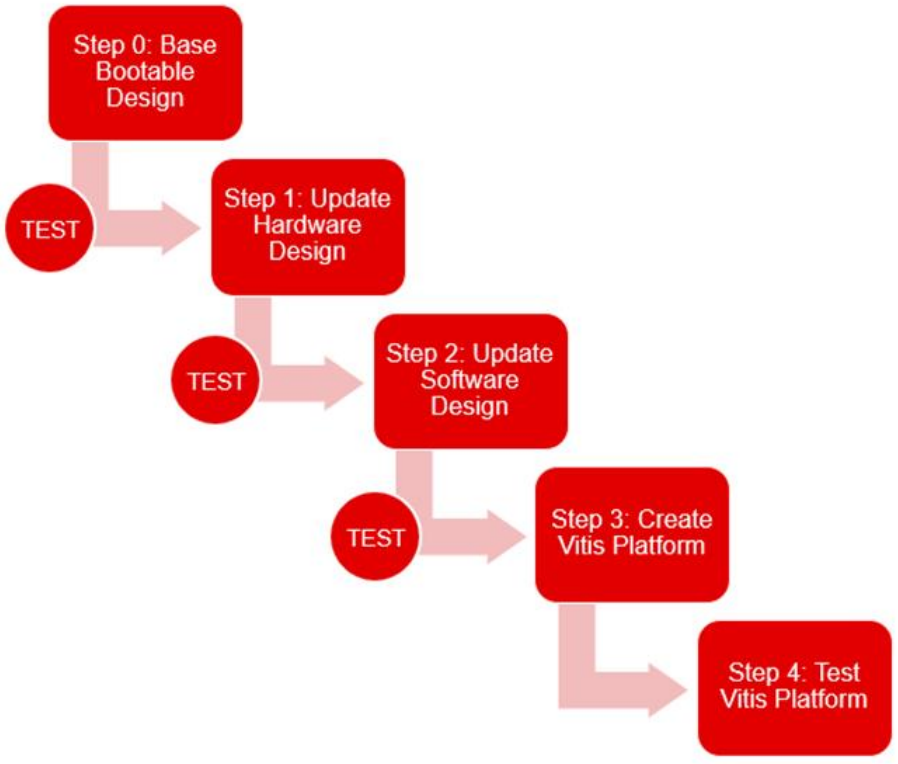
Step 0: Base Bootable Design
TEST
Step 1: Update Hardware Design
TEST
Step 2: Update Software Design
TEST
Step 3: Create Vitis Platform
Step 4: Test Vitis Platform

Figure 23 – Basic platform design flow

Extend the basic static platform: to cater for design specific elements, like support for proprietary FMC cards, an existing platform can be extended by re-inserting it into the Vivado flow to add to or adapt the initial bit stream. Some of the components of the technical work packages use this methodology to encapsulate the contribution. Specifically additions for safety setups or additional separation between the heterogenous cores are benefitting from this approach.

Extend the design with accelerators: the basic provided platform allows for software defined extensions that manifest as hardware kernels. Such approaches are highly adaptable for fast design turn around and dynamic workload support. Typically the components for pre- and post-processing of AI workloads utilize this design methodology, where the respective kernels are often implemented by high level synthesis (HLS).

Integrate software and drivers: The Vitis toolchain allows for development of setup or control flow software directly within the same workspace as the hardware development described above. The components that are using this flow target any of the heterogenous processor cores or even full OS- based application.

While the higher level software components for use case applications running on the FRACTAL node microservice based framework are supported within these IDE workspaces, such software development is also available from the PetaLinux[17] tools, and any build system of own definition can be used.

7.5 Methodologies for PULP platform

The Parallel Ultra Low Power (PULP) platform is a RISC-V based open-source platform for energy efficient computing. While there are multiple PULP systems with different capabilities, within the FRACTAL project the reference PULP system is the PULPissimo system available under:

https://github.com/pulp-platform/pulpissimo

The platform is fairly mature and has been used as part of various ASIC tape-outs so far and comes with FPGA images for popular XILINX boards such as the Genesys II, allowing partners to hit the ground running. As part of FRACTAL, the goal was to improve and adapt this basis system for several use cases with additional capabilities developed by FRACTAL partners as part of technical developments WP4, 5 and 6.

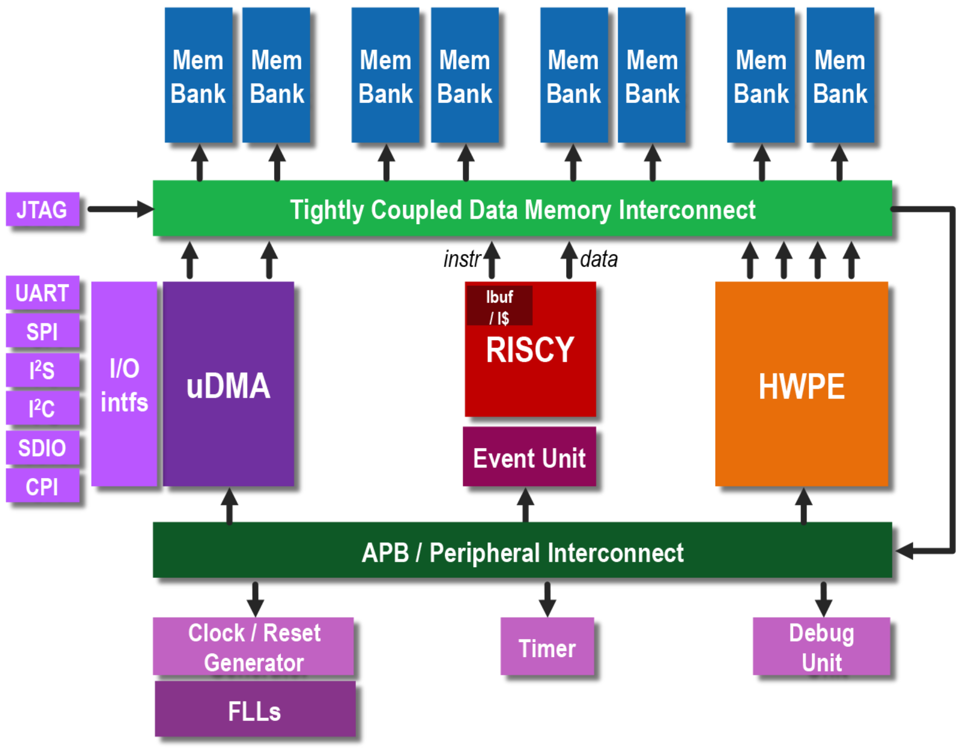

Figure 24 – PULPissimo block diagram

Specific improvements to the PULP platform supported through the involvement in FRACTAL have been:

[17] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842250/PetaLinux

- The ISA extensions for low-precision training on RISC-V cores. These include 8- and 16-bit floating point units that can perform dot product operations and accumulate on 16- and 32-bit registers.
- On-demand grouping of RISC-V cores in many-core architectures to improve reliability. This way a three-core system can either operate in high-performance mode where each core contributes to computation, or the three can be grouped together to work as a triple redundant mode.
- Adding specialized ML accelerators optimized for temporal convolutional networks quantized to ternary values to be used as part of a dynamic vision sensor.
- Developing systematic fixes to timing-channels in our processor cores, which introduces a fence.t instruction.
- Working on fine-grained power gating techniques to reduce the leakage power of many-core architectures.

In addition to these specific contributions, there were many improvements and fixes in the open-source repositories improving the general quality and availability of the PULP platform for all partners.

From the onset, the positioning of the PULP platform has been complimentary to the XILINX Versal platform. While Versal is addressing high performance, industrial maturity and established processor cores, for FRACTAL, PULP was designed to address low power IoT operation, with a flexible and open architecture allowing partners to easily integrate changes to the platform using an open-source ISA.

The training videos and material available on the PULP platform site under:

https://pulp-platform.org/pulp_training.html

For example, the following (3.5h) training video explains the design and features of PULPissimo:

https://www.youtube.com/watch?v=27tndT6cBH0

There are several different ways in how changes to PULPissimo as part of developments in FRACTAL have been implemented:

- **Adapting software** to be compatible with PULPissimo. In addition to FreeRTOS and seL4 there have been successful ports of various other operating systems and libraries. PULP based systems rely on standard GCC/LLVM based SW development flow and also include a custom software development kit. The following tutorial explains the PULP software development kit: https://youtu.be/Ydd9TlKQiO4
- Adding **accelerators and peripherals** to the PULPissimo system to enhance its hardware capabilities. In addition to standardized AXI/APB compatible peripherals, PULPissimo also supports accelerators that can access a shared scratchpad memory with the RISC-V processor greatly reducing the overhead

of data transfers. The following tutorial is on the HW/SW co-development needed for such modifications: https://youtu.be/B7BtaYh3VqI

- Adding **instruction set extensions** to the RISC-V core in the system, which could improve the performance of the system significantly. RISC-V has a very clear methodology for such extensions, and ETH Zurich has significant experience with them. In fact, the default processor core in PULPissimo (RI5CY/CV32E40P) has already many extensions for digital signal processing. However, SW tools must be made aware of these modifications, so that code can be generated that maps to these additional instructions.

Some FRACTAL partners have utilized sophisticated systems from the PULP platform like the HERO (heterogeneous Research Platform) or the 64-bit Ariane/CVA6 system with Linux support. ETH Zürich has guided partners that want to explore these options, but has concentrated its main effort around the PULPissimo.

The flexibility offered by the platform and the experience of ETH Zurich on actual ASIC implementations of the platform has helped partners to assess the cost and benefit of various improvements and changes that are explored throughout FRACTAL. Such evaluations has provided value for partners even in cases where a direct use case implementation was not feasible due to technical reasons.

7.6 Methodologies for hardware accelerators

Naturally, Deep Neural Networks (DNN) perform many computations on a huge amount of input data for generation of the outcome. The use of classic general-purpose computer designs has been shown to be not very efficient for execution of such networks due to the sequential concept of computation that these architectures apply. Therefore, building a specialized hardware that provides better performance and is energy efficient is required.

However, DNNs are a central part of machine learning which means they must be trained before they infer. During the training phase the DNN's parameters (weights and bias) are derived. As a process training is much more complex than inference, where inference computes the outputs based on the previously trained weights and biases. Since training of DNN is a onetime process and requires much more computation power it can be performed on high performance machines that have no restriction on energy consumption. On the other hand, inference is usually applied on many devices located near the sensors and runs repeatedly with different input data patterns. Thus, specialized hardware needs to be customized for inference execution to satisfy performance requirements and energy efficiency goals.

The following subsections describe the methodology used for building and evaluation of the hardware accelerators in FRACTAL project. After listing the main components required for building of the hardware accelerator, the metrics that need to be observed for evaluation and optimization of the performance are described. All these steps are performed with the help of Catapult High-Level Synthesis tool, which enables the hardware designer to evaluate performance parameters at an earlier stage of the design before generating the hardware implementation details. The tool itself has integrated features that makes the performance evaluation a straightforward process. Once the design satisfies the performance requirements defined in the project the High-Level Synthesis tool automatically generates the implementation outcome which can be for ASIC or FPGA technology.

7.6.1 Hardware Accelerator Architecture for Deep Neural Network

DNN accelerators typically consist of an array of processing elements (PE) and memory blocks interconnected by a Network on Chip (NoC). The PEs are simple Multiply-Accumulate (MAC) units capable to perform multiplication of inputs and weights and add the resulting products to the partial sum.

Memory of hardware accelerators are organized hierarchically consisted of register files (RF), global buffers, and main memory. RFs are the smallest and fastest memory units located on PEs. They hold data immediately available to the MAC unit. The type of data stored in RF can be weights (weight stationary), partial sums (output stationary), or a combination of both types (row stationary). Which type of data is located on RF depends on the data flow model in use. There are hardware accelerators as well without RFs (no local reuses) in case the size of the chip area is critical. A global buffer is an intermediate memory layer located on-chip to hold fragments of the weights and inputs. Its location and size enable a global buffer to respond faster and efficiently when its content is reused by PEs. The last layer is the main memory, usually in form of off-chip DRAM memory, that holds all weights and input data.

Hardware accelerators for DNN can either be implemented on Field Programmable Gate Arrays (FPGA) or as Application Specific Integrated Circuits (ASIC). FPGAs are used for development of prototypes since the design for such technology is simple and the time to develop the product is shorter. On the other hand, ASICs can be optimized for much more parameters, have lower energy consumption and better computation performance. However, for ASIC technology the production costs are much higher combined with longer development cycles compare to FPGA solution.

7.6.2 Metrics for building optimized HW accelerator

To build an efficient hardware accelerator for DNN it is important to take into consideration a set of metrics that are key for achieving an optimal solution. Mainly the efficiency of the accelerator is associated with the number of operations per Watt, but this metric is a composition of accuracy, throughput, latency, energy consumption, cost, flexibility, and scalability[18]. *Accuracy* indicates the quality of

[18] Vivienne Sze et al. „Efficient processing of deep neural Networks: A Tutorial and Survey" – Proceeding of the IEEE, Vol:105, Issue: 12, Dec 2017

inference outcome and should be high enough to perform correct classification. The accelerator should have capability to process enough data in a given period of time so its *throughput* can achieve real-time performance. Also, the time between input and output should be short if it is required to have low *latency*. When an accelerator is used within an edge device the *power consumption* and *energy efficiency* must be taken into consideration during design phase to ensure that the accelerator operates within the boundaries of the power envelope. It has been observed that frequent access to the main memory for data read/write is one of the main sources of energy consumption and compared to the demand of the various arithmetic operation performed in the accelerator it is few magnitudes higher. *Cost* is constrained by the required hardware volume and the size of the market. Obviously, it is important the designed hardware accelerator is attractive from a financial point of view. If the accelerator is more generic, it can execute a wider variety of different inference tasks, a feature which makes the accelerator more *flexible*. The last metric, *scalability*, refers to how well the accelerator can adopt from perspective of throughput and energy consumption when its number of components increases. Thorough upfront evaluation of all the mentioned metrics allows the hardware designer to evaluate if the accelerator is a beneficial and viable solution for a given application.

7.6.3 High Level Synthesis to build HW accelerator

Nowadays, hardware design processes have become quite complex and sometimes manual coding is even impossible due to increased complexity of the needed hardware solution. High-level synthesis (HLS) aims to generate synthesizable register transfer level (RTL) implementation of the hardware derived from its high-level specification. Its goal is to automate all the intermediated processes performed between specification of the hardware and the RTL level. The approach raises the level of abstraction on functional and implementation details and thus eliminates all the steps that were performed manually in the past. The whole concept is like the approach of compiling high-level code into assembler one. HLS also makes the verification process faster compared to the time required to perform the same process on Register Transfer Level (RTL), which sometime can take so long that the process itself becomes impractical. By elimination of the manual steps and reduction of the verification time HLS brings a lot of benefits:

- The automated generation of RTL implementation is less prone to errors due to the predefined rules applied for its generation.
- Well-designed automation tool can generate RTL that outperforms in quality compared to human design RTL level.
- Gives hardware designer more time to spend on design space exploration rather than on implementation.
- HLS simulation runtime is few times faster than RTL simulation runtime.
- HLS makes the design platform independent by enabling its implementation on a wide range of platforms.

The process of generating the RTL hardware description from its specification is a multi-level task[19] as shown in figure 25. The process starts with describing the desired functionality of the hardware in a high-level language. The HLS tool takes the input, compiles it and generates a formal model. This model is then converted into a structured network of components (functional units, memories, controllers, and interfaces). During this stage the HLS firstly allocates the resources that are needed for computation. Next, the HLS schedules the execution order of derived operations. Thirdly, it binds the allocated resources to the corresponding operations derived from the formal model. These three tasks are interrelated and for optimal results they should be done in conjunction. The output is an RTL description of the hardware consisted of control and data path. It is important that HLS takes as input untimed high-level hardware description and transforms it into a fully timed hardware implementation.

The most popular languages used in HLS tools for functional description are SystemC and C/C++. The resulting RTL description is generated in form of popular Hardware Description Languages (HDL) like VHDL or VERILOG. However, not all C/C++ code structures can be converted into an HDL. Usually, non-synthesizable segments of the code are sections used for system calls, input/output structures, and pointers. Therefore, it is required from the implementer to distinguish between code structures that can and cannot be synthetized.

[19] Philippe Coussy et al. "An Introduction to High-Level Synthesis" – IEEE Design & Test Computers, 2009

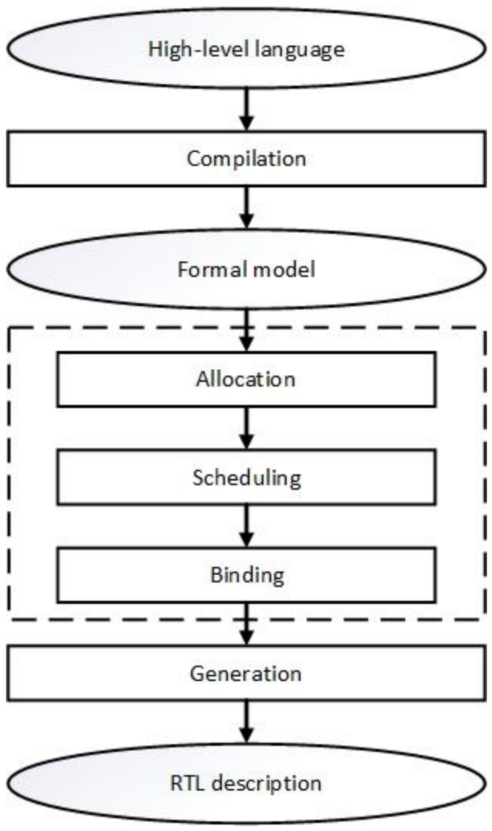

Figure 25 – HLS design flow

Mainly, HLS generates the HDL based on a set of predefined transformation rules. However, these rules can be overruled with pragmas within the source code if a better outcome can be produced. The usual techniques used in HLS design for performance improvement of the hardware are pipelining, unrolling, and in-lining[20]. Pipelining enables parallel execution within the loop, unrolling generates more processing elements in parallel, while in-lining removes the hierarchy. All these techniques can be implemented in form of pragmas or within the HLS tool.

7.7 Integration of Speech-based Signal Processing algorithms

The number of connected devices has increased rapidly. The Internet of Things (IoT) framework and the diffusion of related enabling technologies, such as Device-to-Device (D2D) communications, cloud and edge computing and big data analysis have strongly improved the feasibility of connecting and communicating through many mobile nodes, often in non-ideal environmental conditions. Moreover, the importance of audio speech processing is demonstrated by the use of this type of approach in a

[20] Adam Taylor "Porting Vivado HLS design to Catapult HLS Platform" – White paper

wide variety of commercial applications. This growing interest in speech and speaker recognition is witnessed by the widespread use of applications, such as speaker verification and authentication procedures, gender recognition and language recognition.

Other common applications of speech processing techniques lie in the range of accessibility solutions: the most remarkable examples of this kind are the speech-to-text and text-to-speech functionalities. Moreover, in numerous practical cases a speaker talks in an environment in which many smart devices (e.g., mobile phones) are present, for example during a seminar presentation, a conference call, or during a lecture in a classroom.

For this reason, within the FRACTAL project, understanding audio context represents an important tool that can be extremely useful in several realistic scenarios. As the quality of audio signals is deeply influenced by the environmental conditions, an exhaustive study of the performances of the most common speech processing techniques in variable noise conditions and at different source-receiver distances is required. For this reason, many literature works address the issue of speech processing in challenging environmental conditions, proposing noise robust audio processing techniques, able to provide good performances even if noise is corrupting the audio signal.

Our contribution within the FRACTAL project is to provide a useful tool for idiom recognition, in order to support the user in common use case applications. For instance, a totem node supporting users in public places could be used by people of different nationalities, and therefore a custom-tailored response to the user based on his/her native language would be beneficial for user experience and tool usability.

Many techniques have been used to tackle the language recognition problem, among which the most widely used is certainly hotword-based recognition. This method is rather quick, matching the strict constraints of almost real-time applications, and also very effective in terms of system accuracy. The rationale is to collect a speech utterance, remove unnecessary silence and noise, and study the correlation in terms of signal features and characteristics, in order to match a pre-defined template signal. Such signal will be related to a specific language, thus allowing the system to interact with the user and select proper content based on the processing outcome.

The main purpose of speech-based signal processing algorithms is to analyse audio signals and correlate them with a set of hotwords matching different languages. In this way a user speaking in his/her native language will be able to interact with smart systems in a transparent and easy way.

8 Security Risk Management Methodology

This chapter is a summary of deliverables D4.1, sections 4.5.2 and 4.5.3, and D4.5, sections 4.4, 4.5, and 4.6. The methodology described in this section is the one that has been followed for the development of the risk analysis that is explained in chapter 6 of deliverable D4.5 and for the development of the WP4T44-02 OS Security Layer component.

For more information, check the sections referred above.

8.1 Introduction to security in embedded systems

The use of embedded systems for industrial applications in Industry 4.0 has grown in complexity, including increased connectivity for remote monitoring and management. However, this increased connectivity also poses a greater risk of attacks and intrusions, making it necessary to develop safe and reliable systems. Historically, embedded systems have focused on functionality and efficiency rather than security, leaving them vulnerable to attacks. The industry is responding by implementing the IEC 62443 standard, which sets requirements for the development life cycle of these systems and defines safe product development processes. During the design phase, security requirements must be established for later implementation and systems must be tested for cybersecurity. The IEC 62443-4 standard must be considered and used as a precedent for analysing system security requirements.

8.2 Background on IEC 62443

The IEC 62443 international series is the standard to consider when it comes to industrial cybersecurity, as it defines the development of secure industrial control systems. Developed by the International Electronic Commission, it is a precursor to the ISA99 standard of the International Society of Automation.

It is structured in several documents which are classified in four main groups:

- IEC 62443-1: General concepts, which presents the main concepts, terms and acronyms related to cybersecurity in an industrial environment.
- IEC 62443-2: Policies and procedures, which describes how security should be managed within an organization and the requirements that security management systems must meet.
- IEC 62443-3: Organization and distribution, which explains how a plant should be divided into zones with different levels of security, and their requirements and security systems. Four security levels are established, SL1, SL2, SL3 and SL4, which are applied according to the use to which the product is intended and define the cybersecurity functions implemented that make it more resistant to threats.

- IEC 62443-3-2 establishes the procedure to determine the zones, conduits, and risk assessment requirements. It defines five steps and the fifth one is to perform a detailed cyber security risk assessment process. For that, it proposes some possible risk assessment methodology, such as the ISO 27005 which is the most common for cybersecurity applications.
- IEC 62443-3-3: Establishes the security requirements for the system's architecture, design, and management, including the definition of the system's security zone and its relationship with other systems.

- IEC 62443-4: Component development, which establishes the specifications that must be included in the components of the cybersecurity industry, and the development life cycle that must be followed to develop these components.
  - IEC 62443-4-1: Product development requirements, which defines the cybersecurity development life cycle that new industrial control systems must follow, although they can also be applied to existing systems.
  - IEC 62443-4-2: Security Requirements, which establishes the seven requirements that a cybersecure industrial control system must have. Depending on the security level that a component has, certain requirements must be met, where the highest level is the one that must meet all of them.

8.3 Risk Management Methodology (ISO 27005)

According to IEC 62443 establishes that any detailed risk assessment methodology can be followed as long as the selected methodology satisfies the risk assessment requirements such us the ISO/IEC 27005, which It is a systematic establishment, assessment and treatment process of all risks associated and/or related to a given scenario or purpose.

The Risk Management process is divided in different stages, those are Context Establishment, Risk Assessment and Risk Treatment.

Context Establishment is needed to determine the environment and conditions in which the risk assessment takes place.

The STRIDE thread modelling is one widely used approach for identifying potential threats in the context establishment stage. This method involves considering six types of threats: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

Risk assessment process plays an important role in the cybersecurity management processes, since the identification and qualification of the security threats and risks is essential when it comes to the protection of assets. This task and responsibility shall be jointly addressed by all actors/entities involved in the cybersecurity management process of the system under consideration. For this purpose, three main activities are performed, Risk identification, Risk Estimation and Risk Evaluation.

From the risk assessment results, a risk treatment is generated. Risk Treatment is the process of selecting and implementing of measures to modify risk. Risk treatment measures can include avoiding, optimizing, transferring, or retaining risk[21]. It is a good part of the cybersecurity requirements to be met / developed in the product.

8.4 STRIDE

The STRIDE methodology is a tool created by Microsoft to identify potential threats related to computer security. It includes consideration of six types of threats represented by the acronym, Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. Each letter represents a different type of threat that can be used to identify potential vulnerabilities in a system. In addition to these, the methodology also includes consideration of five other elements, External Entities, Process, Data flow and Data Store.

External entities are actors or systems that interact with the system under consideration, such as users or other systems. Process represents the actions or activities that take place within the system, such as data processing and decision making. Data flow refers to the movement of data within the system, including input, output, and storage. Data Store represents the locations where data is stored and managed within the system, such as databases or file systems. Trust boundary refers to the point at which the system trusts the identity or actions of external entities and allows them access to resources or information.

The goal of using the STRIDE methodology is to identify all potential threats and vulnerabilities in a system, by considering these six types of threats and five elements. Once identified, appropriate countermeasures can be put in place to mitigate the risk of a successful attack. This can include implementing security controls, such as firewalls, intrusion detection systems, and encryption, as well as developing security policies and procedures to help prevent and respond to potential attacks.

8.5 Application of the methodology in the FRACTAL node

The methodology described has been applied in the development of the WP4T44-02 OS Security Layer component. This component implements the necessary security measures to protect the FRACTAL node and has been the result of the risk assessment process.

Firstly, the different use cases in which one or several FRACTAL nodes are applied have been analysed and a set of assets have been obtained. From these assets, a risk analysis has been carried out. These assets have then been analysed following

[21] https://www.enisa.europa.eu/topics/threat-risk-management/risk-management/current-risk/risk-management-inventory/rm-process/risk-treatment

the STRIDE methodology, resulting in a series of security countermeasures. These countermeasures have been implemented in a Yocto customization layer.

This entire risk analysis process is described in detail in chapter 6 of Deliverable D4.5.

In summary, the ISO 270005 methodology and the STRIDE process have been used to develop the WP4T44-02 OS Security Layer component. Likewise, this component is used on UC5. The applications, assets and risk analysis have been thoroughly described in the deliverable, ensuring that the necessary security measures are in place to protect the FRACTAL node.

9 Conclusions

In this deliverable, first, we have introduced the workflow of the project from capturing requirements to FRACTAL features definition and product composition. Then we introduced the methodology and workflow for AI and Safe Autonomous decisions, from different perspectives, starting from the framework to the services, libraries, strategies and algorithms.

This methodological framework also relates to the safety aspects to be considered in FRACTAL nodes, that can be addressed by a series of different safety standards depending on the target applications. Apart of the diversity of application specific standards to be considered we also have considered safety communications, safety aspects of Artificial Intelligence and how the building block approach can fit into the FRACTAL framework.

Integration aspects have also been considered. For this purpose, the methods that are used to integrate the use cases into the FRACTAL platforms have been introduced. First the operational integration was explained with an example use case, from the selection of features to the selection of components and how the building process would be done. Then, the SW integration on the HW platforms, considering several aspects for each specific HW.

Finally, an overview about the Risk Management Methodology that has been followed for the development of the risk analysis in WP4 is presented. where the need to develop secure and reliable systems is considered. In addition, the most important standards to be considered in these developments are presented.

A general methodology for FRACTAL system development has been presented here and has been used throughout the FRACTAL project. We described the overall workflow of the project with a focus on the identification of the key enabling technologies, the task of this deliverable, based on the use case descriptions.

10 List of Figures

11 List of Tables

12 List of Abbreviations

ACAP	Adaptive Compute Acceleration Platform (relates to VERSAL)
ACE	AXI Coherency Extensions
AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
AI	Artificial Intelligence
AIP	Artificial Intelligence Processor
AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric Multi-Processing
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASIC	Application Specific Integrated Circuits
ASIL	Automotive Safety Integrity Level
ASP	Application-Specific Processors
AXI	Advanced eXtensible Interface
BIST	Built-In Self Test
BPF	Band Pass Filter
CCF	Common Cause Failure
CLAHE	Contrast Limited Adaptive Histogram Equalization
CNN	Convolutional Neural Network
COTS	Component Off The Shelf
CPS	Cyber Physical System
CPU	Control Processing Unit
CSP	Concurrent Sequential Processes
DFD	Data Flow Diagram
DL	Deep Learning
DMA	Direct Memory Access
DNN	Deep Neural Network
DR	Diabetic Retinopathy
DRAM	Dynamic Random Access Memory
DoA	Description of Action
DS	Dedicated Systems
DSE	Design Space Exploration
DSP	Digital Signal Processor
DT	Decision Tree
ECC	Error Correction Code
EDDL	European Distributed Deep Learning Library
ERI	Energy Ratio Index
ESL	Electronic System Level
ETC	Energy-To-completion
EU	European Union

FIT	Failure In Time
FMC	FPGA Mezzanine Card
FODA	Feature-Oriented Domain Analysis
FPGA	Field Programmable Gate Arrays
GAN	Generative Adversarial Network
GCC	GNU C Compiler
GPP	General-Purpose Processor
GPU	Graphics Processing Unit
HD	High Definition
HDL	Hardware Description Language
HEPSYCODE	HW/SW CO-DEsign of HEterogeneous Parallel dedicated SYstems
HERO	Heterogeneous Research Platform
HLS	High-level synthesis
HPC	High Performance Computing
HPS	Heterogeneous Parallel Systems
HPV	Hypervisor technologies
HW	Hardware
IC	Instruction Count or Integrated Circuit (context dependent)
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IO	Input/Output
IoT	Internet of Things
IP	Intellectual Property
ISA	Instruction Set Architecture
ISO	International Organization for Standardization
JPEG	Joint Picture Action Group
JU	Joint Undertaking
KET	Key Enabling Technology
LEDEL	Low Energy DEep Learning Library
LIME	Local Interpretable Model-Agnostic Explanations
LLVM	Low Level Virtual Machine (compiler)
LR	Logistic Regression
MAC	Multiply-Accumulate
MFCC	Mel-Frequency Cepstral Coefficients
ML	Machine Learning
MLI	Maximum Likelihood Index
MLP	Multilayer Perceptron
MMU	Memory Management Unit
MoC	Model of Computation
MPSoC	Multi Processor SoC
NF	Non-Functional
NFC	Non-Functional Constraints

NoC	Network on Chip
NN	Neural Network
NP	Network Processor
NPI	NoC Programming Interface
OAA	One Against All
OAO	One Against One
ONNX	Open Neural Network eXchange
OS	Operating System
PCB	Printed Circuit Board
PE	Processing Element
PM	Probability Matrix
PULP	Parallel Ultra Low Power
QM	Quality Management
QoS	Quality of Service
R&D	Research & Development
RAM	Random Access Memory
REQ	Requirement
RF	Register File
RI	Reference Inputs
RIA	Research and Innovation Actions
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
RTL	Register Transfer Level
RTOS	Real Time Operating System
SBM	System Behaviour Model
SBS	System Behaviour Specification
SECDED	Single Error Correction Double Error Detection
SFI	Spectrum Flatness Index
SMP	Symmetric Multi-Processing
SoC	System-on-Chip
SOTIF	Safety of Intended Functionality
SPI	Synchronous Peripheral Interface
SPL	Software Product Line
SPP	Single-Purpose Processors
SSD	Single Shot Detector
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of Privilege
SU	Statistics Unit
SVM	Support Vector Machine
SW	Software
TL	Technologies Library
TRC	Time-To-Reaction
TTC	Time-To-Completion constraint

TTNoC	Time Triggered NoC
UART	Universal Asynchronous Receive/Transmit
UC	Use case
V&V	Verification and Validation
VAD	Voice Activity Detection
VCA	Video Content Analysis
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WP	Work Package
XAI	Explainable AI
YOLO	You Only Look Once

The short names of FRACTAL partners are not considered as abbreviations: ACP, AITEK, AVL, BEE, BSC, CAF, ETH, HALTIAN, IKER, LKS, MODIS, OFFC, PLC2, PROINTEC, QUA, ROT, RULEX, SIEG, SIEM, SML, THA, UNIGE, UNIMORE, UNIVAQ, UOULU, UPV, VIF, ZYLK.