



<b>Deliverable Reference</b>	: D4.2
<b>Title</b>	: Advanced CDFF Architecture and ICD
<b>Confidentiality Level</b>	: PU
<b>Lead Partner</b>	: SPACEAPPS
<b>Abstract</b>	: This Deliverable presents the Architecture Design of the Common Data Fusion Framework (OG3) and a description of the relevant interfaces (ICD) with other OGs
<b>EC Grant N°</b>	: 730014
<b>Project Officer EC</b>	: Christos Ampatzis (REA)



InFuse is co-funded by the Horizon 2020  
Framework programme of the European Union

## D4.2: Advanced CDFF Architecture and ICD

DOCUMENT APPROVAL SHEET			
	Name	Organization	Date
<b>Prepared by:</b>	Raul Dominguez Shashank Govindaraj Simon Lacroix	DFKI SPACEAPPS CNRS-LAAS	08/05/2017
<b>Updated by:</b>	Raul Dominguez Shashank Govindaraj Simon Lacroix Raul Dominguez	DFKI SPACEAPPS CNRS-LAAS DFKI	10/05/2017   13/09/2017
<b>Cross-reviewed by:</b>	Shashank Govindaraj Jeremi Gancet	SPACEAPPS	12/05/2017 15/09/2017

## Using InFuse Preliminary Design deliverables in Perspective of the H2020 Space Robotics Technologies SRC Call 2

### Overview of the Preliminary Design WP deliverables' content and purpose/logics within InFuse

**D4.1:** The goal of D4.1 is to present the choices of the data processing and data fusion functions that will be developed within InFuse.

After an analysis of operational scenarios which exhibit the required InFuse outputs, the list of these outputs ("data products") is defined. For each data product, a brief analysis of possible solutions is made, and a solution is selected. Each solution is defined by a "Data Fusion Process Compound" (DFPC), and is described with a coarse level of precision. The refinement of the DFPCs into a series of elementary "Data Fusion Nodes" (DFNs, i.e. elementary data processing functions), and the definition of the organisation of the DFNs to compose a DFPC is matter for further work for the InFuse project.

The DFPCs are defined for both the planetary and orbital reference implementations. While both implementations mostly require different DFPCs, many DFNs will be shared between the two targeted contexts.

The document also presents a sets of operational scenarios for each reference implementation. They have the purpose of putting in context the proposed DFPCs.

**D4.2 (this deliverable):** This deliverable focuses on the one hand on software architecture considerations, with the preliminary specification of all the key components that the InFuse CDFF consists of, and on the other hand on the identification of all relevant interfaces, both internal to InFuse and the external ones with respect to other OGs (OG2-ERGO and OG4-I3DS) in particular.

Note that the architecture and ICD material introduced in this document are essentially application independent – application specific considerations will be introduced at a later stage, during the detailed design of the InFuse CDFF (each of the orbital track and planetary track will then be tailored, accordingly).

This deliverable will serve as a starting point for the detailed design work, in the following work package.

**D4.3:** The purpose of this document is to define the strategy and overall approach for the testing activities to be carried out internally (OG3), so that to ensure that the developed software is sound while meeting the requirements expressed in the earlier phases of the project. The content is orbital / planetary independent: it is not in the scope of this deliverable to specify the testing approach with facilities and EGSEs specific to either the orbital or planetary scenarios. These aspects will instead be addressed in the detailed design activities that are just starting as this document is being written, and will be released by the CDR milestone.

### Relation between WP4 deliverable's content and other "common building block" OGs

#### D4.2: Advanced CDFF Architecture and ICD

---

The scope of OG3-InFuse lies essentially between OG4-I3DS, that produces raw sensor data, and OG2-ERGO, which controls all the rover activities. The interfaces with these two OGs are defined in the deliverable D4.2:

- The interfaces with OG4-I3DS are defined by sensor data types
- The interfaces with OG2-ERGO are defined on the one hand by the types of data products (mainly terrain maps and localization information related to the rover and the terrain, and to the chaser and target satellites), and on the other hand by requests made by OG2-ERGO to OG-InFuse.

OG2-ERGO is interfaced with OG3-InFuse at the granularity of the DFPCs: OG2-ERGO has no view of the inner mechanisms of OG3-InFuse that assembles DFNs into a DFPC. An OG2-ERGO request triggers the activation of a DFPC (which can either be synchronous or not), along with given parameters, and the DFPC returns the requested data products with an execution report. Within a DFPC, the DFNs are assembled, sequenced and triggered via pre-defined scripts, which are configured according to the parameters associated to the OG2-ERGO requests.

The definition of the DFPCs provided in D4.1 is generic, and makes no hypothesis regarding the integration middleware within which they will be developed. The way the developed functions will be integrated within the OG1-ESROCOS framework (and potentially other target middlewares) is depicted in the document D4.2.

Finally the content of D4.3 is largely centered on InFuse internal testing and validation activities – in that context, “integrated test plans” deal with the joint testing of several sub-parts of the InFuse framework, not InFuse and other OGs. Still, several components of the InFuse CDFF have interfaces with other OGs – mainly OG2-ERGO and OG4-I3DS. For these ones, it is foreseen in the test plan to develop specific components as placeholders of ERGO and I3DS, exposing the interfaces that are assumed to be the ones with which InFuse should integrate in the upcoming Space Robotics SRC projects. We call them M-OG2 and M-OG4 (M standing for Mock). Their purpose again is only to make it possible, internally, to carry out end-to-end tests and ensuring the soundness of the CDFF interfaces.

#### Applicability to the H2020 Space Robotics Technologies SRC upcoming calls (i.e. OG7 to OG11a/b).

Environment perception, be it to model the environment or to localize the controlled robotic platform within this environment, is at the core of autonomous operations, and is therefore required in all the applications defined by the upcoming calls.

For each of the future Operational Grants, a set of relevant DFPCs identified in D4.1 document is identified below as having particular relevance (note however that this list is based on very preliminary, and still limited knowledge of the content and scope of each upcoming OG).

**OG7: (Orbital Support Services):** Long/Mid/Close-range Tracking and Detection, 3D Target reconstruction, Point Cloud based Localisation

**OG8: (Modular Robotized Assembly):** Mid/Close-range Tracking and Detection, Point Cloud based Localisation

**OG9: (Satellite Re-configuration):** Mid/Close-range Tracking, Point Cloud based Localisation

## D4.2: Advanced CDFF Architecture and ICD

---

**OG10: (Advanced Autonomy):** DEM Building + Soil Type Map, and all the rover localisation DFPCs: Visual Odometry, Visual/LIDAR based SLAM, Scientific Area Localisation, Visual Map-based Localisation, Absolute Localisation.

**OG11a: (Advanced Mobility):** DEM Building + Soil Type Map and Visual Odometry for extreme terrain mobility, plus all the localisation DFPC for the coordination of multiple platforms.

**OG11b: (Robotized Construction):** The required DFPCs for this OG are certainly similar to some of the ones required for the orbital OGs: Long/Mid/Close-range Tracking and Detection, 3D Target reconstruction, Point Cloud based Localisation – though in planetary context. DEM building and rover localization DFPCs remain relevant.

Deliverable D4.2 provides the latest baseline about the InFuse CDFF architecture and ICD. In perspective of the next OGs, it is essential to understand the proposed architecture and mechanisms to handle data, and the proposed approach to generate middleware specific reference implementations from the vanilla (middleware independent) CDFF environment.

In perspective of the upcoming OGs, D4.3 has limited relevance as its purpose is essentially to define the InFuse internal strategy to test the various components of the CDFF. It is therefore of little use for what concerns the preparation of bids for the next call, and similarly would have limited relevance for the future implementation of the next OGs.

## Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>12</b>
1.1	Purpose and scope .....	12
1.2	Document structure.....	12
1.3	Applicable documents.....	12
1.4	Reference documents.....	12
1.5	Acronyms .....	13
<b>2</b>	<b>Architecture Design Rationale .....</b>	<b>14</b>
2.1	CDFF objectives.....	14
2.1.1	Data Products Management and Orchestration.....	14
2.1.2	RCOS Independence .....	14
2.2	Architecture Design Process.....	15
<b>3</b>	<b>Proposed architecture.....</b>	<b>16</b>
3.1	CDFF-Core.....	17
3.2	CDFF-Support.....	18
3.3	CDFF-Dev .....	18
<b>4</b>	<b>Detailed description.....</b>	<b>20</b>
4.1	CDFF-Core.....	20
4.1.1	Data Fusion Node Common Interface .....	20
4.1.1.1	Metadata Handling .....	22
4.1.2	Nodes to be included .....	23
4.1.2.1	Feature Detection Nodes .....	24
4.1.2.2	Recognition and Registration Nodes.....	24
4.1.2.3	State Estimation Methods .....	24
4.1.2.4	Data Filtering and Outlier Removal Nodes .....	24
4.2	CDFF-Support.....	25
4.2.1	Data Fusion Processing Compound .....	26
4.2.1.1	Example of a data Fusion Processing Compound and the Description File.....	26
4.2.2	Orchestrator .....	28
4.2.3	Data Products Management Tool .....	29
4.2.3.1	Motivations .....	29
4.2.3.2	DPM implementation .....	30
4.3	CDFF-Dev .....	30
4.3.1	CDFF-Dev Common Interface .....	30
4.3.2	Middleware Facilitator .....	31

## D4.2: Advanced CDFF Architecture and ICD

4.3.3	Logs and Data Flow Management .....	33
4.3.3.1	Python Bindings for EnviRe .....	34
4.3.4	Data Analysis and Performance Tools.....	35
4.3.5	Visualization Tools .....	39
<b>5</b>	<b>Interface Control Document .....</b>	<b>42</b>
5.1	OG3 Product Definition .....	42
5.1.1	Identification of OG3 Interactions with other OGs.....	42
5.1.2	Data Products Generated by OG3.....	44
5.1.3	Planetary reference implementation .....	44
5.1.4	Orbital reference implementation .....	45
5.2	Internal Interfaces .....	46
5.2.1	Data Product Storage.....	46
5.2.2	Data Product Retrieval .....	46
5.2.3	Activation DFPCs .....	47
5.3	Interface with OG1 .....	48
5.3.1	I/F Requirements.....	49
5.4	Interface with OG2 .....	51
5.4.1	I/F Requirements.....	52
5.5	Interface with OG4 .....	55
5.5.1	Sensor Data Inputs from OG4.....	56
5.5.2	Interfaces Requirements .....	66
5.6	Rover and Manipulation Interface.....	71
5.6.1	Sensor Data Inputs from OG6.....	71
5.6.2	I/F Requirements.....	73
<b>6</b>	<b>Conclusion .....</b>	<b>74</b>
	<b>Appendix 1: Glossary .....</b>	<b>75</b>
	<b>Appendix 2: InFuse CDFF Product Tree .....</b>	<b>77</b>
	<b>Appendix 3: Libraries and Data Types.....</b>	<b>78</b>
	<b>Appendix 4: Requirements Traceability Matrix.....</b>	<b>79</b>
	CDFF Core.....	79
	User Requirements (UserR) .....	79
	Functional Requirements (FuncR) .....	80
	Performance Requirements (PerfR) .....	82
	Interface Requirements (IntR).....	83
	Resource Requirements (ResR).....	84
	Operational Requirements (OpR) .....	84
	Product assurance and safety requirements (ProdR).....	85

---

**D4.2: Advanced CDFF Architecture and ICD**


---

Configuration and implementation requirements (ConfR) .....	85
Test and Validation (ValR) .....	85
CDFF Dev .....	86
User Requirements (UserR) .....	86
Functional Requirements (FuncR) .....	87
Performance Requirements (PerfR) .....	88
Interface Requirements (IntR).....	88
Resource Requirements (ResR).....	88
Operational Requirements (OpR) .....	89
Product assurance and safety requirements (ProdR).....	89
Configuration and implementation requirements (ConfR) .....	89



## List of Figures

Figure 1: InFuse approach towards the development and deployment of data fusion solutions. On the developer's environment a set of utilities for design and evaluation are available. The developed perception solution is design to be easy to integrate in any target system, independently of the RCOS and the specific hardware. ....	16
Figure 2: Currently Components performing a same task and using the same libraries might have implemented at the RCOS dependent level many logic. ....	20
Figure 3: Proposed approach towards RCOS independent Data Fusion Processing Nodes: Component logic should be moved to the library level and a common interface (DFNCI) will facilitate the integration into different RCOSs. ....	21
Figure 4: Artifacts of a Data Fusion Node Common Interface (DFNCI). Information in the Interface Description will ease the process of integration in the target RCOS as well as in orchestrated DFPCs. ....	22
Figure 5: Data Fusion Nodes can be set to listen at the Data Products generated by other Data Fusion Nodes and associate to their output additional metadata. The Data Fusion Node that listens has implemented decision logic on when a data product should be incorporated with metadata. The output of the listener will contain the Data Product or a reference to it and relevant information about it (e.g. spatiotemporal, source, processing steps gone through). ....	23
Figure 6: Data Fusion Nodes can be set to monitor parts of the Data Fusion Process in different ways. In this case the Data Fusion Node 06 listens to three Data Product streams. ....	23
Figure 7: The diagram presents how the three components of the CDFF-Support interact and also the interfacing with OG2 and OG4. The <i>orchestrator</i> (1) performs request to the Data Products Manager based on what OG2 requires (2) sets the operational modes for the sensors and (3) activates or deactivates DFPCs. The <i>Data Product Manager</i> (1) stores under request the data products generated in the DFPCs and (2) retrieves under request data products for the Orchestrator or the DFPCs. ....	25
Figure 8: This figure describes an example of a DFPC, two data sources (laser and dynamixel) connect with two DFNs. The data product that is generated is pointcloud and the input data types are 2D scan samples and servo status (including motor position). ....	27
Figure 9: Internal software components of the Orchestrator and its interactions with other CDFF and external software modules ....	29
Figure 10: An example of Python bindings generated for a DFN. A node that implements the DFNCI is displayed on the left side. On the right side, the auto completion from the IPython REPL is used to display the methods in Python. ....	31
Figure 11: Terminology: MW Component - Library(s) + mw wrapper (system level process); Task - Threads running within the MW Component; DF-Lib - one or more DF algorithm implementations with appropriate interfaces ....	31
Figure 12: The DFNCI template for a DFN will be generated from the node definition file. In addition, we can generate templates for RCOS components and a Python wrapper. ....	32
Figure 13: Class diagram of a node that implements the DFNCI. A Python binding and a template for a Rock component could be generated automatically. Red color indicates files that will be implemented by the developer. ....	33
Figure 14: Conversions from RCOS-specific log formats to an intermediate log format that is independent of the RCOS have to be implemented. On the side of the DFNCI, conversions to the data types of the library have to be implemented. ....	34
Figure 15: Sequence diagram that shows how a user could interact via REPL with the log player and the data flow control. ....	34

## D4.2: Advanced CDFF Architecture and ICD

Figure 16: Visualization of pySPACE node chains. pySPACE can distribute the execution of independent node chains that will be compared over different processing units, e.g. CPUs. ....	35
Figure 17: pySPACE provides a large number of nodes for preprocessing, feature generation, classification, visualization, etc. ....	36
Figure 18: pySPACE offers various metrics and visualization tools to compare different machine learning and signal processing methods.....	37
Figure 19: pySPACE will be used as a tool to evaluate signal processing and machine learning nodes that will be deployed on the robot. ....	38
Figure 20: Class diagram that shows how a hypothetical algorithm <i>MyAlgorithm</i> can be integrated in pySPACE as a pySPACE node. ....	39
Figure 21: The 2D representation of the Envire graph can be visualized using Envire Visualization. The 2D view presents all the transformation as well as the objects stored in each frame with their correspondent type name. In this image a robot model is visualized. ....	40
Figure 22: The 3D representation of the Envire graph can be visualized using Envire Visualization. The different transformations, frames as well as the objects contained them can be inspected. In this visualization the visual components of a robot model are shown. ....	41
Figure 23: The figure above allows to identify the potential set of interactions between OG3 on the one hand and OG2 and OG4 on the other hand. OG4 is understood as including a hardware part – i.e. the physical sensors, and a software part consisting of the drivers and pre-processing capabilities included in I3DS.....	42
Figure 24: Summary of the interactions of OG3 with other OGs .....	43
Figure 25: The internal interfaces are (1) Data Product Retrieval and (2) Data Product Storage implemented by the Data Product Manager and (3) Activation DFPCs implemented by the DFPCs ..	46
Figure 26: Workflow to integrate OG3 in OG1 .....	48
Figure 27: Communication workflow between OG2 and OG3. On the left diagram the main operational loop is presented. On the right one, the operation to stop a request.....	51
Figure 28: Communication workflow between OG3 and OG4.....	55
Figure 29: Example of a DFPC. A DFPC receives sensor data as inputs (provided by OG4) and produces Data Fusion Products (delivered to OG2 by the Orchestrator). ....	76
Figure 30: Product Tree of InFuse Common Data Fusion Framework.....	77

## List of Tables

Table 1: List of features implemented in the CDFF. CDFF-Core and CDFF-Support incorporate all features that are included both in the target system and in the developer's environment. CDFF-Dev includes features which are useful for the developer to design a data fusion processing solution as well as to evaluate existing ones using logged data. None of the features of CDFF-Dev will be deployed in the target system.....	17
Table 2: Interface Data Product Storage implemented by the Data Products Manager .....	46
Table 3: Interface Data Product Retrieval implemented by the Data Products Manager .....	47
Table 4: Interface of the DFPCs for the Orchestrator .....	47
Table 5: Sensor data inputs from OG4 .....	64
Table 6 Metadata Data Types.....	65
Table 7 Data inputs from the Rover and Manipulation interface.....	72
Table 8: Tentative list of libraries and type definitions that will be used in InFuse .....	78

# **1 Introduction**

## **1.1 Purpose and scope**

This document aims to provide an advanced review of the architecture of the Common Data Fusion Framework as well as the latest assumptions about the interfaces to inside and outside components.

At this stage of the design, a good level of refinement was introduced as to which specific data products will be generated, how they will be handled, which core libraries will be integrated as well as how the interactions with the other OGs will exactly be.

Note that is not in the scope of this document to detail the data fusion capabilities of InFuse, which are rather covered by D4.1.

## **1.2 Document structure**

In brief, the document is structured as follows:

Section 1: This introductory material.

Section 2: Architecture design rationales

Section 3: Proposed architecture

Section 4: Detailed description

Section 5: Interface Control Document

Section 6: Conclusion

## **1.3 Applicable documents**

AD1 InFuse Grant Agreement (CO document)

AD2 InFuse Consortium Agreement (CO document)

## **1.4 Reference documents**

RD1 D3.1: Technological review (PU document, available on demand)

RD2 D3.2: System Requirements and Operational Concept (PU document, available on demand)

RD3 D3.3 Architecture Design and ICD Early Iteration (PU document, available on demand)

## **1.5 Acronyms**

API: Application Program Interface

CDFF: Common Data Fusion Framework

DEM: Digital Elevation Map

DPM: Data Product Management

DF: Data Fusion

DFM: Data Flow Management

DFN: Data Fusion Node

DFNCI: Data Fusion Node Common Interface

DFPC: Data Fusion Process Compound

GIS: Geographical Information System

ICU: Instruments Control Unit

IMU: Inertial Measurement Unit

LDFM: Logs and Data Flow Management

RCOS: Robotics Control Operating System (e.g. ROS, Rock, GenoM)

MW: Middleware. In this document is used as synonym for RCOS.

PCL: Point Cloud Library

REPL: Read Eval Print Loop

RI: Reference Implementation

SLAM: Simultaneous Localization and Mapping

TBD: To be defined

## 2 Architecture Design Rationale

### 2.1 CDFF objectives

The main objective of the CDFF is to provide a framework for the development, evaluation and deployment of space robotics data fusion technologies in the context of the PERASPERA cluster of project.

For the case of the relation between InFuse and ESROCOS this means that any data fusion process developed using InFuse should be portable to ESROCOS with minimal efforts. This also means that within InFuse it would be redundant to implement another RCOS or aim for taking decisions that are within the realm of the RCOS (e.g. processes priorities). Similarly, InFuse does not take decisions on the sequencing and control of its processes, as these pertain to the ERGO autonomy framework. Yet, InFuse has mechanisms to select and parametrize its processes on the basis of requirements expressed by ERGO. Finally regarding our relationship with the sensors system (OG4) InFuse does not pursue to control the internal state of the sensors but rather knows a set of operational modes that can be accessed and which provide data streams with different characteristics.

#### 2.1.1 Data Products Management and Orchestration

*InFuse has the focus set on the generation and management of a consistent and coherent environment representation which can be accessed externally.* In this direction the *Data Products Management* stores in a consistent manner the environment representation data products and the *Orchestrator* analyzes which of the available sensor's operational modes and data fusion processes are the most adequate to successfully process data product requests.

#### 2.1.2 RCOS Independence

*InFuse provides a set of libraries and development tools to support the work done by data fusion solutions developers. InFuse aims for RCOS and robotic platform independence but pursues an easy transition to the target RCOS in various ways (e.g. Compatible interfaces, provision of deployment recommendations and middleware facilitators for partially automating deployment in a target RCOS).* The decision of implementing the current set of utilities and not a complete inter-process communication and logging mechanism was taken after analyzing what could be useful for robotics perception researchers, the context of OG3 - responsibilities with respect to OG1 - and the amount of available resources.

Researchers on perception often have the goal of analyzing how a given algorithm or library performs on given logged data, most often using multiple specific parameterizations. For such analyses, the researcher is normally not interested on the details of the platform it could run on, or the target RCOS due to the inherent independence of the algorithm or library itself. Currently, one of the problems a researcher is facing when evaluating algorithms is the requirement of installing and running a RCOS (which might not be stable), taking the developer into long periods of working out of his realm of interest and expertise. *InFuse aims to facilitate the data fusion expert's work by providing a framework to evaluate data fusion solutions with little effort in a simple as possible environment.*

The transition between the developer's environment and the target system raises challenges which are platform and RCOS specific. Thus, this is out of the scope of the project as we aim for RCOS independence. It is arguable that it is possible to ease even more the work in this direction by providing a more realistic communication system between data fusion nodes so as to bring it closer to the final RCOS setup but this would imply the simulation of multiple RCOS communication mechanisms in InFuse, and furthermore the final system's communication will still require testing. It is also possible to think of solutions which could be directly deployed in target systems without any transition, but again

## **D4.2: Advanced CDFF Architecture and ICD**

---

these are RCOS specific solutions which, given the stage of OG1 development, does not seem realistic (e.g. some deployment steps in TASTE have to be done currently via GUI).

### **2.2 Architecture Design Process**

The process of design has followed these stages: First, the requirements of the framework in its designer environment as well as for the deployed data fusion solution were collected. Second, an initial components diagram of the overall architecture was designed. Third, the design was further detailed by designing the sequence diagrams of some of the identified use cases.

In the second iteration of the architecture design further insights on the implementation of the use cases are done by focusing on the design and the deployment of a Data Fusion Processing Compound for a scenario of long term navigation.

The process of designing the architecture started with collection of the requirements which was started in the previous work package (refer to D2.2). In parallel many concepts were discussed to achieve a common understanding among the members of the consortium. As result of the discussions among other documents, a glossary incorporating all the important concept to understand the architecture designed has been produced.

Once the requirements were identified, the initial phase of architecture design produced a sketch of the system architecture in a components diagram. This represents software modules in an organized manner so that each module has a particular identity defined by the features it provides to the overall system.

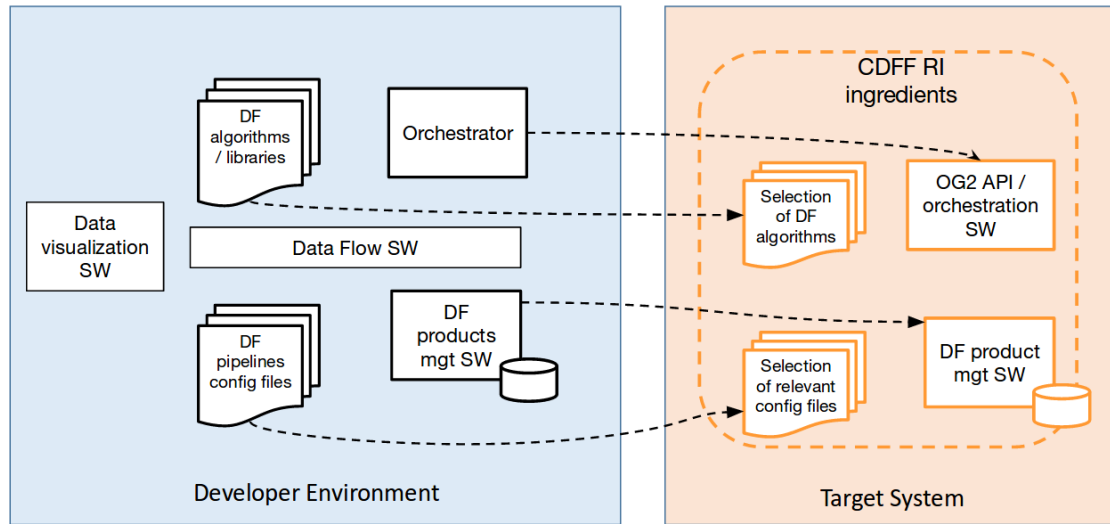
The initial architecture is a sketch and a more in depth insight to how the requirements are to be provided must be done. For this purpose, use cases which require multiple functionalities from different components were selected, designed and analyzed with the partners. This process leads to a deeper understanding of what to do and how to approach the software implementation.

In order to further validate the approach and the adequacy of third party tools, critical use cases have been selected and prototyped to verify the validity of the proposed architecture. This approach aimed to identify at early stages of the development the real efforts to implement the architecture features, and to find the most appropriate way to implement them before completely defining the architecture.

Another important part of the design of the architecture is the definition of the interfaces with other elements of the final system. In the case of InFuse these are the autonomy framework (OG2), the sensors suite (OG4) and the RCOS (OG1). In this context various meetings were held with the corresponding OG representatives. As result of these meetings, in the first iteration of the architecture design phase an initial ICD was defined. With respect to the integration with OG1, the interfacing is of a different nature as the other ones. In this case, the interface defines how any solution defined by the Common Data Fusion Framework can be ported to the target RCOS. Discussions with the leading partner from OG1 helped to understand the data types that will be supported in the communication between processes of the target Robotic Framework.

### 3 Proposed architecture

The proposed process of developing data fusion solution solutions consist of two steps: (1) development and initial evaluation on the developer's environment and (2) deployment and testing on the target system. These two steps can be reiterated: after evaluating the initial solution on the target system, the logs generated by the target system's RCOS logging module can be used in the developer's environment to analyze and improve the solution.



**Figure 1: InFuse approach towards the development and deployment of data fusion solutions. On the developer's environment a set of utilities for design and evaluation are available. The developed perception solution is design to be easy to integrate in any target system, independently of the RCOS and the specific hardware.**

An important distinction must be done between features implemented by components that belong to the Developer's Environment uniquely (CDFF-Dev) and components that belong to both the Developer's Environment and the Target System (CDFF-Core and CDFF-Support). In Table 1 a comprehensive list of the different component and the set to which they belong is provided.



## D4.2: Advanced CDFF Architecture and ICD

Feature	CDFF-Core	CDFF-Support	CDFF-Dev	On Target System	On Developer Environment
Core Libraries	X			X	X
Common Interface (C++ Level)	X			X	X
Data Fusion Processing Compound		X		X	X
Orchestrator		X		X	X
Data Product Management Tool		X		X	X
Common Interface (Python Level)			X		X
Middleware Facilitator			X		X
Logs and Data Flow Management			X		X
Visualizer			X		X
Data Analysis and Performance Tools			X		X

**Table 1: List of features implemented in the CDFF. CDFF-Core and CDFF-Support incorporate all features that are included both in the target system and in the developer's environment. CDFF-Dev includes features which are useful for the developer to design a data fusion processing solution as well as to evaluate existing ones using logged data. None of the features of CDFF-Dev will be deployed in the target system.**

InFuse is not conceived to be an RCOS: some features that an RCOS deal with are left aside: realtime communication layer, hardware interaction, intelligent processes deployment... In our view these are features that the RCOS should provide and eventually will aim at providing, as it is the only subsystem with the complete picture of the system running components. Nevertheless, the developer can provide helpful information in the Configuration Files to facilitate a smart deployment while the DFN Common Interface (DFNCI) is conceived to ease the integration process in different RCOS.

### 3.1 CDFF-Core

The CDFF-Core provides the "core" data fusion set of state of the art algorithms and techniques currently in use and applicable to the Planetary and Orbital RIs within the scope of InFuse, as a collection of ready-to-use libraries or packages. These libraries represent the CDFF's processing methods necessary to fuse sensory data. The CDFF-Core is to be implemented in a modular fashion with a

## D4.2: Advanced CDFF Architecture and ICD

---

common interface to allow high flexibility in configuration as well as in operation as a distributed system on multiple platforms. The core libraries are to be deployed on the **target system** (robotic platform) as well as on the **designer's environment**.

Examples of core libraries include low-level functions such as feature detection, registration and recognition, data association, state estimation, outlier removal, and filtering as well as building environmental representations, achieving 3D object reconstruction or SLAM...

### 3.2 CDFF-Support

The CDFF-Support provides the necessary tools for the instantiation and execution of Data Fusion Processing Compounds. Under the CDFF-Support functionalities can be found the Data Fusion Processing Compound (DFPC), the Orchestrator, and the Data Product Management Tool (DPM). CDFF-Core entities will be deployed with no change into the final target system. All components of CDFF-Support will be deployed in the Target System and are also available for programming and testing in the Developer Environment.

1. The **Data Fusion Processing Compound** (DFPC) is a combination of several nodes designed to provide a certain data product (e.g. pose estimation, map...). The DFPC defines the connections between input and output ports of different data fusion nodes.
2. The **Orchestrator** is the component that deals with the activation and deactivation of DFPCs and configures the data connections between the nodes. It is also the component that receives and answers the requests from the Autonomy Module (OG2). The selection of one or another DPCH is done based on availability and quality of data sources and on the data product required.
3. The **Data Product Management** (DPM) is a tool which acts as a long term memory for the data fusion products being generated by DFPCs to be used by OG2 or other DFPCs.

### 3.3 CDFF-Dev

The CDFF-Dev provides the tools to develop, test, visualize, and perform analysis on data fusion products. The CDFF-Dev include tools such as a DFPC inspector, data log replay, visualization tools, and data analysis tools. None of the components of the CDFF-Dev are deployed on the target system.

1. The CDFF-Dev **Common Interface (Python level)** are python bindings provided for the DFN common interface. This provides the developer the possibility to evaluate the data processing solution (or library) directly from Python without the need to port the solution to a specific RCOS.
2. The **Middleware Facilitator** provides the CDFF the capability to partially convert a DFPC from the designer's environment in the corresponding DFPC on the target RCOS. This utility is foreseen to provide a nominal level of corresponding target RCOS specific wrapper code for the CDFF-Core and CDFF-Support components.
3. The **Logs and Data Flow Management** is a tool that allows to inject logged data to a data fusion process in a chronological fashion. It is envisioned to allow loading of log data produced by different RCOS<sup>1</sup>. It replaces the communication layer of the RCOS but as this is not a tool that will be deployed in the target system, there are no requirements regarding real time communication. Nevertheless, the data will be timestamped so that a realistic estimation of the

---

<sup>1</sup> To achieve this, specific converters for each RCOS to the CDFF types have to be implemented. In the course of the project, converters to ROCK will optionally be implemented.

#### D4.2: Advanced CDFF Architecture and ICD

---

processing time required by the DFNs is available to the designer. The outputs of the nodes in the DFPC are also stored in logs which are used as input for posterior nodes in the processing compound.

4. The **Data Analysis and Performance Tools** are comprised of statistical analysis tools and graphical representations necessary for comparison of data fusion products resulting from different processing chains. The goal is to provide the designer methods to assess the quality of the data products and of the DFPC. The logs of the internally generated data are for example useful to compare data products generated using different DFPCs.
5. The **Visualizer** is responsible for presenting graphical representations of the different data products (e.g. 2D/3D plots, maps, camera images...).

Note that during the design of the architecture other features were identified with potential interest, which could be implemented in the future. For instance, the **Data Fusion Processing Compound Configurator** is a tool that enables the designer to define data fusion processing compounds (i.e. DFNs used, order, connections, frequencies, priorities etc.). The result of using this tools is a DFPC Configuration File. This configuration file is then used by the Middleware Facilitator to generate the correspondent DFPC for the target RCOS. The DFPC Configuration File is also required to design the operation logic of the Orchestrator which might handle multiple DFPCs. CDFF-Dev could provide in the future monitoring of the execution time and memory consumption of individual components in order facilitate the design of an embedded DFPC.

## 4 Detailed description

In the following sections a more detailed review of the already presented components is presented.

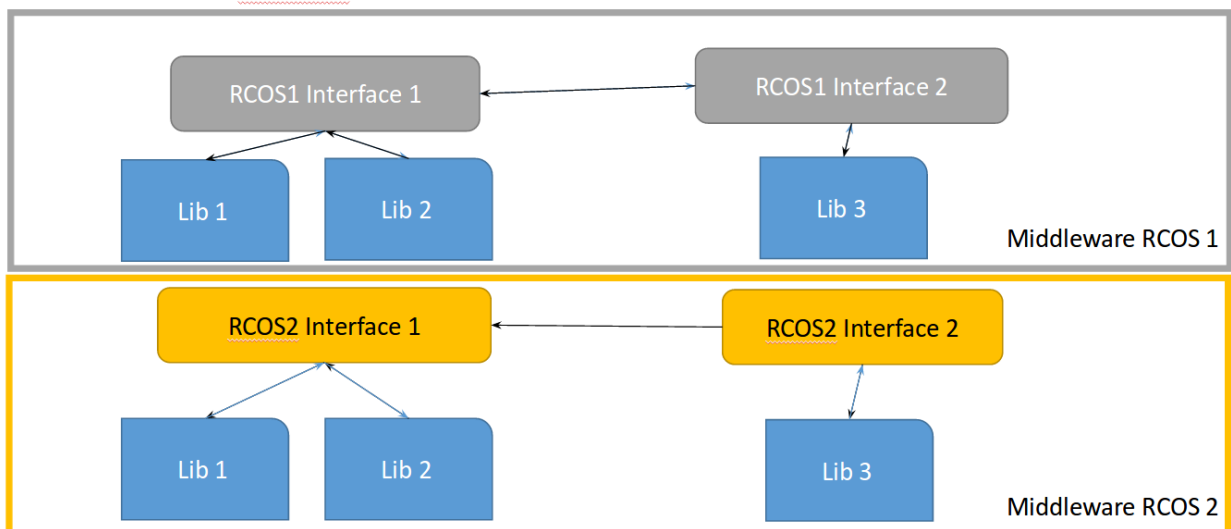
### 4.1 CDFF-Core

#### 4.1.1 Data Fusion Node Common Interface

The Data Fusion Node Common Interface (DFNCI) provides the abstraction layer to encapsulate each data fusion node internal processing. The DFNCI is designed to be general enough to standardize all cases of data fusion processing nodes, but on the other hand, it is also specific for the context of perception and pose estimation which are at the core of data fusion in robotics.

In InFuse, the aim is to provide a RCOS independent framework. Thus, it is aimed to leave as much as possible RCOS dependent features out of the DFNCI. On the other hand, it has been kept in mind that any final perception solution developed and evaluated will eventually be deployed in a target RCOS. The DFNCI eases the RCOS integration process by capturing information regarding the DFN libraries and provides a standardized interface to harmonize different libs for future porting to target middleware and test using different programming languages and test utilities (e.g. Python Pandas).

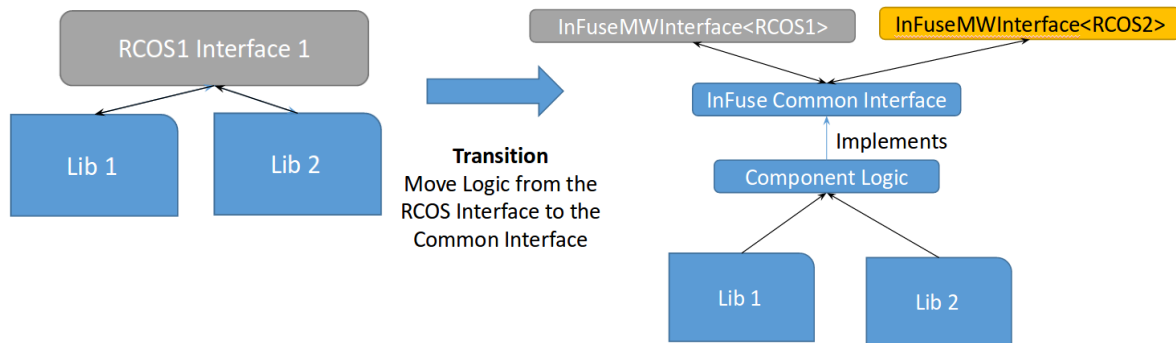
At the moment, RCOS components tend to have a dependency on the framework in which they were initially developed. Furthermore, a component performing the same task might be available in more than one RCOS, with different interfaces, and implemented with an ad-hoc glue code and logic. Our approach to achieve framework independence is to move the logic from the RCOS interface to the library level and to provide the generation of the glue code as much as possible automatically and supplemented by the developer.



**Figure 2: Currently Components performing a same task and using the same libraries might have implemented at the RCOS dependent level many logic.**

For CDFF-Core, current RCOS-specific implementations of data fusion nodes will be replaced by an RCOS-independent implementation that contains the component logic. The component logic will expose a common interface that can easily be wrapped with code for the target RCOS ( e.g. for type conversions in any RCOS). The component logic will also contain additional glue code for inter-library method invocations, data sharing etc.

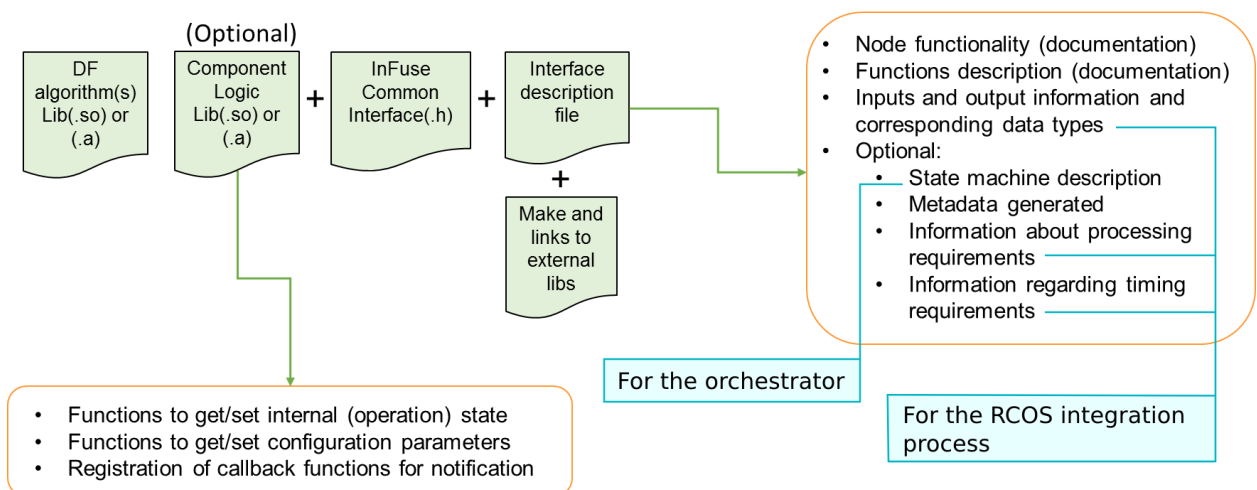
## D4.2: Advanced CDFF Architecture and ICD



**Figure 3: Proposed approach towards RCOS independent Data Fusion Processing Nodes: Component logic should be moved to the library level and a common interface (DFNCI) will facilitate the integration into different RCOSs.**

The DFNCI includes:

- Methods to receive and output data
- Methods for configuration of internal DFN parameters (optional)
- Methods for handling of Meta-Information of the processed data (optional). We consider these three as the minimally required metadata fields:
  - Spatiotemporal information (transformation and timestamp)
  - Source information (sensor + parameters)
  - Processing steps gone through
- Allows to capture information regarding processing requirements.
- Allows to capture information regarding timing requirements (e.g. periodicity, synchronicity)
- Provides optional interfaces for control of the internal states by the orchestration



**Figure 4: Artifacts of a Data Fusion Node Common Interface (DFNCI). Information in the Interface Description will ease the process of integration in the target RCOS as well as in orchestrated DFPCs.**

#### **4.1.1.1 Metadata Handling**

The three main information that the metadata provides along with the data products are: (1) *what*: type of the data product and what it refers to - this comes along with the necessary information to process or fuse the data, e.g. calibration parameters, (2) *when*: date of data used have been acquired, and (3) *how*: the DFN and DFPC that produced it. The first two types (*what* and *when*) of metadata are mandatory for the DFNs and the DPM to properly handle the data, while the third type (*how*) would rather be needed for execution control (orchestration) purposes, and certainly goes beyond the requirements of the CDFF architecture. Note also that the metadata could include more information depending on the application scenario, their definition and representation being to be defined by the final developer.

In InFuse we evaluate various potential approaches to manage the metadata and currently not a final decision has been taken to which one will finally be established. The following options are foreseen:

##### **Option 1: Metadata in the data types**

In this approach the metadata is already part of the types that the CDFF uses. The convenience of this approach lies in the ease with which the metadata can be manipulated in and accessed for every data product without the need for implementation of any additional metadata communication infrastructure. The main drawback that we see is that the incorporation of this metadata might not be well accepted by the maintainers of the types as it includes data that for other applications could be of no use.

##### **Option 2: Compositions of data types**

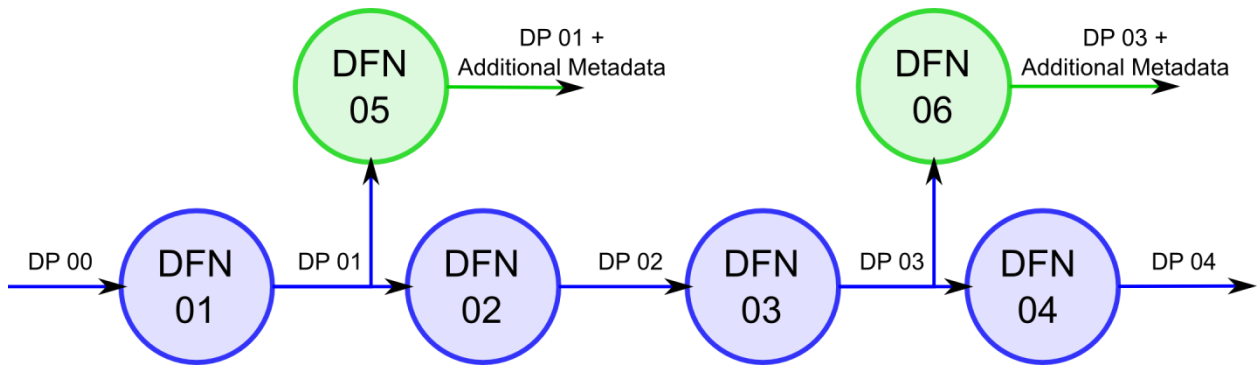
In this approach EnviRe could be used to implement the associations of data and metadata. EnviRe can associate multiple objects to a frame and in this frame store also structures defining relations of the objects in the frame. This solution is entire applicable internally in OG3 and would not imply any addition of data to the base types, but the use of this approach will be more complex when communicating this data through the middleware of OG1. Nevertheless the problem could be tackled.

##### **Option 3: DFN for metadata association**

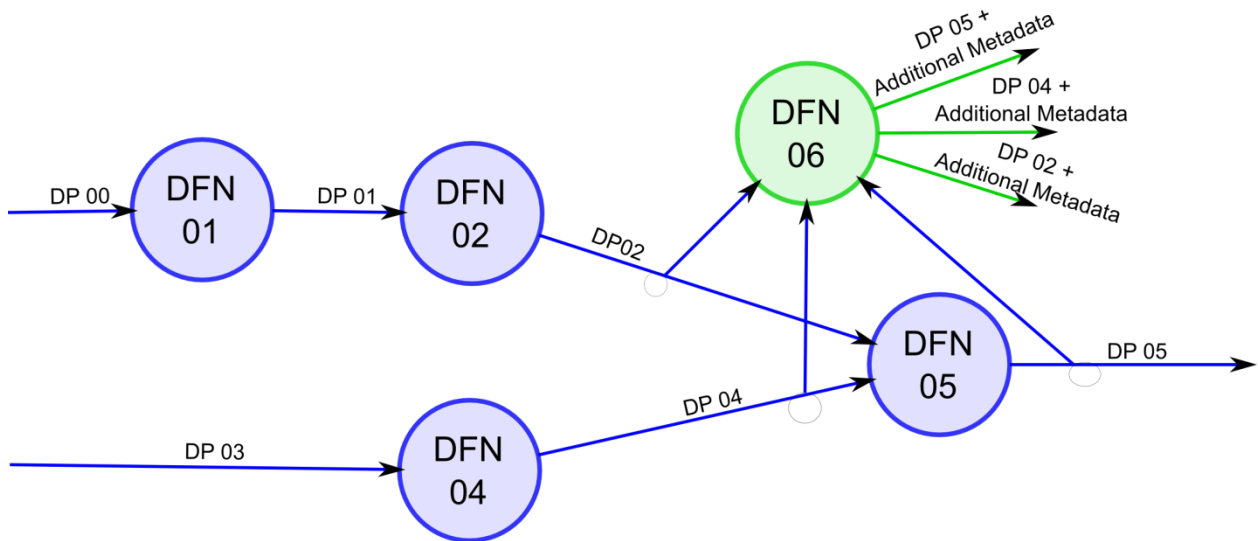
In order to facilitate the integration of metadata along with the normal data products in a flexible and non resource consuming manner (i.e. minimum computations and communications), the generation and association of additional metadata (i.e. not included in the data type itself) could be done as an optional feature that any DFN can implement.

The metadata would be generated by data fusion nodes which will read streams of data and will generate -according to its internal implementation- the metadata associated to the data product or products. This approach facilitates the inclusion of metadata where the developer might want to add it, and also when it is relevant to generate (e.g. continuously, only when certain object is recognized...). Furthermore, this approach encapsulates additional handling from all other data management. This could be beneficial for instance for optimization. Figure 5 and Figure 6 present two data flow examples of how the MetaData Fusion Nodes can be set.

## D4.2: Advanced CDFF Architecture and ICD



**Figure 5: Data Fusion Nodes can be set to listen at the Data Products generated by other Data Fusion Nodes and associate to their output additional metadata. The Data Fusion Node that listens has implemented decision logic on when a data product should be incorporated with metadata. The output of the listener will contain the Data Product or a reference to it and relevant information about it (e.g. spatiotemporal, source, processing steps gone through).**



**Figure 6: Data Fusion Nodes can be set to monitor parts of the Data Fusion Process in different ways. In this case the Data Fusion Node 06 listens to three Data Product streams.**

### 4.1.2 Nodes to be included

A large set of algorithms and technologies are available for inclusion in DFNs. To select the most appropriate and effective set of algorithms, test implementations will be created from existing code where available and from scratch where there is no available code. Test scenarios will create data for each type of algorithm as listed below. The most suitable methods for each application will be selected for C++ implementation as a data fusion node. In the ideal case, one or two algorithms will be included as the primary fusion in each category and others can be added optionally. Performance comparisons could be done in pySPACE in the Developers Environment -not directly on the target system- once actual outputs have been logged and assuming expected ones are also available.

---

## **D4.2: Advanced CDFF Architecture and ICD**

---

The final list of DFNs to develop and integrate within InFuse will be defined after the precise analyses and trade-offs initiated in [RD5], which will be refined in WP5, tasks 5.1 and 5.2 (detailed specification of the orbital and planetary RIs). The following lists the DFNs that will most likely be integrated.

### **4.1.2.1 Feature Detection Nodes**

The following feature detection methods will be evaluated for inclusion in nodes:

- Hough Transform
- Harris Detector
- ORB
- Optional: SIFT
- Optional: SURF

For three dimensional point clouds the following feature detection methods will be evaluated:

- Harris 3D
- SHOT (Signature of Histogram of Orientations)
- Optional: PFH/FPFH
- Optional: SURF 3D

### **4.1.2.2 Recognition and Registration Nodes**

The following recognition and registration methods will be evaluated:

- ICP
- RANSAC
- Optional: Levenberg-Marquardt

In addition, the following general data association techniques will be evaluated:

- K-Nearest Neighbors
- Linear Classifier
- Bayesian Classifier
- Dense registration

### **4.1.2.3 State Estimation Methods**

The following non-probabilistic state estimation methods will be evaluated:

- Optical Flow Estimation
- Fuzzy Logic
- Dempster-Shafer
- Optional: Dezert-Smarandache

The following probabilistic state estimation methods will be evaluated:

- Kalman Filter
- Unscented/Cubature Kalman Filter
- Optional: Particle Filter/SMC

### **4.1.2.4 Data Filtering and Outlier Removal Nodes**

The following data filtering and preprocessing methods will be evaluated for inclusion in nodes:



## D4.2: Advanced CDFF Architecture and ICD

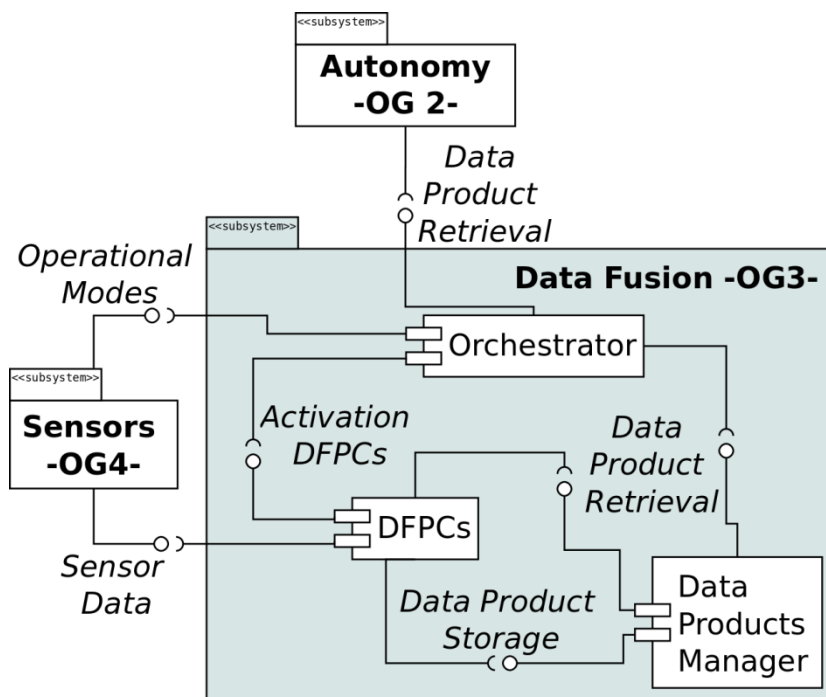
- FFT High/Low/Band pass filters
- Variance filter
- Decimation
- Normalization

The following outlier removal methods will be evaluated:

- Interquartile range / Mahalanobis distance
- One-class support vector machine
- Optional: Gaussian Mixture Models
- Optional: k-means
- Optional: Minimum Volume Ellipsoid

### 4.2 CDFF-Support

The CDFF-Support consists of a set of components that will run on the target system. These components provide supporting tools to use multiple DFNs together and in a coordinated fashion. Furthermore CDFF-Support also provides the Data Products Manager which stores a consistent representation of the environment, a history of acquired pre-processed sensor data, estimated poses, and a selection of the generated fused data products, so as related to deliver them under request to OG2.



**Figure 7:** The diagram presents how the three components of the CDFF-Support interact and also the interfacing with OG2 and OG4. The *orchestrator* (1) performs request to the Data Products Manager based on what OG2 requires (2) sets the operational modes for the sensors and (3) activates or deactivates DFPCs. The *Data Product Manager* (1) stores under request the data products generated in the DFPCs and (2) retrieves under request data products for the Orchestrator or the DFPCs.

### 4.2.1 Data Fusion Processing Compound

The main feature of a Data Fusion Processing Compound is the Description File and the associated DFNs. It states the DFNs involved in the DFPC, their connections and the processing order. This defines the steps to produce a data product from a set of another data product(s).

In more details, the description file contains:

- Description of the data fusion functionality - intermediate processing steps, input data to the DFPC and fused data outputs of the DFPC.
- List of required DFNs and their corresponding libraries
- Configuration, periodicity and priority of each DFN in the DFPC
- Connections description between the DFNs
- Frames (static and dynamic transformations) applicable to each DFN
- Optional
  - Semantics
  - Deployment recommendation
  - Nominal frequency of operation
  - Recommended queues lengths
  - Build instructions

In the Data Fusion Processing Compound Description File, information about possible ways in which the DFNs can be distributed among threads in order to maximize the efficiency of the CDFF depending on the target system is optionally included. The format of the description file will offer the possibility to specify on which thread each DFN should be executed depending on the number of available threads. A format to describe this information will be proposed, in order to facilitate the deployment task to either a user, to a middleware facilitator or target RCOS that could automatically use this information to partially or fully deploy a DFPC in a desired RCOS.

In order to develop a DFPC to be integrated in a robotic platform, the model of the robot is needed: (1) The names of the frames that are associated to the sources of the data (sensors) and to the target frame on which the fused data should be given must match the frames of the robot model. (2) The robot model is needed to set up the transformation graph from which the transformations can be obtained and (3) the robot geometric model might be required for some DFNs (Ex. odometry).

#### 4.2.1.1 Example of a data Fusion Processing Compound and the Description File

The following example was taken from an existing robotic software stack implemented in the RCOS Rock<sup>2</sup>. In the proposed example a lidar sensor is mounted on top of a tilting unit actuated via a dynamixel servo out, while moving up and down the lidar captures 2D scans of the environment. The DFPC performs the composition of the 2D scans taken with varying inclinations into a pointcloud. Furthermore a filter that removes parts of the robot from the 2D scans is included.

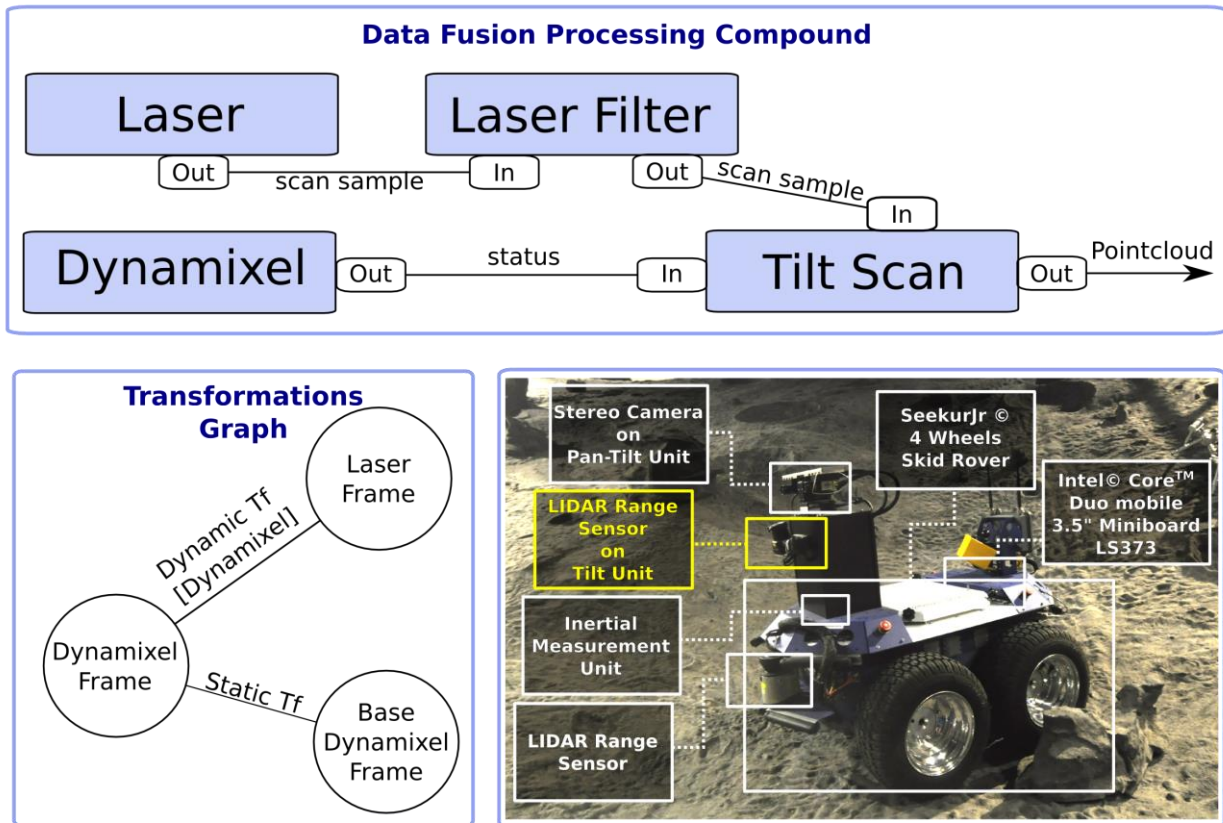
The corresponding DFPC describes a one layer lidar sensor (data source) connected to a lidar filter. The output of this last one is connected to a DFN node called Tilt Scan that produces the pointcloud.

---

<sup>2</sup> Rock RCOS website: <http://rock-robotics.org/stable/>

## D4.2: Advanced CDFF Architecture and ICD

To produce this pointcloud the status -transformations for the tilt- of the dynamixel is passed to the Tilt Scan DFN.



**Figure 8: This figure describes an example of a DFPC, two data sources (laser and dynamixel) connect with two DFNs. The data product that is generated is pointcloud and the input data types are 2D scan samples and servo status (including motor position).**

- Description of the functionality
  - Create pointclouds from of sensor data of a two dimensional lidar which is mounted on a servo and the status of the servo. The generated pointclouds can then be used for instance build DEMs which the autonomy framework will use for planning.
- List of required DFNs and libraries
  - Transformer, robot model and base-types
- Configuration, periodicity and priority of each DFN in the DFPC
  - Frame names:
    - Laser: Laser\_frame <from robot model>
    - Dynamixel: Dynamixel\_frame <from robot model>
    - Dynamixel Base: Dynamixel\_base\_frame <from robot model>
  - Periodicity:
    - Laser\_Filter: scan\_input\_triggered
    - Tilt\_Scan: scan\_input\_triggered
  - Priority:
    - Laser\_Filter: 1
    - Tilt\_Scan: 1
- Networks description
  - Inputs:

## D4.2: Advanced CDFF Architecture and ICD

- Laser: <2D scann\_sample\_type>
  - Dynamixel: <status\_sample\_type>
- Outputs:
  - Pointcloud
  - Laser.out.scan\_samples.connects\_to(Laser\_filter.in.scan\_samples)
  - Laser\_filter.out.scan\_samples.connects\_to(Tilt\_Scan.in.scan\_samples)
  - Dynamixel.out.status.connects\_to(Tilt\_Scan.in.joint\_status)
- Frames (static and dynamic transformations)
  - Static:
    - Dynamixel\_frame To Base\_Dynamixel\_Frame: <Path\_to\_robot\_model>
  - Dynamic:
    - Laser To Dynamixel\_frame: <Dynamixel.out.status>
- Optional
  - Semantics
  - Deployment recommendation
  - Nominal frequency, queues lengths
  - Build instructions

In this particular case the robot geometric model is needed to set the names of the source frame, the dynamixel frame and the target frame, and to set up the transformation graph with which the status of the dynamixel (dynamic transformations) will be feed.

### 4.2.2 Orchestrator

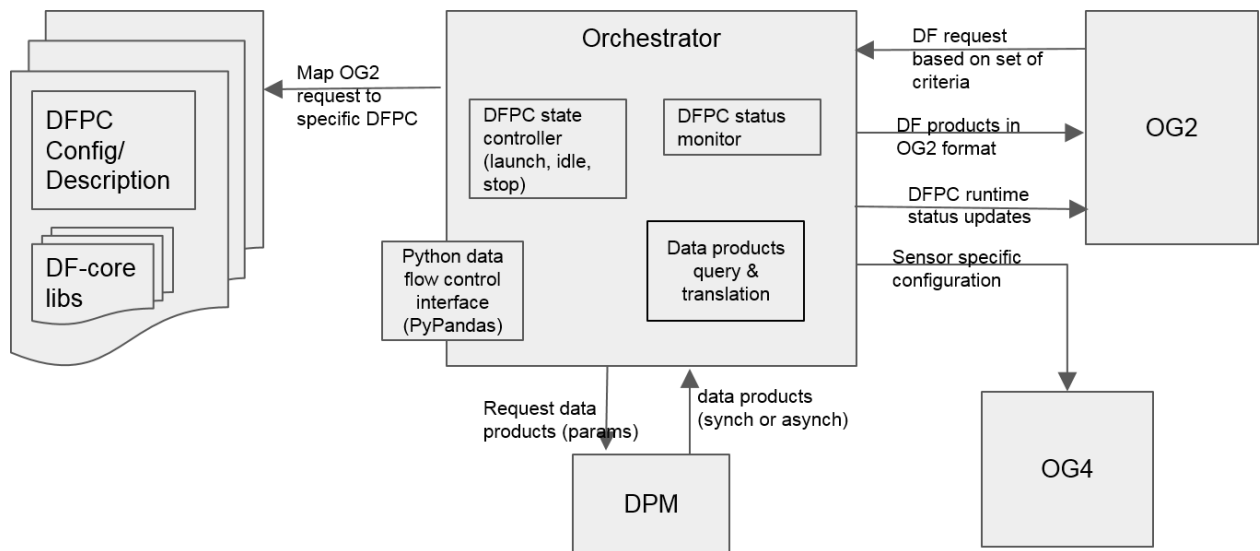
The orchestrator is an important functional component in the CDFF that has the main task of receiving queries from OG2 to activate certain DFPCs within OG3 and provide the fused data products to OG2 in the desired format. It acts as the central coordinator in the target system to control the activation states of DFPCs. The orchestrator has the following functions:

1. Interface between OG2-OG3
  - a. Accepts requests from OG2 with a set of criteria to select an appropriate DFPC
  - b. Provide a response about feasibility of the request
  - c. Provide the fused data to OG2 as a response synchronously (preferred option by OG2)
  - d. Notify OG2 that the fused data is available to be read i.e. asynchronously
2. Translate the perception and localization data into the format required by OG2 (or other DF product consumers) by accessing data products in the DPM.
3. Interface with OG4 Instrument Control Unit (ICU) to configure a limited set of sensor parameters (operational modes) influenced indirectly by OG2 requirement of data products with specific characteristics (resolution, range, data size, update rate etc.)
4. Interface with the Data Product Management (DPM) tool and provide mechanisms for querying fused data products
5. Activation and deactivation of DFPCs according to data product requests and operational modes of the sensors..

Optionally the orchestrator could provide additional functional capabilities:

1. Runtime monitoring of active DFPCs such as processing status, performance and memory constraints.
2. Functionality for selecting sub-components within a DFPC and initiating (or stopping) them for actual execution of the DFPC.
3. Provides a scripting (or user interface) mechanism for inserting or modifying DFPC descriptions that are stored in a repository

## D4.2: Advanced CDFF Architecture and ICD



**Figure 9: Internal software components of the Orchestrator and its interactions with other CDFF and external software modules**

The diagram in Figure 9 illustrates the internal software components of the orchestrator and interactions with the DPM, DFPCs, OG2 and OG4. The DFPC state controller ensures that a selected DFPC is activated by loading the appropriate run time libraries with the associated executable. This includes the task of mapping an OG2 request to the most appropriate DFPC that can satisfy the OG2 criterion.

### 4.2.3 Data Products Management Tool

#### 4.2.3.1 Motivations

Within the CDFF, a series of varied data and data products are processed and generated by a variety of DFNs. There is a need to store in a dedicated database a selection of these data structures, for different purposes such as:

- Serve OG2 requests for specific data products: for instance the Total Rover Map, which will certainly be requested at the end of a Long Traverse mission, requires the integration of (most of) the data gathered during the traverse, or of the various (Fused) Rover Maps that have been produced during the traverse.
- Serve the Operators, who may request specific data after the execution of a mission: for instance some data gathered in the vicinity of detected scientific target for further ground processing and analyses before commanding the rover to sample it.
- Serve internal CDFF purposes. Here various operations require the storage of data, either over the short or the long term, for instance:
  - To properly handle the asynchronicity of the data acquisition. Even if all the data are gathered in a synchronous manner, their processing require some time, and thus some data production may be delayed. For instance the precise position of the robot at the time of a 3D point cloud acquisition may be known only after a visual odometry process has refined the associated position.
  - By essence, SLAM solutions require the memorisation of environment related data (the maps of the M of SLAM, which are not necessarily products delivered to OG2). Depending on the selected SLAM implementation, such data can be landmarks, visual keyframes, point clouds keyframes, chunks of digital maps. etc. Most SLAM suites

---

## D4.2: Advanced CDFF Architecture and ICD

---

actually handle the management of such data internally, but there is an interest to expose them within a more principled spatial database management system.

- Prepare multi-robot cooperation schemes: similarly to the fact that environment related information are at the core of autonomous mobile robots, shared environment related information are at the core of the development of any multi-robot cooperation scheme. For instance a robot traversing an area already traversed by an other one may benefit from the information already gathered, be it for planning or localisation purposes: one robot's OG2 or OG3 sub-systems may require data stored by an other robots OG3 sub-system.

It is the role of the Data Product Management to memorise and manage the acquired data, the processed data and the generated data products that may be required for all these needs.

### 4.2.3.2 DPM implementation

Of course by no means all the data that is processed within the CDFF data can be stored: the DPM must manage the data stored so far, and be able to expose relevant data products to the various client's processes that require them.

The DPM can be seen a robotics-dedicated Geographic Information System (GIS). With respect to the activated DFNs and DFPCs in the CDFF, the DPM will process the data insertion requests. Internally, it manages all the spatial related data by implementing insertion, deletion or update functions, aiming at satisfying future needs for data products and storage constraints. With respect to the other systems and subsystems, the DPM acts as a server of data products for client processes, whatever they are (i.e. internal OG3-InFuse processes or OG2-Ergo processes).

Internally the DPM will manage an EnviRe Graph in which multiple data products will be stored in a spatiotemporal consistent manner. Data Products might be serialized in this module to free memory usage. Given a request, the serialized data products shall be deserialized and deliver or the reference to a serialized object might be delivered instead. EnviRe currently provides serialization for various data types related to environment representation (e.g. maps) and it is extendable to enable serialization of further data types.

## 4.3 CDFF-Dev

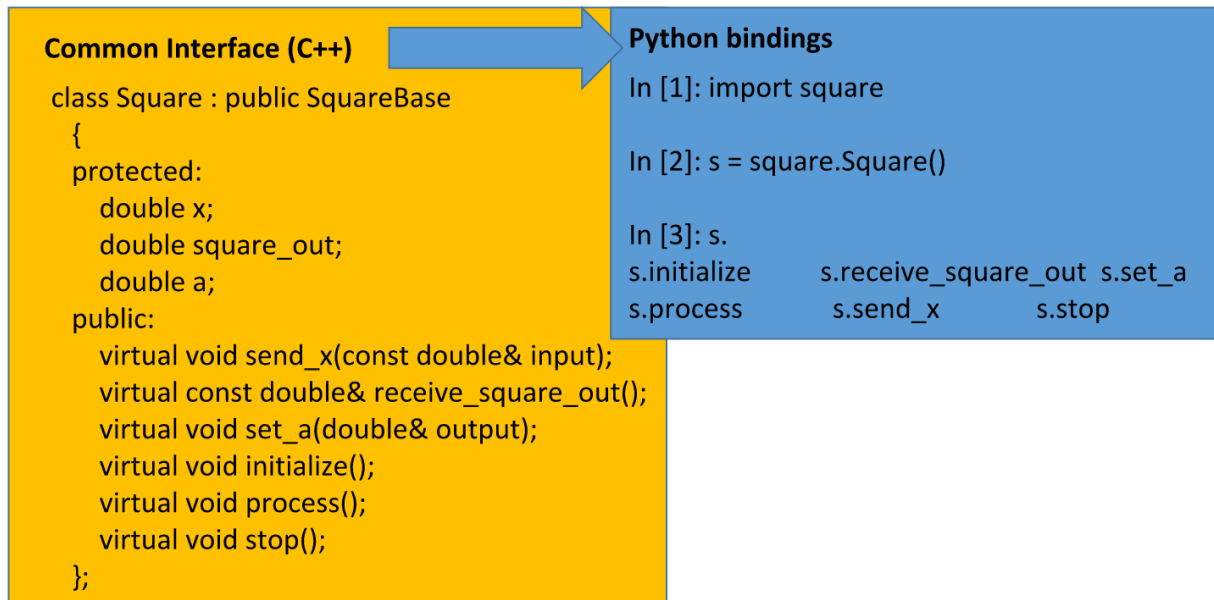
Through the CDFF-Dev utilities, the developer will be able to: (1) connect DFNs and evaluate them with logged data, (2) define DFPCs and evaluate them with logged data, (3) test the orchestrator with logged data, (4) test the DPM tool with logged data, (5) run performance analysis to evaluate at least data filtering and outlier removal nodes and (6) visualize data communicated through DFNs using the Envire visualizer. Additionally, CDFF-Dev provides a Middleware Facilitator which eases the transition from the developer environment to the target system.

### 4.3.1 CDFF-Dev Common Interface

The DFNCI will provide Python bindings to allow prototyping of DFPCs or newly integrated DFNs. The bindings will be automatically generated, either based on the Node definition file or directly based on the C++ header of the DFN that implements the DFNCI. A tool to automatically generate wrappers for DFNs is provided in CDFF-Dev if the DFN only uses InFuse data types at its interface. The bindings will be based on Cython, which makes it easy to develop Python extensions that directly wrap existing C++ libraries. An example of a generated wrapper is shown in the next figure.



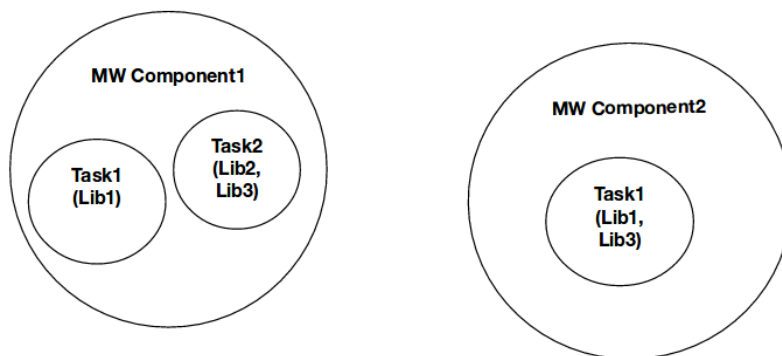
## D4.2: Advanced CDFF Architecture and ICD



**Figure 10: An example of Python bindings generated for a DFN. A node that implements the DFNCI is displayed on the left side. On the right side, the auto completion from the IPython REPL is used to display the methods in Python.**

### 4.3.2 Middleware Facilitator

The Middleware Facilitator component eases the process of transferring the data fusion solution from the developer's environment to the target system - RCOS and robot specific-.



**Figure 11: Terminology: MW Component - Library(s) + mw wrapper (system level process); Task - Threads running within the MW Component; DF-Lib - one or more DF algorithm implementations with appropriate interfaces**

Figure 11 introduces some terminology that will ease the understanding of the following description. The middleware facilitator includes a code generator that will:

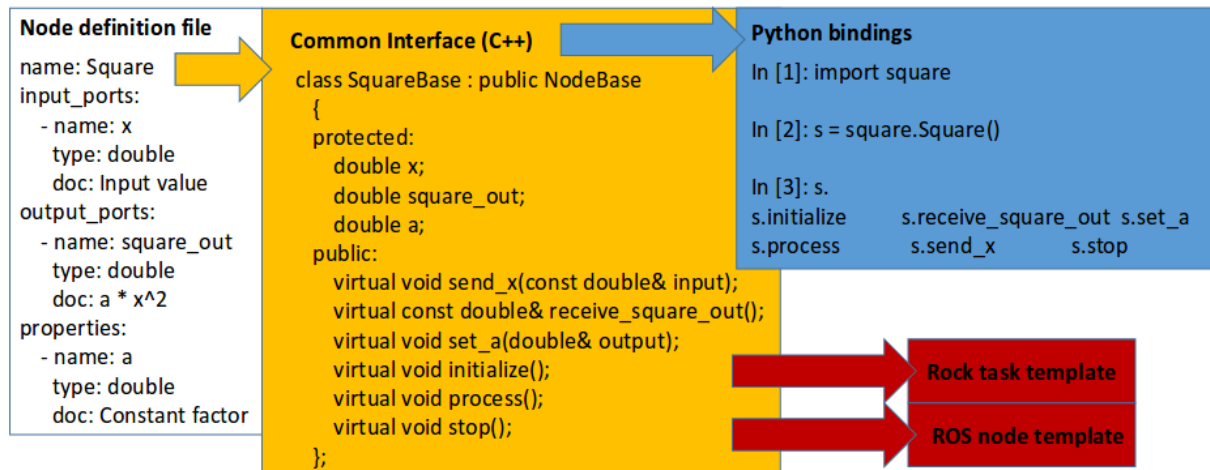
- Generate templates for the DFNCI
- Generate Python bindings for the DFNCI
- Generate, a template (e.g. for Rock tasks) wrapping the DFNCI, including
  - Port names and types
  - Properties and types

## D4.2: Advanced CDFF Architecture and ICD

- Type conversions

It will not contain temporal alignment of input ports, component frequencies, a description of the state machine or complex inheritance structures, and it will not generate the build configuration files because this is considered too complicated for a framework-agnostic code generation tool.

The Figure 12 gives an example for a very simple case. The developer writes a node definition file which defines inputs and outputs of the node, and configuration options. A base class that implements the DFNCI is generated automatically. The developer has to implement the logic that uses a RCOS-independent library to implement the DFNCI. From the DFNCI, Python bindings can be generated. In addition, templates for several RCOS like Rock or ROS could be generated. For the case of the RCOS TASTE the generation of such template might not be possible, because the classical design of modules involves the use of Graphical User Interfaces. The developer has to configure these templates, e.g. in order to align the inputs temporally, define the component frequencies, or configure the build process of the component. Libraries for type conversions between RCOS types and InFuse types must be available to facilitate the generation and development of the RCOS component.

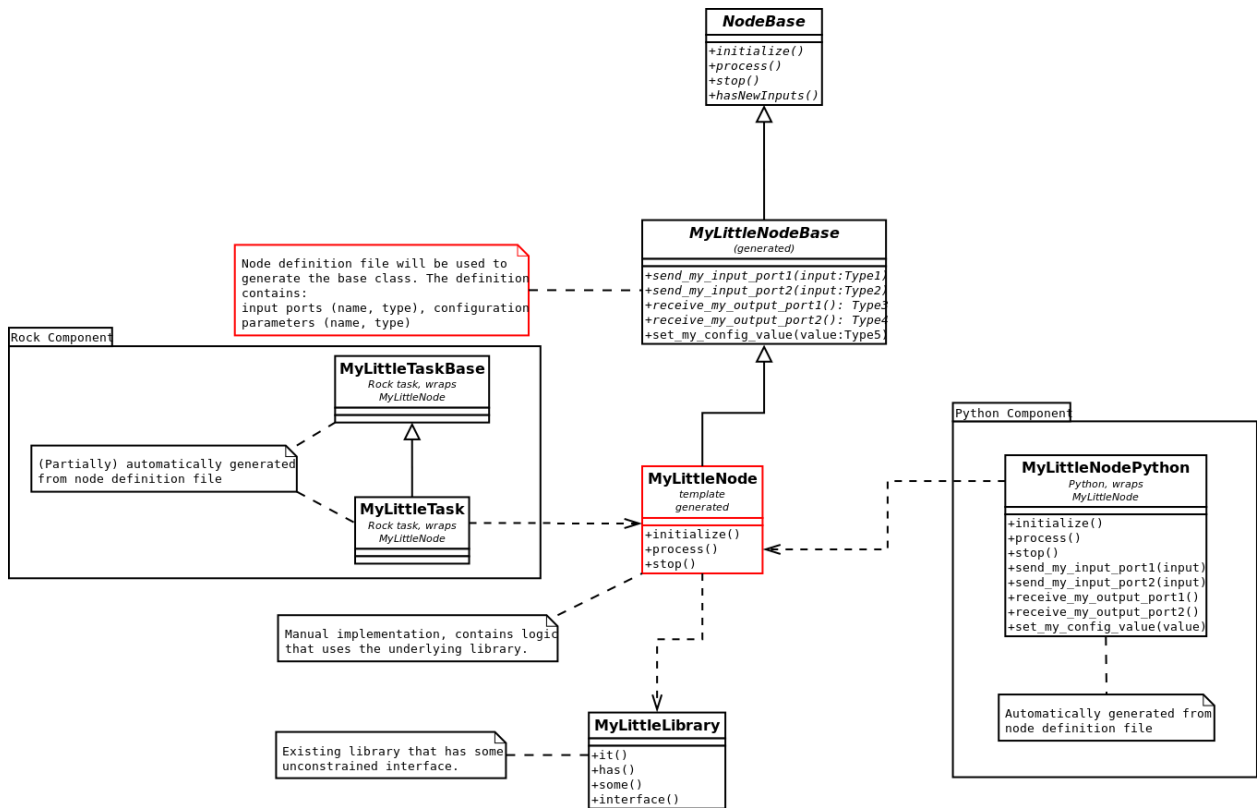


**Figure 12: The DFNCI template for a DFN will be generated from the node definition file. In addition, we can generate templates for RCOS components and a Python wrapper.**

The Figure 13 presents a class diagram that shows the relations of the generated and manually implemented modules.



## D4.2: Advanced CDFF Architecture and ICD

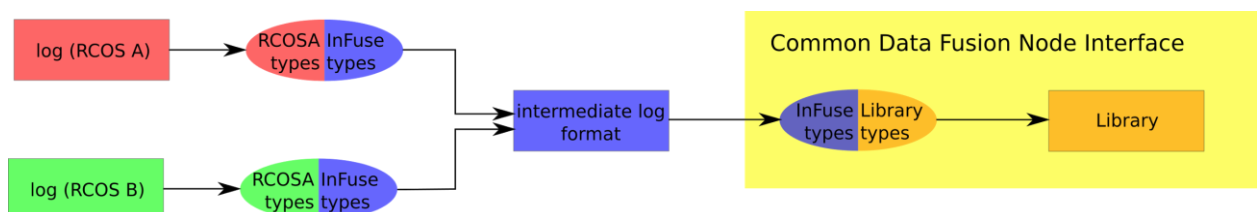


**Figure 13: Class diagram of a node that implements the DFNCI. A Python binding and a template for a Rock component could be generated automatically. Red color indicates files that will be implemented by the developer**

### 4.3.3 Logs and Data Flow Management

The Logs and Data Flow Management (LDFM) module is a set of utilities that facilitates the Developer of DFPCs the connections between logs and core libraries (with the Data Fusion Node Common Interface) inputs and outputs. The LDFM belongs to the CDFF-Dev set of features and it is not deployed in the target system. The LDFM is not conceived as substitute of an RCOS but rather as an utility to evaluate - in the developer's environment - the adequacy of a DFN or a DFPC with logged data.

The Log Player is a tool that allows to replay a data fusion process in a chronological fashion following the timestamps of the data samples processed. The player should allow to play log data produced by different RCOS. To achieve this, specific converters for each framework to the intermediate log format have to be implemented. InFuse types must be serializable in the intermediate log format.



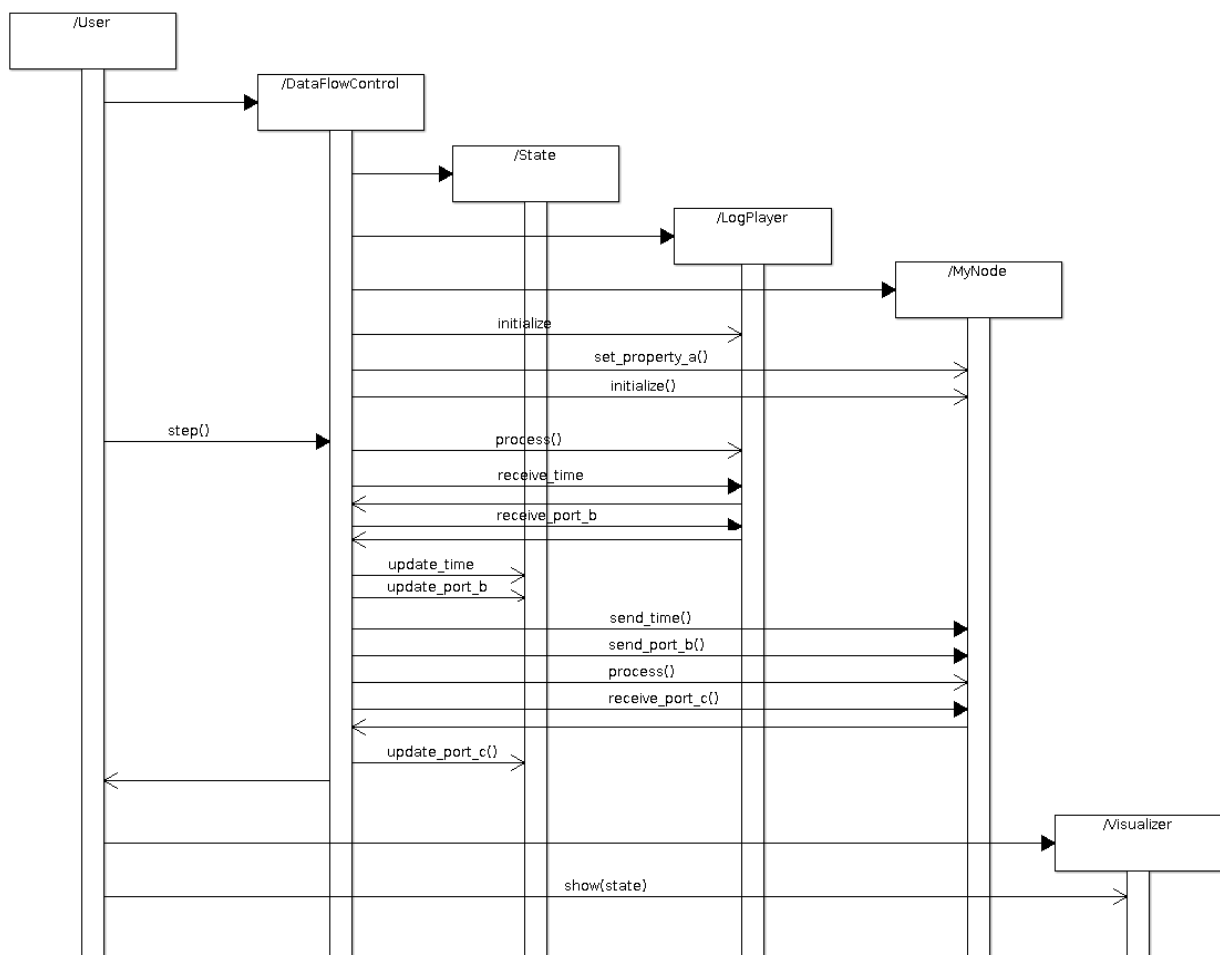
## D4.2: Advanced CDFF Architecture and ICD

**Figure 14: Conversions from RCOS-specific log formats to an intermediate log format that is independent of the RCOS have to be implemented. On the side of the DFNCI, conversions to the data types of the library have to be implemented.**

The following subcomponents will be implemented:

- Data Flow Control: emulates the communication layer of an RCOS
- Log Player: allows to replay data fusion processes in a chronological fashion

The implementation of the data flow control is in a single process with one thread, everything is sequential. It will be lightweight; it should not be an RCOS replacement. It will have a simple API that can be directly controlled, e.g. from a Python REPL. An example of a sequence diagram in a simple scenario is shown in Figure 15.



**Figure 15: Sequence diagram that shows how a user could interact via REPL with the log player and the data flow control.**

### 4.3.3.1 Python Bindings for EnviRe

Envire is a package for representing arbitrary information on the environment for robotics. The purpose is to have a common way of holding any information related to the environment of a robot and how the information relates to each other in a consistent representation. The applications of EnviRe varies

## D4.2: Advanced CDFF Architecture and ICD

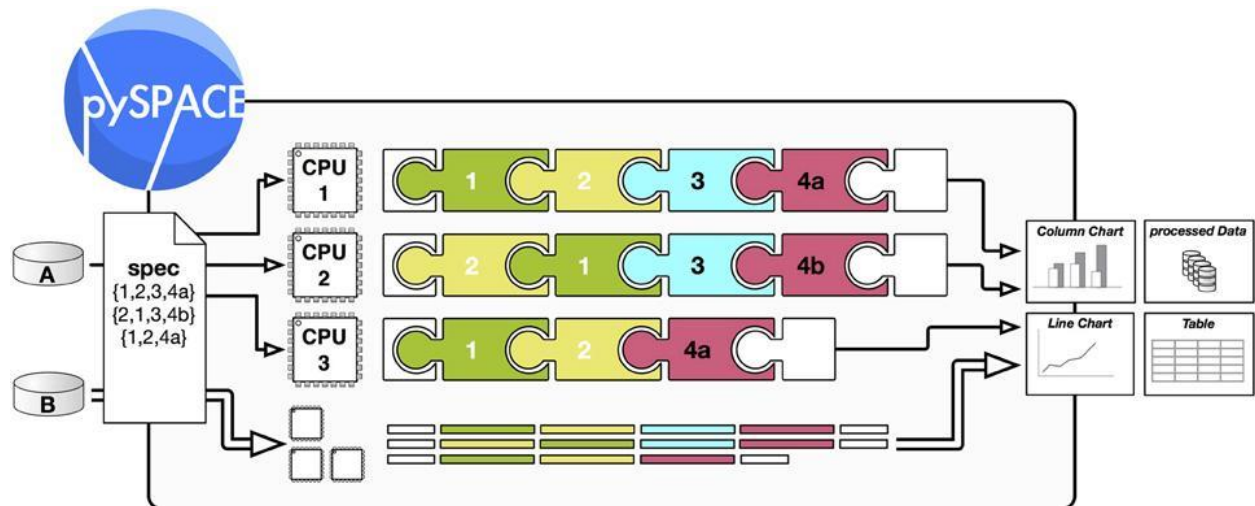
between environment representation, navigation, planning and simulation. The model is a strongly connected directed graph which allows data acquisition, processing, and operations regarding autonomous systems demands.

In the context of InFuse, the EnviRe graph will be used as symbolic representation and structuring of the spatio-temporal information, as well as to facilitate the attribution of metadata. The visualization of EnviRe will be used to analyze data that is stored in the graph.

In order to use EnviRe as a tool in CDFF-Dev, Python bindings will be implemented as the graph and the visualization must become available in Python. The tool Cython will be used to wrap the C++ libraries directly. Data types that should be stored in EnviRe graphs must also be wrapped in Python. Because Python has a dynamic type system and C++ has a static type system, the exact template specialization for EnviRe have to be known during compilation of the Python extension, hence, it is not possible to write generic Python bindings for EnviRe that accept arbitrary Python types.

### 4.3.4 Data Analysis and Performance Tools

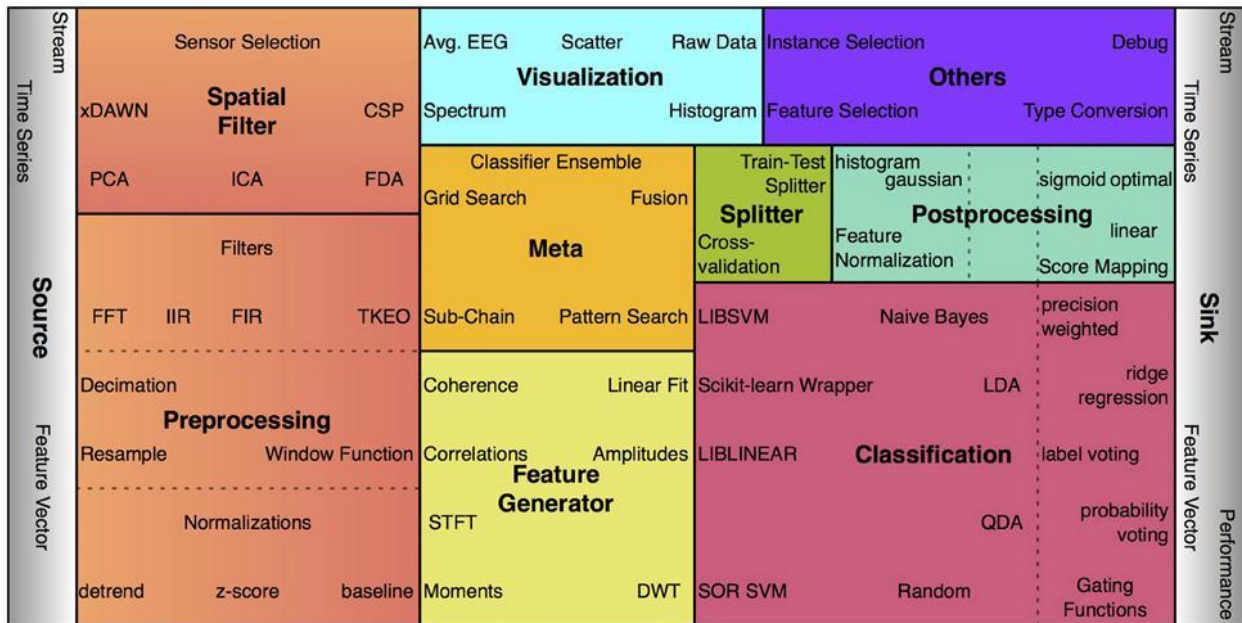
The Python Signal Processing And Classification Environment (pySPACE) will be used to analyze and compare different data fusion processing compounds (DFPCs). In the context of InFuse the main aspect of pySPACE is to be used for benchmarking purposes and for the integration of general filtering and outlier removal for sensor data.



**Figure 16: Visualization of pySPACE node chains. pySPACE can distribute the execution of independent node chains that will be compared over different processing units, e.g. CPUs.**

PySPACE is a powerful tool that enables parallel process execution and offers more than 100 machine learning and signal processing algorithms that are implemented exclusively for pySPACE and wrappers for libraries like scikit-learn, Weka, and libsvm. It is easy to prototype and evaluate node chains that are not yet implemented for an RCOS.

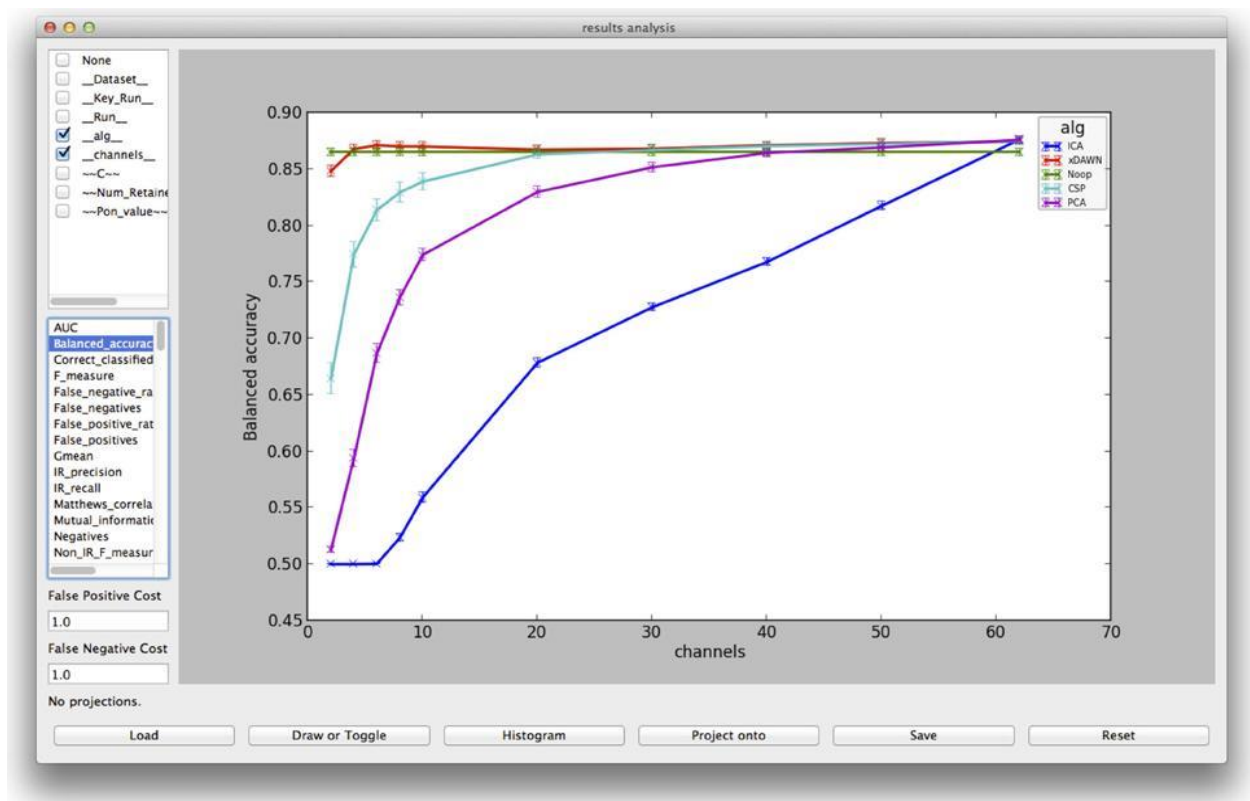
## D4.2: Advanced CDFF Architecture and ICD



**Figure 17: pySPACE provides a large number of nodes for preprocessing, feature generation, classification, visualization, etc.**

PySPACE can be used to compare and analyse the performance (e.g., classification accuracy...) of several algorithms and parameter configurations easily in a graphical user interface. PySPACE also provides a 'launch\_live' mode where you can directly execute signal processing as soon as you have the data in an online fashion. A multitude of metrics is already available for machine learning methods.

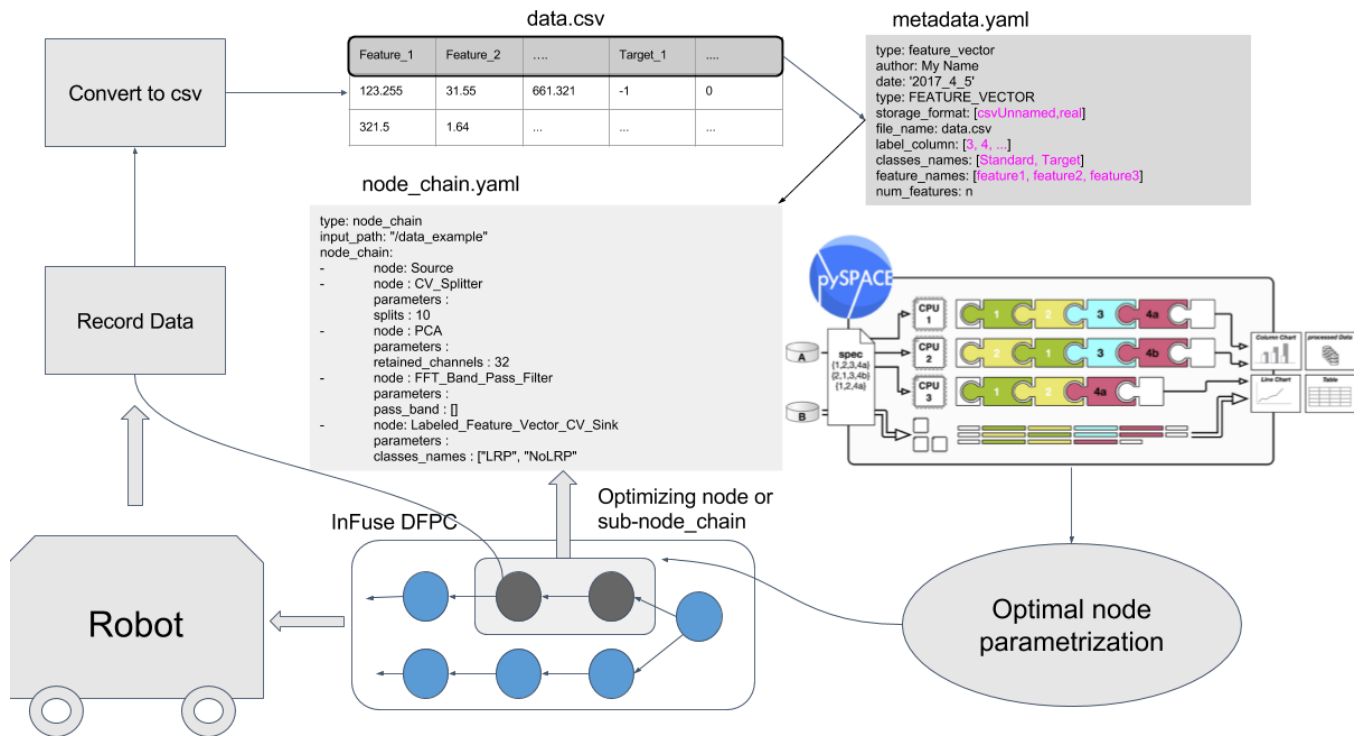
## D4.2: Advanced CDFF Architecture and ICD



**Figure 18: pySPACE offers various metrics and visualization tools to compare different machine learning and signal processing methods.**

The general workflow for optimizing a DFN can be summarized as the following: Provided the data source to a specific node (which can be user specified or supplied by the previous node in the chain), a metadata.yaml file is setup which maps the header of the data being processed into a yaml file that is typically used in pySPACE. In the metadata file the user can choose between selecting all or part of the data features to be processed. Next a configuration file is setup, where the user can chain operation nodes selecting different algorithms as well as different parameter settings to be compared. It is also possible to compare the performance of full processing pipelines. After setting up the configuration file, the processing is then launched and the evaluation results are then synced together into a csv file. The results can be easily visualized using the performance analysis tool (as shown in the figure above). The parameter of the best performing node can then be exported to the corresponding node of the DFPC.

## D4.2: Advanced CDFF Architecture and ICD



**Figure 19: pySPACE will be used as a tool to evaluate signal processing and machine learning nodes that will be deployed on the robot.**

The workflow of developing machine learning and signal processing methods for DFPCs will include an initial evaluation of several methods that can be done in Python with pySPACE. Selected algorithms will be implemented in C++.

The implementation in C++ will have a constrained interface -DFNCI- which will be implemented by the developer of the node. For this interface, a Python wrapper can be generated automatically and a generic node for pySPACE is able to wrap these nodes. The pySPACE node needs a serialization method, i.e. it automatically determines parameters of the model and stores them in a string. There is a distinction between transformation nodes (signal processing) and trainable nodes (machine learning). Method calls from the pySPACE node will be forwarded to C++.

**Example:** in practice, an algorithm can have an unconstrained implementation that is provided by some library (MyAlgorithmLibrary). The pySPACE interface (MyAlgorithmPySpace, C++) has to be implemented. For a transformation node, only the method `execute()`, which does the transformation, has to be implemented. For a machine learning node, the methods `isTrainable()`, `isSupervised()`, `train(...)`, `stopTraining()`, and `incTrain(...)` must be implemented. Serialization and deserialization methods (e.g. `fromYaml(...)`, `toYaml()`) are required. A tool to automatically generate the Python bindings (MyAlgorithmPySPACE, Python) will be available. For transformation nodes, the `CppTransformationNode` will be available in pySPACE, for machine learning nodes, the `CppTrainableNode` will be available.

## D4.2: Advanced CDFF Architecture and ICD

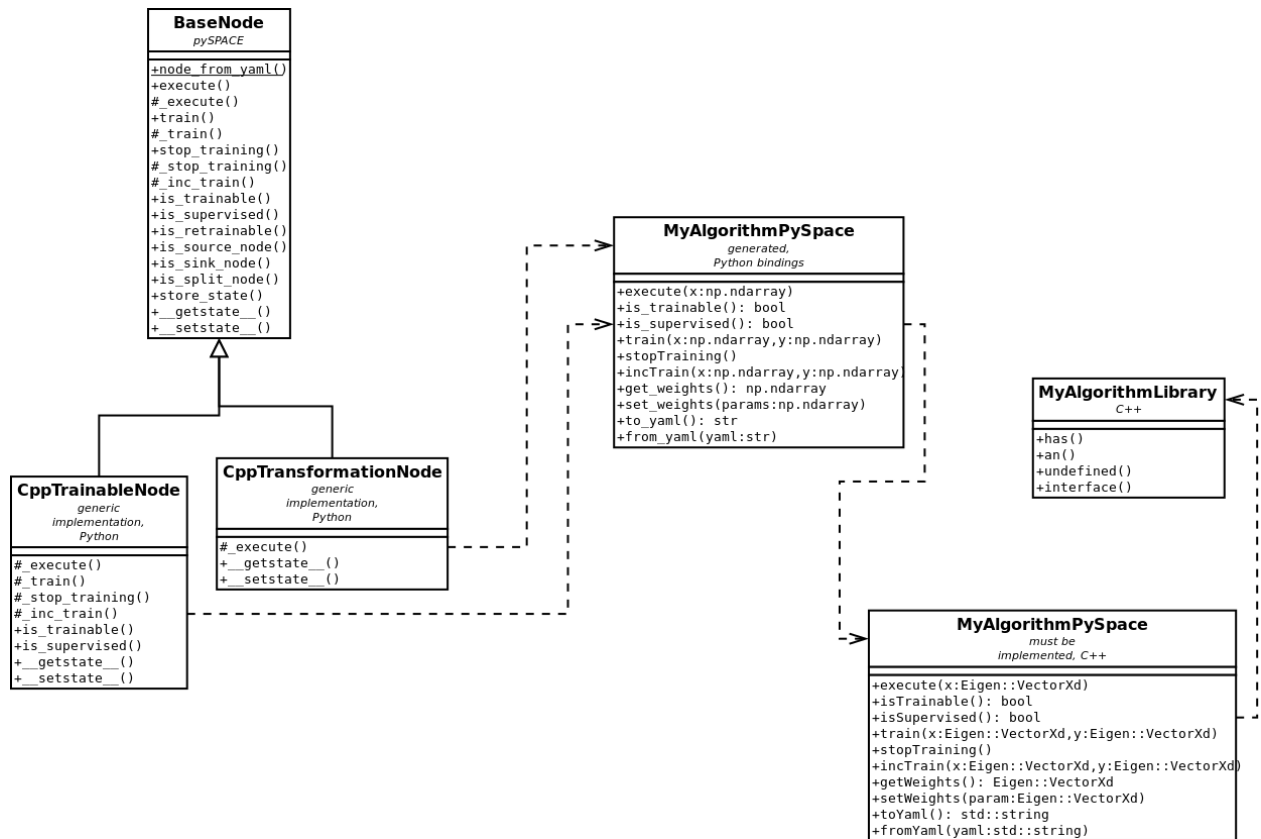


Figure 20: Class diagram that shows how a hypothetical algorithm *MyAlgorithm* can be integrated in *pySPACE* as a *pySPACE* node.

### 4.3.5 Visualization Tools

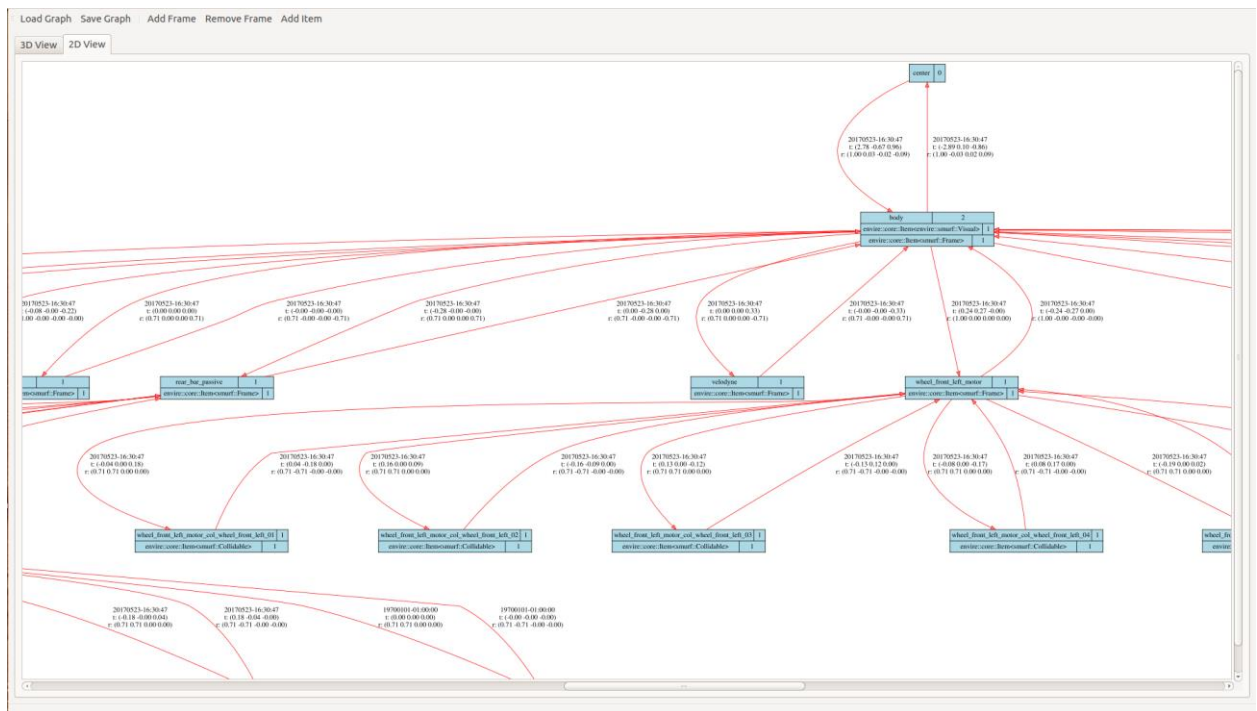
The *EnvRe* visualization tool will be used in the developer's environment to visualize the data products generated at different stages of the DFPCs. In order to visualize the data product Python bindings to the data type will be required as well as a visualization plugin for the type. Currently *EnvRe* Visualizer support *Vizkit* visualization plugins. There exist already *Vizkit* plugins for the base-types library<sup>3</sup> upon which the OG1 types are based, for map types compatible with those<sup>4</sup> and for other types commonly used in robotic applications.

<sup>3</sup> Base-types: <https://github.com/rock-core/base-types>

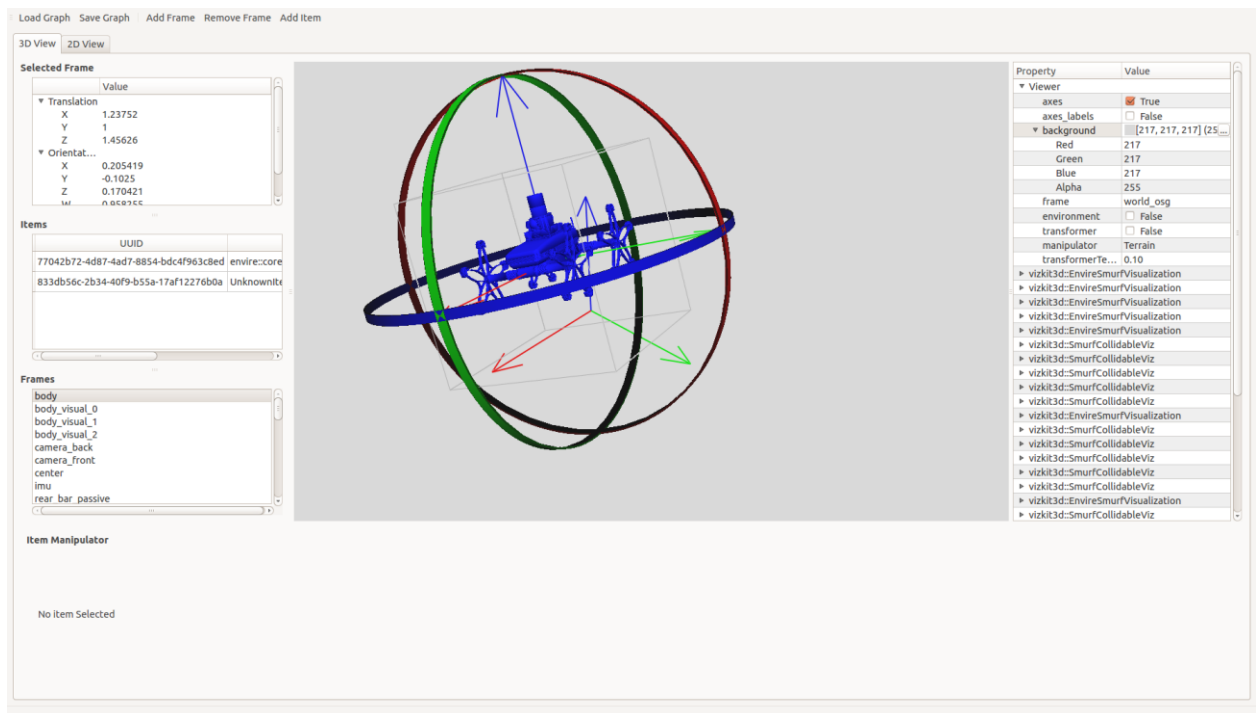
<sup>4</sup> Slam maps: <https://github.com/envire/slam-maps>



## D4.2: Advanced CDFF Architecture and ICD



**Figure 21: The 2D representation of the Envire graph can be visualized using Envire Visualization. The 2D view presents all the transformation as well as the objects stored in each frame with their correspondent type name. In this image a robot model is visualized.**



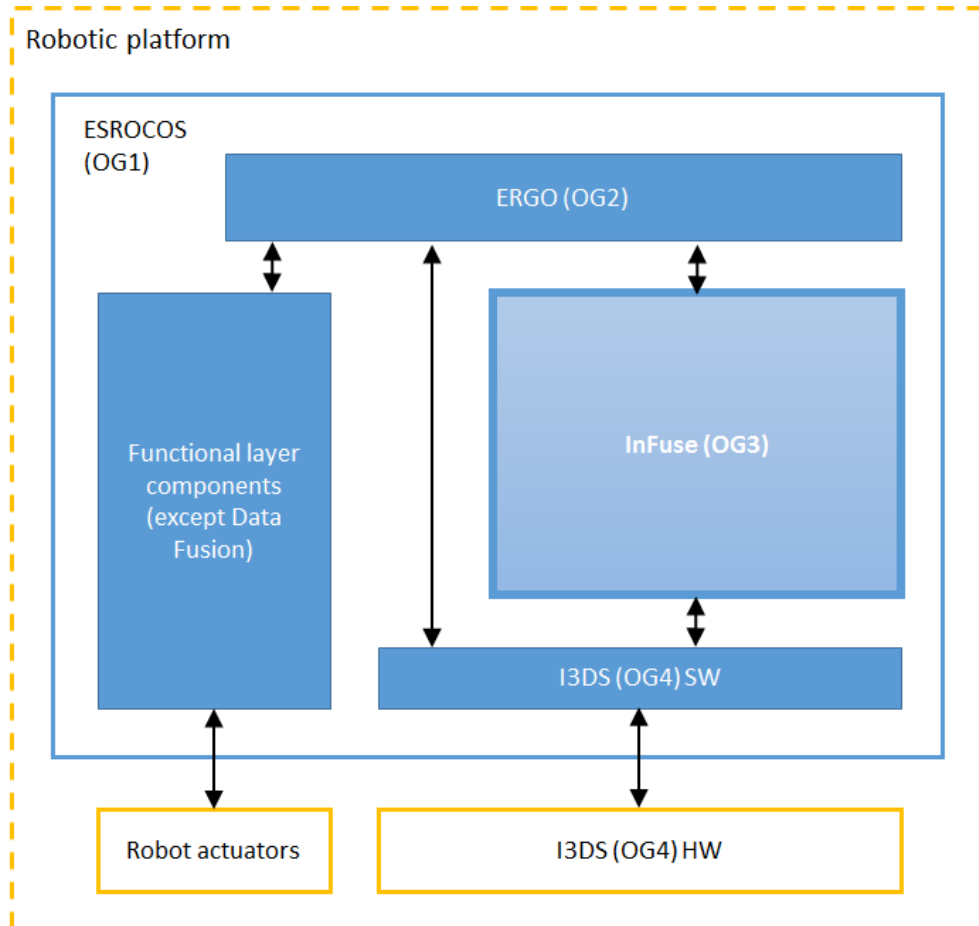


**Figure 22: The 3D representation of the Envire graph can be visualized using Envire Visualization. The different transformations, frames as well as the objects contained them can be inspected. In this visualization the visual components of a robot model are shown.**

## 5 Interface Control Document

### 5.1 OG3 Product Definition

OG3 offers the Data Fusion Module for processing sensor data produce and generate from it the pose information and maps, required by the planning module (OG2). In Figure 23, the modules of autonomy (OG2), perception (OG3) and sensor (OG4) are presented as well as the RCOS (OG1).



**Figure 23:** The figure above allows to identify the potential set of interactions between OG3 on the one hand and OG2 and OG4 on the other hand. OG4 is understood as including a hardware part – i.e. the physical sensors, and a software part consisting of the drivers and pre-processing capabilities included in I3DS.

#### 5.1.1 Identification of OG3 Interactions with other OGs

OG3 is controlled by OG2 in two ways: (1) OG2 is responsible for the initialization of OG3 and (2) OG2 through request of data products triggers the activation in OG3 of the most suitable Data Fusion Processing Compounds to satisfy the request. OG2 is also responsible for the activation of OG4 (sensor).

Although OG2 is responsible for the decisions in the overall architecture, OG3 can trigger the activation of operational modes of OG4. These operational modes determine the preprocessing steps, the sensor data generated and the interfaces through which is sent. Each pre-processed sensor data delivered by

## D4.2: Advanced CDFF Architecture and ICD

OG4 is associated to its correspondent interface. Each interface is associated to only one specific type of sensor data and pre-processing steps.

The data products generated by OG3 will also be communicated through specific interfaces for each data type associated to the request received. In Figure 24 the consensual approach for the interactions of OG3 with other OGs is summarized.

The requests for sensor data will always go through OG3. In order to increase efficiency, only the fused data will come from OG3 and any direct output from OG4 requested by OG2 will come directly from OG4 to OG2.

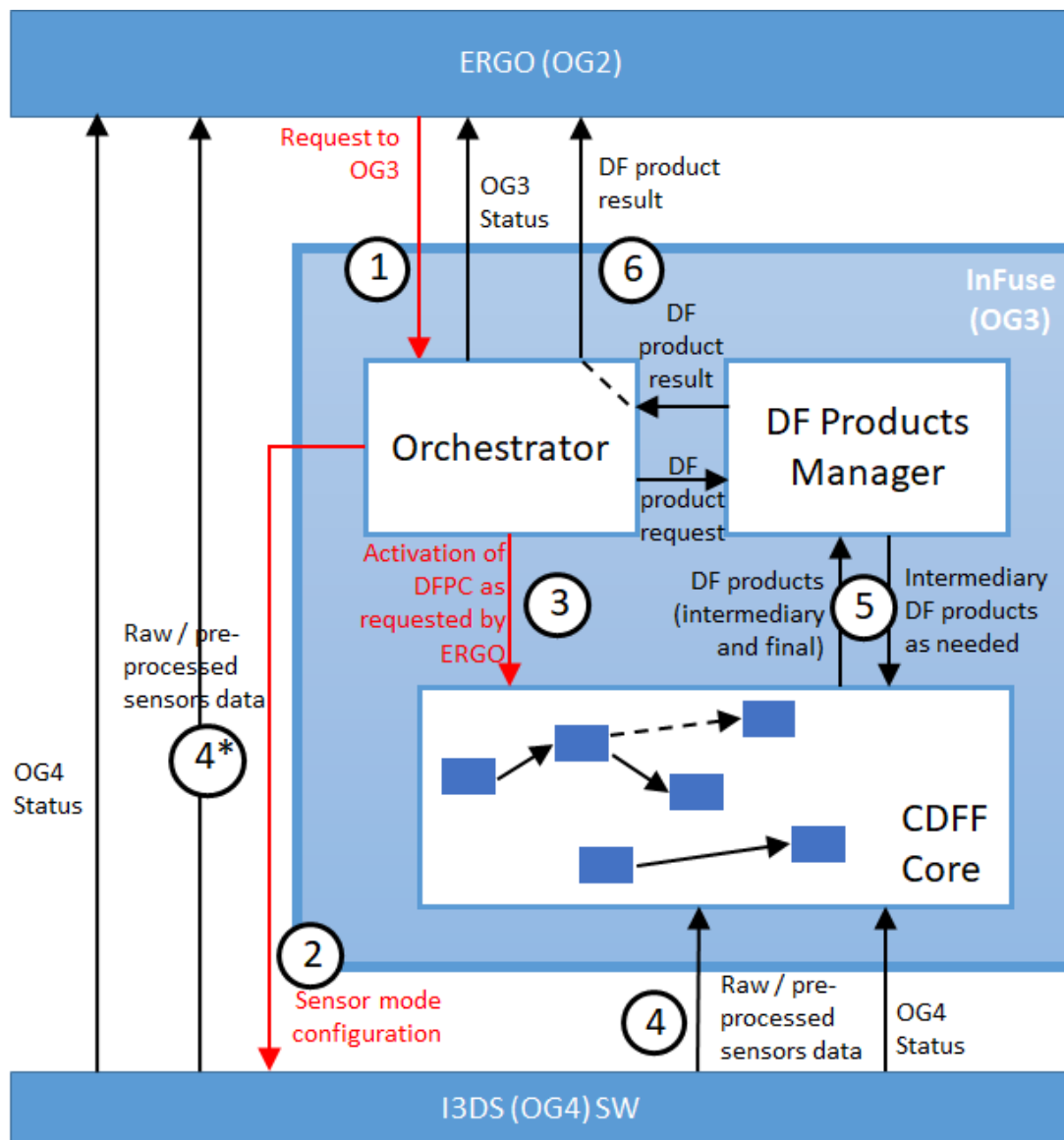


Figure 24: Summary of the interactions of OG3 with other OGs

### 5.1.2 Data Products Generated by OG3

The data products generated by OG3 are specified by the needs expressed by OG2-ERGO in ERGO\_D1.2 “System Requirements”. They are split in two families: information related to the environment (terrain or target), and information related to the position of the controlled system (rover or servicing satellite) with respect to the environment.

Note that besides the data products exported to OG2-ERGO, OG3 generates data products for its own needs: they are not listed in this document.

### 5.1.3 Planetary reference implementation

The main information related to the environment exploited by OG2-ERGO are Digital Elevation Maps (DEM), a 2.5D representation of the terrain defined over a regular Cartesian grid, which encodes terrain heights. These maps are possibly completed with an Uncertainty Map and a Soil Type Map, that share the same structure, but encode respectively the uncertainty on the height and the soil type.

Depending on the data that has been fused to produce these maps, their resolution, extent and reference frame differ. ERGO\_D1\_2 has defined three different maps:

- The *RoverMap* only integrates data perceived from a single rover position, and is expressed in the rover reference frame (this map is used to detect and avoid obstacles while driving)
- The *FusedRoverMap* integrates data perceived from various recent positions of the rover, and is expressed in a local terrain reference frame (this map is used to plan local rover trajectories)
- The *FusedTotalMap* integrates all the data gathered by the rover and other available information, and is expressed in a global absolute frame (this map is used to plan long term itineraries)

Note the associated data structures are defined by the following parameters: map extent (rectangular area, in meters), map resolution (size of the DEM cell, in meters), and map content (height, uncertainty, soil type).

Rover localization information is required by OG2 for the following purposes:

- At a rather high frequency (10Hz), to control the execution of the short term planned paths
- To plan local paths, so as to generate trajectories that follow the long term planned path
- To plan long term path using in particular the orbiter map

What distinguishes pieces of localization information (besides the inner OG3 processes that produce them) is the reference frame in which they are expressed. OG3 proposes the following reference frames:

- *AbsoluteFrame*: is the “geo-reference” frame attached to the planet
- *GlobalTerrainFrame*: is a reference frame in which is defined a whole navigation mission (a long term path). The Cartesian Coordinates of the frame should be  $x/y/z = \text{East/Nort/Up}$ .
- *LocalTerrainFrame*: is a reference frame in which are defined the local trajectories. For this reference frame the  $z$  component corresponds to upward.

---

## D4.2: Advanced CDFF Architecture and ICD

---

- RoverFrame is the frame associated to the rover body. The localization information pertains to this frame. Its orientation is usually "x-forward, y-leftward, z-upward".

Given the fact that the the various localization information are produced by different processes, at different frequencies (and sometimes upon request) and express an information in different reference frames, we define three different interfaces:

- LocalPose: Pose of the BodyFrame in the LocalTerrainFrame
- GlobalPose: Pose of the BodyFrame in the GlobalTerrainFrame
- AbsolutePose: Pose of the BodyFrame in the AbsoluteFrame

Each of these pose consists of 3-axes attitude and 3-axes position information, with associated uncertainties (format to be defined: uncertainties on the 6 pose parameters, covariance matrix ...).

### 5.1.4 Orbital reference implementation

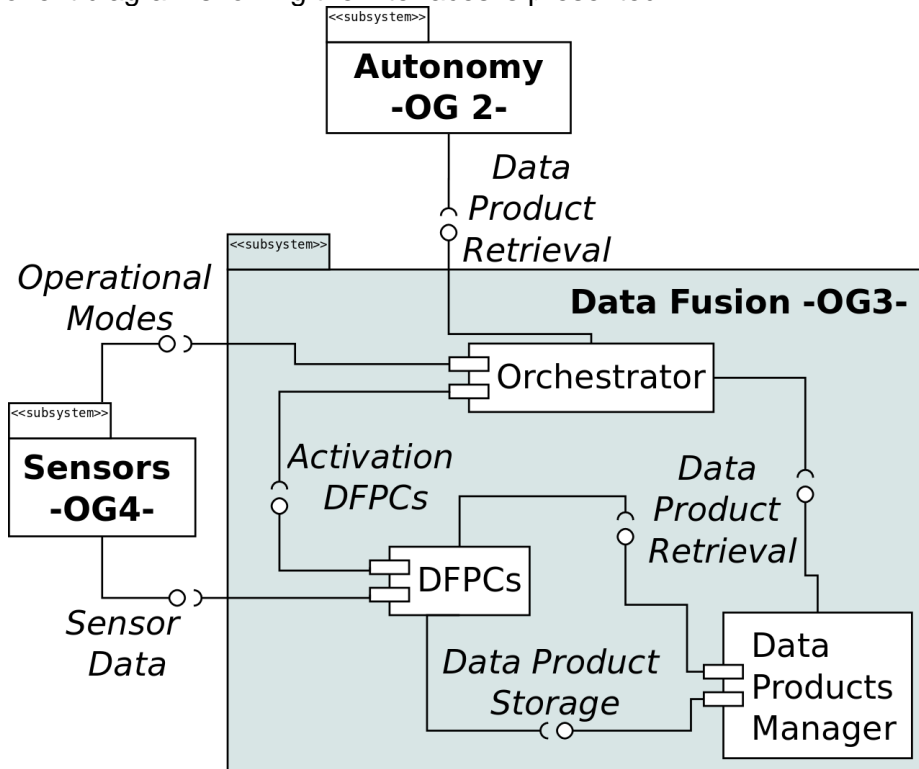
The sole information related to the environment in the orbital scenario are the following:

1. The partial 3D model of the target spacecraft, built from data gathered by the servicing chaser spacecraft
2. The localization information is the relative position of the target Body Frame with respect to the Servicing Body Frame, associated to relative speeds.

## D4.2: Advanced CDFF Architecture and ICD

### 5.2 Internal Interfaces

In this section the list of the methods identified for the internal interfaces is provided. In Figure 25 a component diagram showing the interfaces is presented.



**Figure 25: The internal interfaces are (1) Data Product Retrieval and (2) Data Product Storage implemented by the Data Product Manager and (3) Activation DFPCs implemented by the DFPCs**

#### 5.2.1 Data Product Storage

This interface is implemented by the DPM to receive requests for the storage of data products. Details for these requests are still to be defined internally.

Interface	Method	Description
DPMI	storeDEM	A DEM data product is passed as parameter which should be stored by the DPM
DPMI	storePose	A Pose data product is passed as parameter which should be stored by the DPM

**Table 2: Interface Data Product Storage implemented by the Data Products Manager**

#### 5.2.2 Data Product Retrieval

This interface is implemented by the DPM to answer requests for the retrieval of data products. Details for these requests are still to be defined internally.

## D4.2: Advanced CDFF Architecture and ICD

Interface	Method	Description
DPMI	getDEM	A DEM data product or the serialized reference to it is returned.
DPMI	getPose	A Pose is returned by the DPM.

**Table 3: Interface Data Product Retrieval implemented by the Data Products Manager**

### 5.2.3 Activation DFPCs

Table 4 presents the methods of the DFPC interface. In this case, to clarify the relationship with the Data Fusion Nodes the Data Fusion Node Common Interface related methods are also included.

Interface	Method	Description
DFNCI	initialize	Activates a DFN. It can be called multiple times, e.g. after stop has been called or before any other method has been called.
DFNCI	stop	Deactivates a DFN. Can only be called after initialize has been called.
DFPC	initialize	Calls 'initialize' for each DFN of the DFPC.
DFPC	stop	Calls 'stop' for each DFN of the DFPC.

**Table 4: Interface of the DFPCs for the Orchestrator**

### 5.3 Interface with OG1

The developer designs a deployment network. If necessary, type conversions between internal types and externally communicated types must be implemented by the developer. Transmitted data types can be custom designed by the developer (in agreement with other relevant OGs) or are predefined (as defined in OG1). The developer integrates a component network in the interface view. OG1 will generate code scaffolds for each OG1 component that will be implemented by the developer. The source code of the whole component or a static library must be provided by the developer. In the deployment view, the developer will specify how the components will be deployed on the target system. In Figure 25 the workflow to integrate the products from OG3 in OG1 is presented.

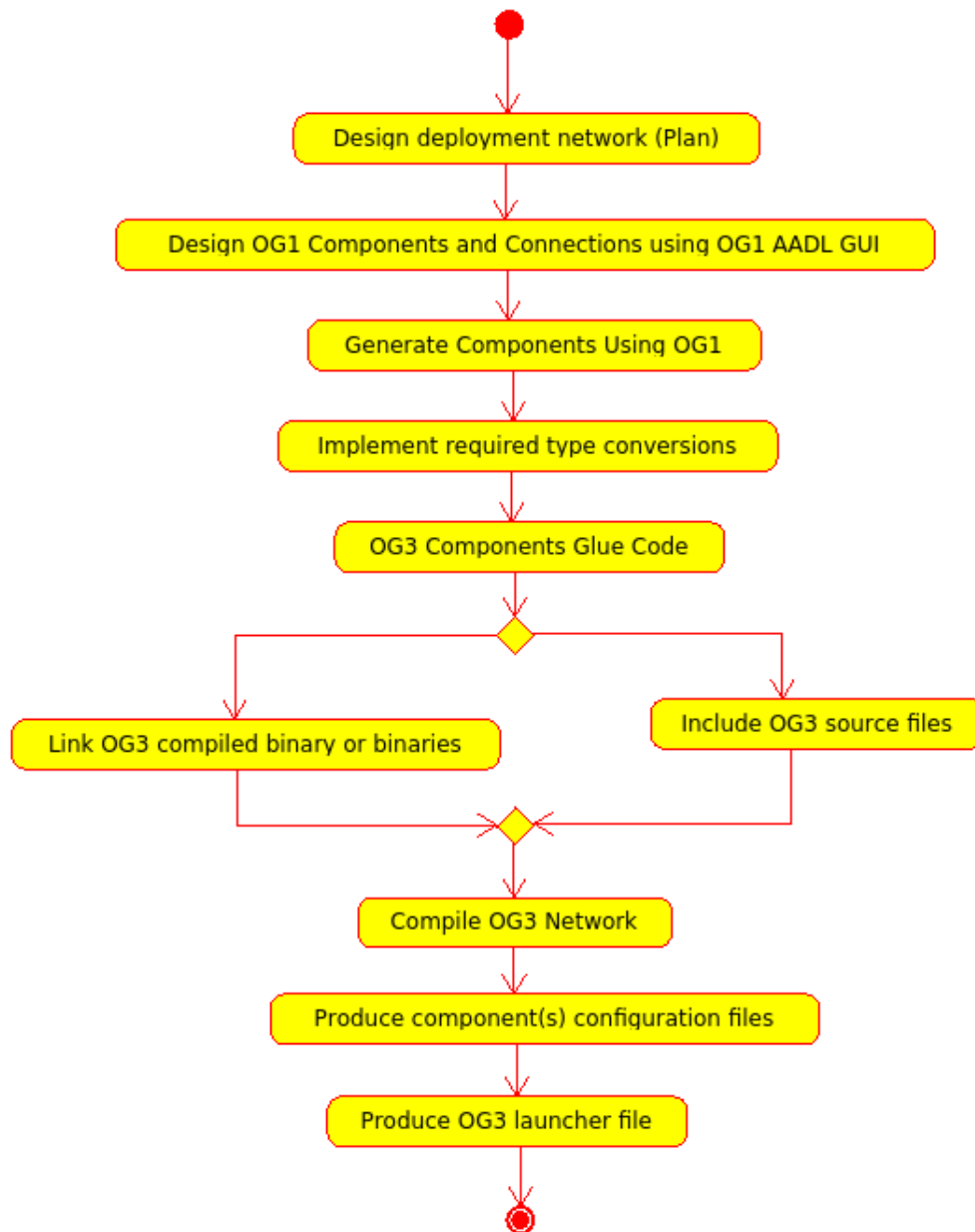


Figure 26: Workflow to integrate OG3 in OG1



### 5.3.1 I/F Requirements

#### **OG3-OG1/S/001/001/V0.1**

Data View: Data that will be transmitted between OG1 components will be represented by ASN.1 types. OG1 provides an interface to design new data types that are transmitted between components.

#### **OG3-OG1/S/001/002/V0.1**

A predefined set of types that can be used by OG3 will be provided by OG1 (base types).

#### **OG3-OG1/S/002/001/V0.1**

Interface view: OG1 provides a graphical user interface to design component networks. It will be possible to generate a C++ code scaffold for each component of the network. Code scaffolds will be filled by OG3 with glue code (e.g. type conversions) so that the components that will be developed in OG3 (Data Product Manager, the Orchestrator and Data Fusion Processing Compounds) are available in OG1. For the cases where InFuse components use internally ASN1 types, less glue code will be needed when integrating with ESROCOS (e.g. type conversions not needed).

#### **OG3-OG1/S/002/002/V0.1**

There are multiple options how the components developed in OG3 can be mapped to OG1 modules:

1. Data Product Manager (DPM), Orchestrator, and Data Fusion Processing Compounds (DFPC) form a single OG1 module.
  2. DPM and Orchestrator are in separate OG1 module. Each DFPC is in a separate OG1 module.
  3. The Orchestrator is in a separate module, the DFPCs and the DPM are in a single module.
  4. DFPCs can be fragmented to the point of having each data fusion node as an OG1 module
- OG3 won't control the scheduling of any OG1 module (e.g. stopping a module).

#### **OG3-OG1/S/003/001/V0.1**

OG1 component lifecycle: OG1 components provide a predefined component life cycle that includes a configuration and a runtime state.

#### **OG3-OG1/S/003/002/V0.1**

It must be possible to set the configuration of a OG1 node at least once before processing begins.

## D4.2: Advanced CDFF Architecture and ICD

---

### OG3-OG1/S/004/001/V0.1

OG1 provides a library for geometric coordinate frame transformations.

### OG3-OG1/S/004/002/V0.1

OG1 provides a forward and instantaneous kinematics computation library.

### OG3-OG1/S/005/001/V0.1

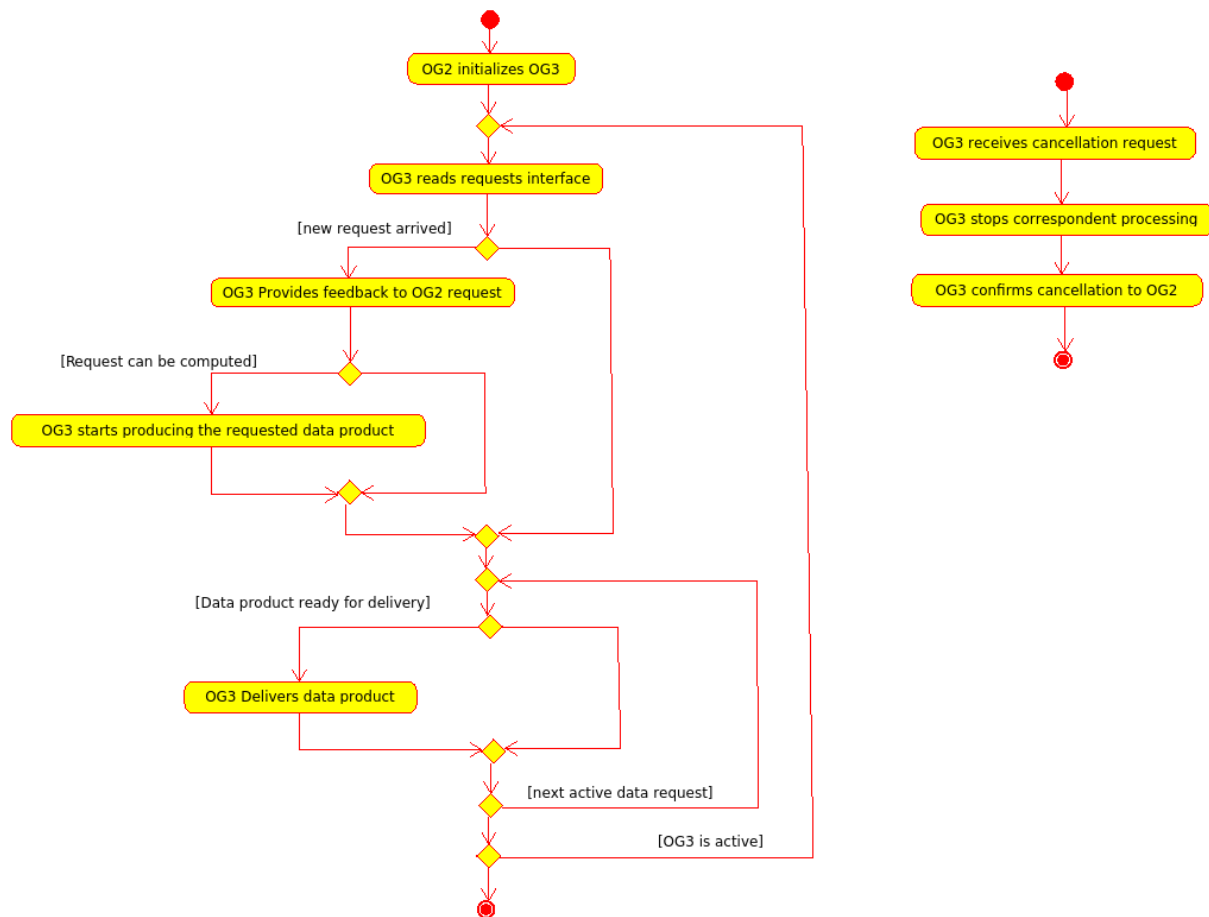
Deployment view: The component network that has been defined in the interface view must be compiled to an executable program. OG1 must provide the functionality.

### OG3-OG1/S/005/002/V0.1

Build system: Components that are developed in OG3 will either be compiled to a single static library that can be linked into an OG1 node or all the necessary C++ headers and implementation files will be available so that they can be used to compile the OG1 component.

## 5.4 Interface with OG2

OG2 is responsible for planning during the complete mission. The internal state of OG3 needs to be initialized to begin accepting requests to start DF processes. As long as the mission continues, OG2 can send to OG3 to generate fused data products (maps or poses). These request will be done through a generic querying method. Once the request has been received and processed, OG3 will either (1) confirm that the request will be performed, or (2) in case that the request cannot be performed, report to OG2 with the correspondent message. After generating the correspondent data product, it is delivered to OG2. The delivered data product can be a periodic data update or a unique data product. The streams are provided to OG2 synchronously (possibly a polling mechanism to check if the data is available) interface which is univocally associated to the data product requested. At any point after performing a request, OG2 can send a cancellation of that same request. In Figure 26 the communication workflow between OG2 and OG3 is presented.



**Figure 27: Communication workflow between OG2 and OG3. On the left diagram the main operational loop is presented. On the right one, the operation to stop a request.**

### **5.4.1 I/F Requirements**

#### **OG2-OG3/S/001/002/V0.1**

OG3 provides an interface to OG2 to initialize, reset or put the CDFF framework into an idle state (no processing). This can be used by OG2 to command the runtime internal state of the data fusion framework (not to be confused with the OG1 run time states). The specific parameters could be desired state of system:

1. Initialize - Inform CDFF to be prepared for accepting requests from OG2
2. Idle - Inform CDFF to stop processing DFPCs from previous requests
3. Reset - Inform CDFF to clear the previous OG2 requests that are currently being processed by stopping the corresponding DFPCs. This will enable OG2 to send a completely new set of requests.

#### **OG2-OG3/S/002/001/V0.1**

OG3 provides an interface for OG2 to query and get a notification on the current state of the CDFF framework to OG2. This can be used by OG2 to check the state before posting a new request that influences the internal runtime state of OG3 as indicated in OG2-OG3/S/001/001/V0.1. The return type is the current state of the system with associated data. This needs to be elaborated in further iterations.

#### **OG2-OG3/S/003/001/V0.1**

OG3 provides an interface for OG2 to query the CDFF orchestrator based on a set of criteria of parameters for activating DFPCs. OG3 internally maintains a list of possible CDFFs that can be selected indirectly by OG2 based on a specific set of parameters such as:

1. List of sensors activated by OG2
2. Default configurations or operational modes of sensors
3. Type of fused maps (rover map, fused rover map and fused total map)
4. Resolution of fused data (maps)
5. Update frequency (localization and map updates)
6. Area coverage (map)
7. Time bounds for providing specific fused data for aperiodic requests
8. Priority of request (to manage request queue in OG3)

The match between OG2 request and the selection of OG4's mode of operation will be clear and explicit. All of this will be pre-defined, for each reference implementation, so that in the scope of the project neither OG2 nor OG3 will have provision decision making mechanisms for what concerns the sensors selection. The decided approach eliminates the possibility of sending to OG4 conflicting operational modes commands from OG2 and OG3.

The orchestrator software component within CDFF has the task to map the criteria to a specific DFPC that can optimally satisfy the above requirements.

#### **OG2-OG3/S/003/002/V0.1**

This interface can be used by OG2 to get an update regarding the runtime status of the DFPC in terms of the quality of data inputs from OG4 (raw and pre-processed) and the quality of fused data products in terms of maps and localization data.

#### **OG2-OG3/S/004/001/V0.1**

## **D4.2: Advanced CDFF Architecture and ICD**

---

Interface provided by OG2 to OG3 for providing fused data updates. There are 2 mechanisms that are foreseen:

1. For aperiodic requests from OG2 (Ex. absolute localization, global fused map etc.), OG3 will notify OG2 regarding the availability of fused data and provide access to the data buffer with corresponding information on the data type. The notification is foreseen to be asynchronous due the inherent delay in producing fused data for a specific query from OG2.
2. For OG2 requests that require OG3 to generate periodic data (local rover map, high frequency local pose estimates etc.), a predefined data stream for specific data types will be used to send the fused data periodically to OG2.

### **OG2-OG3/D/001/001/V0.1**

This interface produces the Rover Map.

### **OG2-OG3/D/002/001/V0.1**

This interface produces the Fused Rover Map

### **OG2-OG3/D/003/001/V0.1**

This interface produces the Fused Total Map

### **OG2-OG3/D/004/001/V0.1**

This interface produces the LocalPose Pose of the BodyFrame in the LocalTerrainFrame

### **OG2-OG3/D/005/001/V0.1**

This interface produces the 3D model of the target spacecraft.

### **OG2-OG3/D/006/001/V0.1**

This interface produces the GlobalPose Pose of the BodyFrame in the GlobalTerrainFrame

### **OG2-OG3/D/007/001/V0.1**

This interface produces the AbsolutePose Pose of the BodyFrame in the AbsoluteFrame

### **OG2-OG3/D/008/001/V0.1**

#### **D4.2: Advanced CDFF Architecture and ICD**

---

This interface is the relative pose (3 axes position and 3 axes attitude) of the target Body Frame expressed in the chaser Body Frame, with associated uncertainties (format to be defined)

##### **OG2-OG3/D/009/001/V0.1**

This interface is the relative speed (3 axes translation speeds and 3 axes rotation speeds) of the target Body Frame expressed in the chaser Body Frame, with associated uncertainties (format to be defined)

## 5.5 Interface with OG4

Before any interaction between OG4 and OG3 can take place, both modules have to be initialized. It is assumed at this stage that OG2 is responsible for the initialization of the different components and OG1 is the responsible of the initialization of OG2.

As long as OG3 is active, the activities of OG3 will can be summarized as follows: As long as there is an active Data Product Request, OG3 will keep activate the correspondent most suitable *preset* (i.e. configuration values agreed in advance) of OG4 and read the from the correspondent data stream. Each preset has associated OG4 Producer Interfaces (PI) in a unitary manner (i.e. one PI only produces one type of data according to the associated preset). OG3 will then the correspondent PI's associated to the preset. If there is not any active Data Product Request, nothing will be done.

In Figure 27 the communication workflow between OG3 and OG4 is depicted.

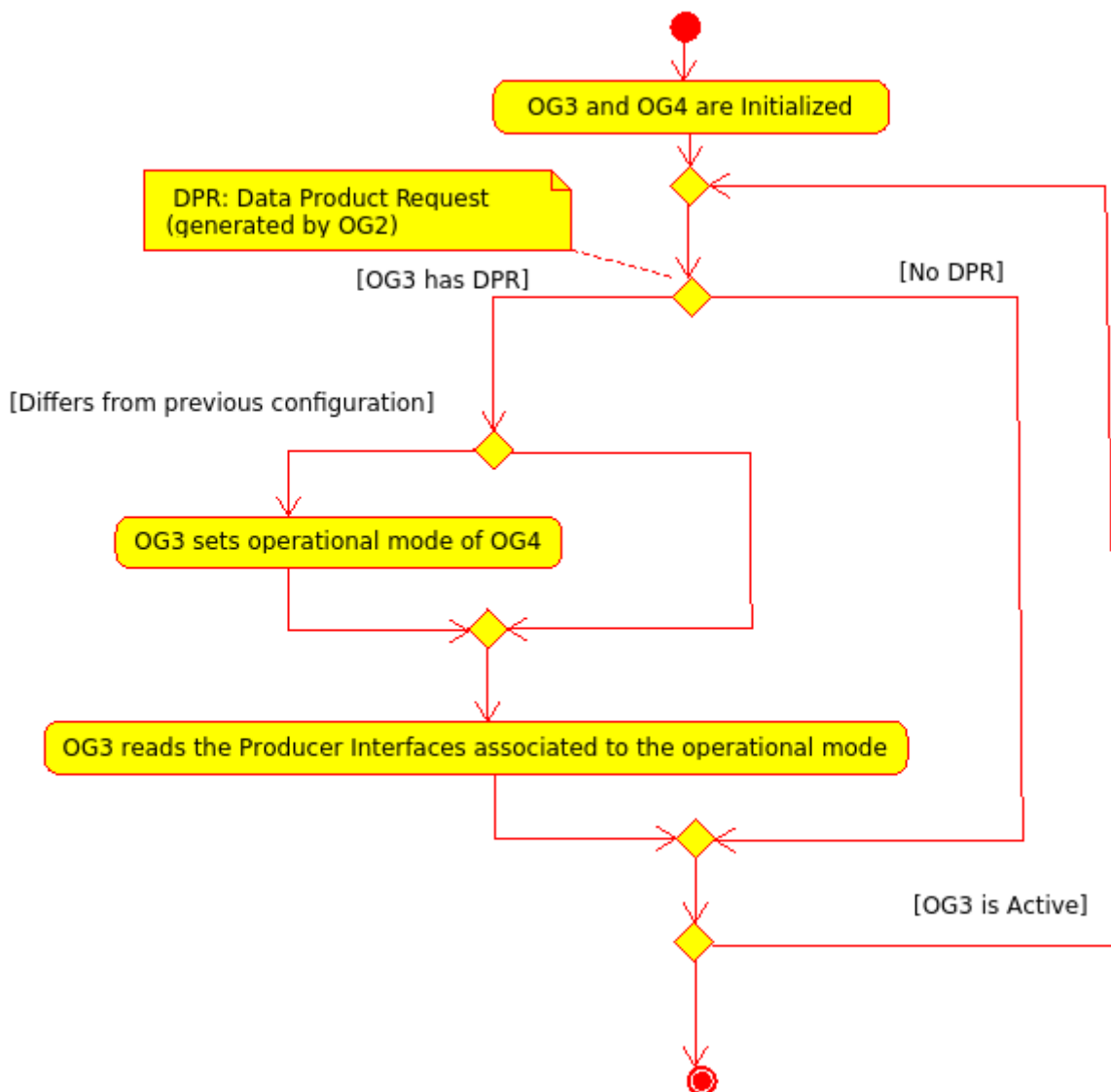


Figure 28: Communication workflow between OG3 and OG4

### 5.5.1 Sensor Data Inputs from OG4

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
HR camera	Image (RGB) Transform metadata must be available.	asn/Frame.asn	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status, calibration parameters  <i>Geometric frame name (not in type definition)</i>	Calibration parameters
Stereo camera	Disparity map	Similar to asn/Frame.asn type for frame but with float values	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status,  <i>Geometric frame name (not in type definition)</i>	<b>The type does not exist</b>
	Depth map	asn/DepthMap.asn	ref-time,	No modifications required

<sup>5</sup> ASN1 types defined in ESROCOS: <https://github.com/ESROCOS/types-base>



#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
			timestamps, vertical-projection, horizontal-projection, vertical-interval, horizontal-interval, vertical-size, horizontal-size, remissions,  <i>Geometric frame name (not in type definition)</i>	
	Point cloud <sup>6</sup> arranged in a so that correspondences between points and pixels and the reference camera (usually the left) are possible	asn/Pointcloud.asn	ref-time, colors,  <i>Geometric frame name (not in type definition)</i>	No modifications required
	Raw images	asn/Frame.asn	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status,	Calibration parameters have to be included

<sup>6</sup> Pointcloud units should be meters

#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
			calibration parameters  <i>Geometric frame name (not in type definition)</i>	
	Corrected images	asn/Frame.asn	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status, calibration parameters, Baseline of the stereovision bench,  <i>Geometric frame name (not in type definition)</i>	Calibration parameters have to be included. Baseline of the stereovision bench has to be included.

#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
LiDAR <sup>7</sup>	Point cloud <sup>8</sup>	asn/Pointcloud.asn	ref-time, colors, <b>reflectances</b> ,  <i>Geometric frame name (not in type definition)</i>	The Reflectances are missing in the type, this metainformation about each point of the pcl can be used to determine information about the object. Some lidars provides sequences of reflectances and not just one. Thus, the sequence of Reflectance objects. The Reflectance object can be defined as: Reflectance SEQUENCE (SIZE(1..maxPointcloudSize)) OF float
	2D laser scan	types-sensor_samples/asn/LaserScan.asn	ref-time, start-angle, angular-resolution, speed, ranges, minRange, maxRange, remission, <b>reflectances</b>  <i>Geometric frame name (not in type definition)</i>	The Reflectances are missing in the type, this metainformation about each point of the pcl can be used to determine information about the object.

#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
TIR camera	Image	asn/Frame.asn	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status, Calibration parameters  <b>spectral_range</b>  <i>Geometric frame name (not in type definition)</i>	The spectral range has to be included it can be defined as:  SEQUENCE (SIZE(1..2)) OF T-Float

<sup>7</sup> Depth map if specific to the COTS sensor

<sup>8</sup> Pointcloud units should be meters

#### D4.2: Advanced CDFE Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
ToF camera	Point cloud <sup>9</sup>	asn/Pointcloud.asn	Intensity Image, Confidence map, ref-time, colors  <i>Geometric frame name (not in type definition)</i>	Intensity Image and Confidence map have to be included in a new type extending the Point Cloud  For the intensity image and the Confidence map the asn/Frame.asn type can be used.
Radar	Point cloud <sup>10</sup>	asn/Pointcloud.asn	ref-time, colors  <i>Geometric frame name (not in type definition)</i>	No modifications required

<sup>9</sup> Pointcloud units should be meters

<sup>10</sup> Pointcloud units should be meters

#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
Structured light camera	Depth map	asn/DepthMap.asn	ref-time, timestamps, vertical-projection, horizontal-projection, vertical-interval, horizontal-interval, vertical-size, horizontal-size, remissions  <i>Geometric frame name (not in type definition)</i>	No modifications required
	Raw image	asn/Frame.asn	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status, calibration parameters, Baseline of the stereovision bench,  <i>Geometric frame name (not in type definition)</i>	Calibration parameters have to be included.

#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
Structured light camera	Corrected images	asn/Frame.asn	frame-time, received-time, attributes, datasize, data-depth, pixel-size, row-size, frame-mode, frame-status, calibration parameters, Baseline of the stereovision bench,  <i>Geometric frame name (not in type definition)</i>	Calibration parameters have to be included.
IMU	Inertial measurements	asn/IMUSensors.asn	Timestamp  <i>Geometric frame name (not in type definition)</i>	No modifications required
End effector force/torque sensor	Linear forces or torques at end effector	Asn1 type for Wrench <sup>11</sup>	Timestamp, Geometric frame name	No modifications needed

<sup>11</sup> ESROCOS ASN1 Type for Wrench: <https://github.com/ESROCOS/types-base/blob/master/asn/Wrench.asn>

#### D4.2: Advanced CDFF Architecture and ICD

Sensor Type	Data type	ESROCOS ASN 1 data type <sup>5</sup>	Metadata	Modification required in the ASN1 type
Start tracker	Rotation with respect to tracked star.	asn/RigidBodyState.asn	timestamp, sourceFrame, targetFrame	No modification required

**Table 5: Sensor data inputs from OG4**



#### D4.2: Advanced CDFD Architecture and ICD

Metadata Type	Data Type	Asn1 Data Type	Modification required in the ASN1 type definition
Time	Time	asn/Time.asn	No modifications required
Frame-size-t	Width height	types-sensor_samples/asn/Frame.asn	No modifications required
Frame-mode-t,	mode-undefined, mode-grayscale, mode-rgb, mode-uyvy, mode-bgr, mode-rgb32, raw-modes, mode-bayer, mode-bayer-rggb, mode-bayer-grbg, mode-bayer-bggr, mode-bayer-gbrg, compressed-modes, mode-pjpg, mode-jpeg, mode-png	asn/Frame.asn	No modifications required
Frame-status-t	status-empty, status-valid, status-invalid	asn/Frame.asn	No modifications required
Frame-attrib-t	data T-String, att-name T-String	asn/Frame.asn	No modifications required
Calibration Parameters	4 floats for the intrinsic matrix: - alpha_u, alpha_v, u_0, v_0 5 floats for the distortion: - k1, k2, k3, t1, t2		Type doesn't exists in ASN1 definitions

**Table 6 Metadata Data Types**

## 5.5.2 Interfaces Requirements

### OG3-OG4/D/001/000/V0.1

OG3 supports ASN.1 messages defined by OG4 following OG1 specification for all communication with OG4.

### OG3-OG4/D/002/000/V0.1

OG4 provides data samples (simulated and actual) to OG3 throughout the course of the project.

### OG3-OG4/F/001/000/V0.1

OG4 I3DS ensures compatibility with OG3 Common Data Fusion Framework.

### OG3-OG4/F/002/000/V0.1

OG4 supports operational modes as the primary way of controlling the sensor suite. Each mode consists of a set of active sensors with a given configuration that is verified to operate correctly with regards to throughput and latency.

### OG3-OG4/F/003/000/V0.1

OG4 provides a list of specific modes per use-case to operate the sensor suite ensuring the limitations of available data rates, bandwidth and power consumption.

### OG3-OG4/F/004/000/V0.1

OG4 receives commands from OG3 to enable pre-defined sensors configurations and sensors data provisions.

### OG3-OG4/F/005/000/V0.1

## D4.2: Advanced CDF Architecture and ICD

---

OG4 provides sensor measurements and accepts commands through the OG1 middleware. The supported sensors are listed in the product tree.

### OG3-OG4/F/006/000/V0.1

OG4 performs pre-processing for the Time Of Flight Camera, the Stereo Camera, the High Resolution Camera, the Thermal InfraRed Camera, the High Frequency Radar and the Lidar. **OG3 will have access to the intermediate products.**

### OG3-OG4/F/007/000/V0.1

OG4 performs the following pre-processing for the Time of Flight Camera: Point Cloud Generation. **OG3 will have access to the intermediate products. In this case, Depth Map.**

### OG3-OG4/F/008/000/V0.1

OG4 performs the following pre-processing for the Stereo Camera: Lens Vignetting, Histogram Equalisation, Optical Distortion (Lens radial and geometric distortion correction), Stereo Rectification, Dense Stereo Matching Disparity Map Production (TBD), Sparse Stereo Matching (TBD), Dense Depth Map Production (TBD), Sparse Depth Point Cloud (TBD). **OG3 will have access to the intermediate products.**

### OG3-OG4/F/009/000/V0.1

OG4 performs the following pre-processing for the High Resolution Camera: Lens Vignetting Correction, Histogram Equalisation, Optical Distortion Correction, Feature Detection and Temporal Matching, Structured Light Pattern Detection, Depth Map Point Cloud Production from Structured Light (TBD). **OG3 will have access to the intermediate products.**

### OG3-OG4/F/010/000/V0.1

OG4 performs the following pre-processing for the Thermal InfraRed Camera: Lens Vignetting Correction, Histogram Equalisation, Optical Distortion Correction, Feature Detection and Temporal Matching. **OG3 will have access to the intermediate products.**

### OG3-OG4/F/011/000/V0.1

OG4 performs the following pre-processing for the HF Radar: Point Cloud Generation. The radar information could be used for long or short distance mapping.

## **D4.2: Advanced CDFF Architecture and ICD**

---

### **OG3-OG4/F/012/000/V0.1**

OG4 performs the following pre-processing for the LIDAR: Point Cloud Generation.

### **OG3-OG4/F/013/000/V0.1**

OG4 provides to OG3 either raw sensors data or pre-processed data. Both cannot be performed simultaneously.

### **OG3-OG4/F/014/000/V0.1**

OG4 performs time stamping for pre-processed and raw data.

### **OG3-OG4/O/001/000/V0.1**

The operational mode can become parameters to be configured in the future.

### **OG3-OG4/O/002/000/V0.1**

OG4 does not check the sender of command messages; therefore OG2 and OG3 must coordinate commanding of OG4 to avoid conflicting commands.

### **OG3-OG4/O/003/000/V0.1**

OG4 does not store measurements for later retrieval; therefore OG3 receives the data at the specified rate<sup>12</sup>.

### **OG3-OG4/P/001/001/V0.1**

---

<sup>12</sup> Related Issue: Not clear if OG1 or OG2 controls the communication buffers between OG4 and OG3.

## **D4.2: Advanced CDF Architecture and ICD**

---

Before any interaction between OG4 and OG3 can take place, both modules have to be initialized. It is assumed at this stage that OG2 is responsible for the initialization of the different components and OG1 is the responsible of the initialization of OG2.

### **OG3-OG4/S/001/001/V0.1**

An interface is required for OG3 to be able to configure a limited set of parameters of a group of sensors providing data to a DFPC or alternatively to select among pre-defined configuration modes that are compatible with a DFPC.

After OG2 triggers a set of sensors for a particular task, the default configuration of the sensors might not be optimal for data fusion due to changing conditions (Ex. ambient lighting, camera exposure etc.) that produce multiple outliers or unusable data (Ex. saturated images, lack of correspondence features etc.).

This interface aims to provide a mechanism for OG3 to set a suitable configuration for some limited parameters within each operational mode of OG4 for some sensors relevant for a DFPC to produce the required quality of fused data products (e.g. exposure of the camera or framerate).

### **OG3-OG4/S/001/002/V0.1**

OG4 provides raw or pre-preprocessed sensor data to OG3 from the sensors through this interface. The sensors and the corresponding pre-preprocessing steps are configured by the function OG3-OG4/S/001/001/V0.1.

### **OG3-OG4/S/001/003/V0.1**

The preprocessing steps that the data has gone through in OG4 is known to OG3 because it is associated to the OG4 sensors operation mode selected by function OG3-OG4/S/001/002/V0.1. The software interfaces through which each sensor data is delivered, is defined in the operation mode of OG4. For each pre-processed sensor data there is an associated software interface which is only used for that pre-processed sensor data and is not used in any other operation mode unless exactly the same pre-processed sensor data is produced.

### **OG3-OG4/S/002/001/V0.1**

OG 4 provides metadata related to the pre-processing steps the sensor data has gone through. The metadata is provided through a different interface than the data itself and it is timestamped.

### **OG3-OG4/S/002/002/V0.1**

#### D4.2: Advanced CDFF Architecture and ICD

---

OG4 metadata to OG3 from the sensors and the sensor filters as configured by the function OG3-OG4/S/001/001/V0.1. In the configuration is included the interface for each metadata product.

##### **OG3-OG4/S/003/001/V0.1**

Once OG4 sends any data over the data interface, if OG3 is not ready to process the data, this will be lost. An interface is therefore required for OG3 to inform OG4 that a specific DFPC has been selected based on an OG2 request (OG2-OG3/S/003/001/V0.1) and that the DFPC is in a ready state to process sensor data from OG4. Hence this mechanism allows OG4 to send data after the DFPC is initialized and ready to process sensor data to begin producing fused data as outputs.

## **5.6 Rover and Manipulation Interface**

### **5.6.1 Sensor Data Inputs from OG6**

Some sensors which are typically available in robotics systems are currently not included in OG4 sensor's suite. Thus, an additional interface is defined which should be provided for validation by either (1) OG6, the facilitators framework, by (2) OG2 or (3) OG4.

Sensor Type	Data type	ASN 1 data type <sup>13</sup>	Metadata	Modifications Required in the ASN1 Type
Manipulator joint states	Joint angle positions and velocity	Asn1 type for JointState <sup>14</sup>	Timestamp, Geometric frame name	Position resolution Velocity resolution
Rover wheel encoders	Position of the joints of the wheels	Asn1 type for Joints <sup>15</sup>	Timestamp, Names, Geometric frame names	Position resolution Velocity resolution
Manipulator force/torque sensor	Linear forces or torques at joint	Asn1 type for Wrench <sup>16</sup>	Timestamp, Geometric frame name	No modifications needed
Manipulator force/torque sensors	Linear forces or torques at joints composition	Asn1 type for Wrenches <sup>17</sup>	Timestamp, Names, Geometric frame names	No modifications needed
Wheel-Based Odometry	Global pose estimate and differential pose estimate (based on wheel encoder measurements and rover model)	Asn1 type for RigidBodyState <sup>18</sup>	Timestamp, Source Frame, Target Frame	No modifications needed

**Table 7 Data inputs from the Rover and Manipulation interface**

<sup>13</sup> ASN1 types defined in ESROCOS: <https://github.com/ESROCOS/types-base>

<sup>14</sup> ESROCOS ASN1 type for JointState: <https://github.com/ESROCOS/types-base/blob/master/asn/JointState.asn>

<sup>15</sup> ESROCOS ASN1 Type for Joints: <https://github.com/ESROCOS/types-base/blob/master/asn/Joints.asn>

<sup>16</sup> ESROCOS ASN1 Type for Wrench: <https://github.com/ESROCOS/types-base/blob/master/asn/Wrench.asn>

<sup>17</sup> ESROCOS ASN1 Type for Wrenches: <https://github.com/ESROCOS/types-base/blob/master/asn/Wrenches.asn>

<sup>18</sup> ESROCOS ASN1 Type for RigidBodyState: <https://github.com/ESROCOS/types-base/blob/master/asn/RigidBodyState.asn>



---

## 5.6.2 I/F Requirements

### **OG3-OG6/D/001/000/V0.1**

OG6 supports ASN.1 messages defined by OG6 following OG1 specification for all communication with OG6.

### **OG3-OG6/F/001/000/V0.1**

OG6 provides raw or pre-processed sensor data to OG3 from the manipulator Joints through this interface.

### **OG3-OG6/F/002/000/V0.1**

OG6 provides raw or pre-processed sensor data to OG3 from the Rover Wheel encoders through this interface.

### **OG3-OG6/F/003/000/V0.1**

OG6 provides raw or pre-processed sensor data to OG3 from a particular manipulator force/torque sensor through this interface.

### **OG3-OG6/F/004/000/V0.1**

OG6 provides raw or pre-processed sensor data to OG3 from sets of manipulator force/torque sensors through this interface

### **OG3-OG6/F/005/000/V0.1**

OG6 provides global pose estimation based on the rover encoders and on the robot geometric model to OG3 through this interface

### **OG3-OG6/F/005/001/V0.1**

OG6 provides differential pose estimation based on the rover encoders and on the robot geometric model to OG3 through this interface

### **OG3-OG6/F/001/000/V0.1**

OG4 provides sensor measurements and accepts commands through the OG1 middleware. The supported sensors are listed in the product tree.

## **6 Conclusion**

The architecture and ICD of InFuse are the key components of this deliverables. As of the PDR milestone, an advanced version of these elements were released in this document, that should serve as baseline to carry out a detailed design work in the next work package (dealing with detailed design). By the CDR, it is intended to have a very detailed specification of software components to be implemented, to the level of classes, detailed logics and data flows, etc. using on UML.

On the ICD side, it is still expected that further refinements may occur until the CDR, as the assumptions expressed in this document will be further harmonized with other OGs' ones.

## **Appendix 1: Glossary**

### ***Target System***

The robotic system that will be used for the final mission. This includes also the RCOS under which the different modules run -OG1 in our case-.

### ***Developer's Environment***

The setup that the designer of perception solutions uses to evaluate different approaches. Once a solution has been designed and implemented, it can be exported to the target system.

### ***CDFF-Core***

Part of the CDFF which contains the set of techniques for data fusion implemented in a modular fashion with a Data Fusion Common Interface (DFNCI) to allow high flexibility in configuration as well as in operation as a distributed system on multiple platforms and libraries dedicated to data filtering and outliers removal.

### ***CDFF-Support***

Part of the CDFF which Includes orchestration, data product management. This part of the CDFF is integrated in the final Target System.

### ***CDFF-Dev<sup>19</sup>***

Not integrated in the target system. Encapsulates all the features of the CDFF which are used only in the designer environment (i.e. won't be integrated in the target system).

### ***Data Fusion Node (DFN)***

Is a software library for data fusion which is defined in the CDFF-Core. These libraries are independent of the CDFF itself but as units might not provide a complete data fusion solution.

### ***Data fusion processing compound<sup>20</sup> (DFPC)***

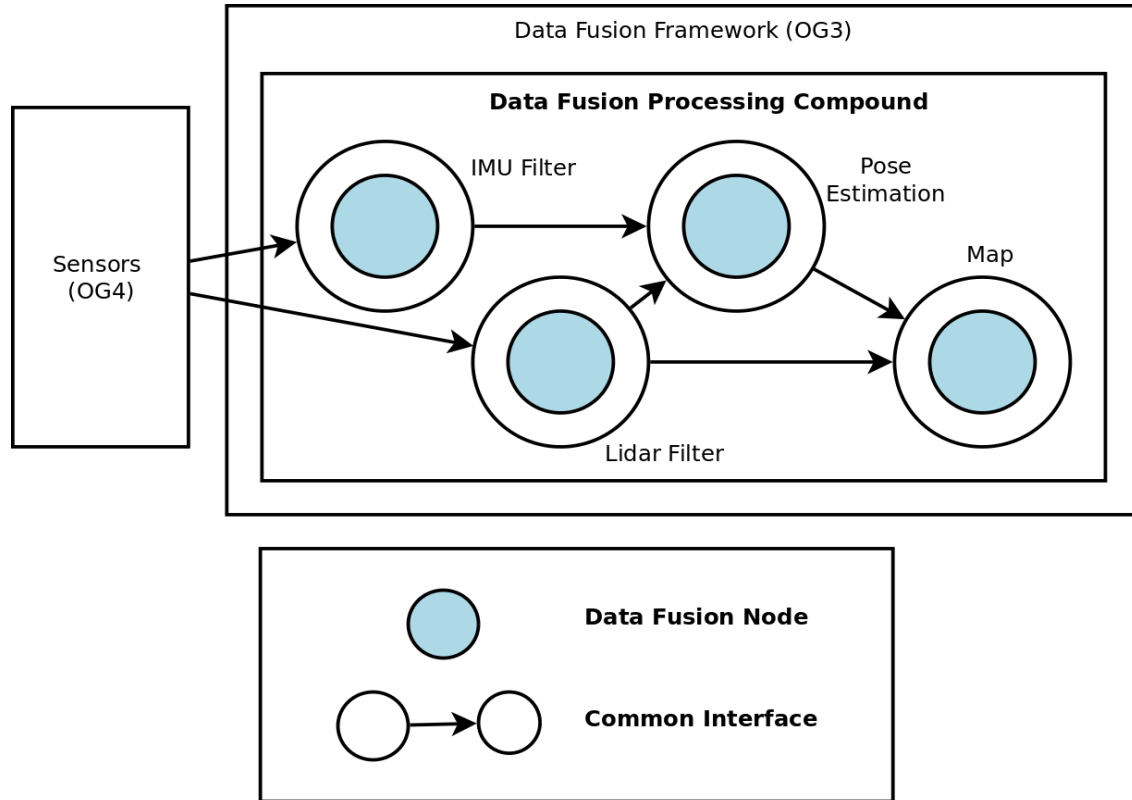
Libraries connected through a DFNCI by the CDFF. A DFPC is designed to provide a certain data product (e.g. pose estimation, map...). A same DFPC can be implemented with different programming languages and can be executed on multiple RCOS, though it must be adapted to the specificities of these. In Figure 28 a simplified DFPC to perform mapping is presented in the context of OG3. A DFPC is composed CDFF-Core Libraries and CDFF-Support DFPC. The DFPC implementation will differ depending on the target RCOS but the functionality should be the same.

---

<sup>19</sup> Named CDFF-Utils in previous documents

<sup>20</sup> Named Data Fusion Processing Chain in previous documents

## D4.2: Advanced CDFF Architecture and ICD



**Figure 29: Example of a DFPC. A DFPC receives sensor data as inputs (provided by OG4) and produces Data Fusion Products (delivered to OG2 by the Orchestrator).**

### **Data Product Management tool (DPM)**

The series of processes that select, structure and store the raw data, intermediary data products and final data products acquired and generated within InFuse.

### **Operating System (OS)**

Software that provides basic computing system functions and an interface to hardware, a Real-Time Operating System (RTOS) implies hard real-time operation.

### **RCOS or Robotics Middleware**

Software framework that facilitates the integration of multiple libraries, supports communications, enables configurations and provides control over them. Furthermore, the RCOS provides control over operating system of the controlled system and to its sensors and actuators.

## Appendix 2: InFuse CDFF Product Tree

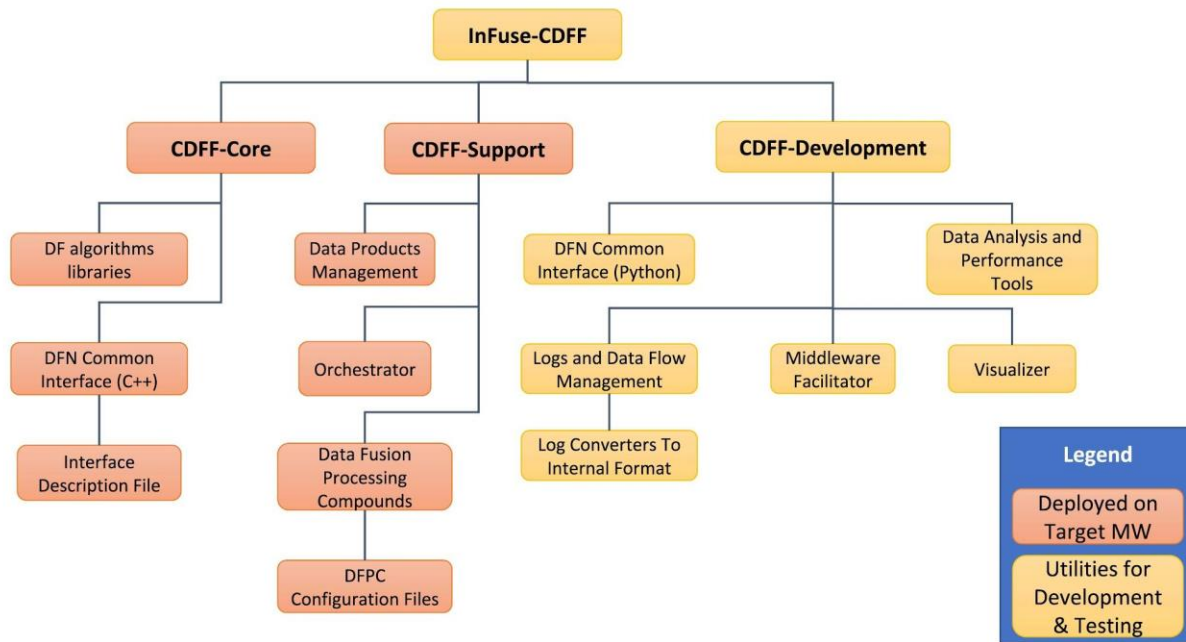


Figure 30: Product Tree of InFuse Common Data Fusion Framework

## Appendix 3: Libraries and Data Types

To implement the different DFNs and support components a list of libraries which the consortium is considering using is provided.

Library name	Description	Link
Open CV	Library for Image Processing.	<a href="http://opencv.org/">http://opencv.org/</a>
PCL	Library for 3D pointclouds processing	<a href="http://pointclouds.org/">http://pointclouds.org/</a>
OctoMap	3D maps library	<a href="https://octomap.github.io/">https://octomap.github.io/</a>
ESROCOS Base Types	Types defined in ASN1 which OG1 supports	<a href="https://github.com/ESROCOS/types-base">https://github.com/ESROCOS/types-base</a>
Base Types	Library for types	<a href="https://github.com/rock-core/base-types">https://github.com/rock-core/base-types</a>
Slam/Maps	Maps Types	<a href="https://github.com/envire/slam-maps">https://github.com/envire/slam-maps</a>
G2O	Graph Optimization Framework	<a href="https://github.com/RainerKuemmerle/g2o">https://github.com/RainerKuemmerle/g2o</a>
URDF	Robot Model Library	<a href="https://github.com/ros/robot_model">https://github.com/ros/robot_model</a>
EnviRe	Libraries for Environment Representation and Visualization	<a href="http://envire.github.io/">http://envire.github.io/</a>
Vizkit 3D	Visualization Plugins	<a href="https://github.com/rock-gui/gui-vizkit_3d_plugins">https://github.com/rock-gui/gui-vizkit_3d_plugins</a>
Eigen 3	Library for linear algebra: matrices, vectors, numerical solvers and related algorithms	<a href="http://eigen.tuxfamily.org">http://eigen.tuxfamily.org</a>
LIBSVM	A library for support vector machines	<a href="https://www.csie.ntu.edu.tw/~cjlin/libsvm/">https://www.csie.ntu.edu.tw/~cjlin/libsvm/</a>
Aquila, sigpack, DSPFilters	Libraries for signal processing which will be studied	<a href="http://aquila-dsp.org">http://aquila-dsp.org</a> , <a href="http://sigpack.sourceforge.net">http://sigpack.sourceforge.net</a> , <a href="https://github.com/vinniefalco/DSPFilters">https://github.com/vinniefalco/DSPFilters</a>

**Table 8: Tentative list of libraries and type definitions that will be used in InFuse**

## Appendix 4: Requirements Traceability Matrix

The requirements were organized in the System Requirements Document (deliverable 3.2) according to a previous categorization of the features. In this appendix the requirements remain organized as previously to facilitate tracking with the previous document.

### CDFF Core

#### User Requirements (UserR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
SR_UserR_A101	Yes	4.1.2.1	
SR_UserR_A102	Yes	4.1.2	
SR_UserR_A103	Yes	4.1.2	
SR_UserR_A104	Yes	4.1.2	
SR_UserR_A105	Yes	4.1.2	
SR_UserR_A106	Yes	4.1.2	
SR_UserR_A107	Yes	4.1.2	
SR_UserR_A108	Yes	4.1.2.4	
SR_UserR_A109	Yes	4.1.2	
SR_UserR_A110	Yes	4.1.2	
SR_UserR_A111	Yes	4.1	
SR_UserR_A112	Yes	4.2	
SR_UserR_A113	Yes	4.1.2	

#### D4.2: Advanced CDFF Architecture and ICD

<b>SR_UserR_A114</b>	Yes		
<b>SR_UserR_A115</b>	Yes	4.2.2	
<b>SR_UserR_A116</b>	Partial. Visualization is only provided on the Developer's Environment	4.3.5	
<b>SR_UserR_A117</b>	Yes	4.1.	

#### Functional Requirements (FuncR)

<b>Req. ID</b>	<b>Compliance (yes / no / partial)</b>	<b>Section where it is addressed</b>	<b>Comments</b>
<b>SR_FuncR_A201</b>	yes, not explicitly addressed	4.1.2	
<b>SR_FuncR_A202</b>	yes, not explicitly addressed	4.1.2	
<b>SR_FuncR_A203</b>	yes, not explicitly addressed	4.1.2	
<b>Req. ID</b>	<b>Compliance (yes / no / partial)</b>	<b>Section where it is addressed</b>	<b>Comments</b>
<b>SR_FuncR_A204</b>	yes, not explicitly addressed	4.1.2	
<b>SR_FuncR_A205</b>	Partial	4.1.2	Question on OG6 / GNC with the orbital track => answer from PSA send by PSA on



#### D4.2: Advanced CDFE Architecture and ICD

			Thursday, 23 March, 2017 7:31:46 PM
SR_FuncR_A206	yes, not explicitly addressed	4.1.2	
SR_FuncR_A207	yes, not explicitly addressed	4.1.2	
SR_FuncR_A208	yes, not explicitly addressed	4.1.2	
SR_FuncR_A209	yes, not explicitly addressed	4.1.2	
SR_FuncR_A210	yes, not explicitly addressed	4.1.2	
SR_FuncR_A211	yes, not explicitly addressed	4.1.2	
SR_FuncR_A212	yes, not explicitly addressed	4.1.2	
SR_FuncR_A213	yes, not explicitly addressed	4.1.2	
SR_FuncR_A214	yes, not explicitly addressed	4.1.2	
SR_FuncR_A215	Yes	5.5.1	
SR_FuncR_A216	Yes	5.5.1	
SR_FuncR_A217	Yes	5.5.1	
SR_FuncR_A218	Yes	5.5.1	
SR_FuncR_A219	Yes	5.5.1	

#### D4.2: Advanced CDFF Architecture and ICD

SR_FuncR_A220	Yes	5.5.1	
SR_FuncR_A221	Yes, not explicitly addressed		
SR_FuncR_A222	Yes	5.5.1	
SR_FuncR_A223	Yes	4.1.2.4	
SR_FuncR_A224	Yes	4.1.2.3	
SR_FuncR_A225	Yes	4.1.2.2	
SR_FuncR_A226	Yes	4.1.2.3	
SR_FuncR_A227	Yes, not explicitly addressed		
SR_FuncR_A228	Partially		The InFuse architecture and functions will be able to support such scenarios, but will be partially demonstrated, only within the InFuse consortium.

#### Performance Requirements (PerfR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
SR_PerfR_A301	Yes	3.3, 4.2.1., 4.2.2	
SR_PerfR_A302	Yes	3.3, 4.2.1., 4.2.2	

#### D4.2: Advanced CDFF Architecture and ICD

<b>SR_PerfR_A303</b>	Yes	3.3, 4.2.1., 4.2.2	
<b>SR_PerfR_A304</b>	Yes	3.3, 4.2.1., 4.2.2	
<b>SR_PerfR_A305</b>	Yes	3.3, 4.2.1., 4.2.2	
<b>SR_PerfR_A306</b>	Yes	3.3, 4.2.1., 4.2.2	
<b>SR_PerfR_A307</b>	Yes	3.3, 4.2.1., 4.2.2	
<b>SR_PerfR_A308</b>	Yes	3.3, 4.2.1., 4.2.2	

#### Interface Requirements (IntR)

<b>Req. ID</b>	<b>Compliance (yes / no / partial)</b>	<b>Section where it is addressed</b>	<b>Comments</b>
<b>SR_IntR_A403</b>	Yes	4.1.1, (5.3)	
<b>SR_IntR_A404</b>	Yes	4.1.1, (5.5.1)	
<b>SR_IntR_A405</b>	Yes	4.1.1.1	
<b>SR_IntR_A406</b>	Yes	4.2.3	
<b>SR_IntR_A407</b>	Yes	4.2.3	
<b>SR_IntR_A408</b>	Yes	4.2.3	
<b>SR_IntR_A409</b>	Yes	4.2.3	

## D4.2: Advanced CDFF Architecture and ICD

---

### Resource Requirements (ResR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
SR_ResR_A501	Yes	4.1.1	
SR_ResR_A502	Yes, not explicitly addressed		
SR_ResR_A503	Yes	4.1.2	
SR_ResR_A504	Yes	4.1.2	
SR_ResR_A505	Yes	4.1.2	

### Operational Requirements (OpR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
SR_OpR_A601	Yes	4.2.2	
SR_OpR_A602	partial		OG1 related. Inspection utilities should be provided by the RCOS
SR_OpR_A603	Yes	4.2.2	

#### D4.2: Advanced CDFF Architecture and ICD

<b>SR_OpR_A604</b>	partial, configuration	via		
--------------------	------------------------	-----	--	--

#### Product assurance and safety requirements (ProdR)

Not Applicable.

#### Configuration and implementation requirements (ConfR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
<b>SR_ConfR_A801</b>	Yes	4.1.1, 5.4	
<b>SR_ConfR_A802</b>	Yes	4.1.1, 5.5	
<b>SR_ConfR_A803</b>	Yes	?	
<b>SR_ConfR_A804</b>	Yes	?	
<b>SR_ConfR_A805</b>	Yes	?	
<b>SR_ConfR_A806</b>	Yes	?	
<b>SR_ConfR_A807</b>	Partial,	5.5	depends on RCOS

#### Test and Validation (ValR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
---------	------------------------------------	----------------------------------	----------

#### D4.2: Advanced CDFF Architecture and ICD

<b>SR_ValR_A901</b>	Yes	-	
<b>SR_ValR_A902</b>	Yes	-	
<b>SR_ValR_A903</b>	Yes	In Related Document D4.1	TBD?
<b>SR_ValR_A921</b>	Yes	-	
<b>SR_ValR_A922</b>	Yes	In Related Document D4.1	
<b>SR_ValR_A923</b>	Yes	-	
<b>SR_ValR_A924</b>	Yes	-	TBD
<b>SR_ValR_A951</b>	Yes	In Related Document D4.1	
<b>SR_ValR_A961</b>	Yes	In Document D4.1	

#### CDFF Dev

##### User Requirements (UserR)

<b>Req. ID</b>	<b>Compliance (yes / no / partial)</b>	<b>Section where it is addressed</b>	<b>Comments</b>
<b>SR_UserR_B101</b>	Yes	4.3.3	
<b>SR_UserR_B102</b>	Yes	4.3.3	
<b>SR_UserR_B103</b>	Yes	4.3.5	
<b>SR_UserR_B104</b>	Yes	4.3.5	

#### D4.2: Advanced CDFF Architecture and ICD

<b>SR_UserR_B105</b>	Partial,	-	OG1 related. Part of the RCOS, depends on the target platform
<b>SR_UserR_B106</b>	Yes	4.1.2	
<b>SR_UserR_B107</b>	Yes	4	

#### Functional Requirements (FuncR)

<b>Req. ID</b>	<b>Compliance (yes / no / partial)</b>	<b>Section where it is addressed</b>	<b>Comments</b>
<b>SR_FuncR_B201</b>	Yes	4.3.3	
<b>SR_FuncR_B202</b>	Yes	4.3	
<b>SR_FuncR_B203</b>	Yes	4.3.5	
<b>SR_FuncR_B204</b>	Yes	4.3.4	The performance analysis can be checked better on the target system with all the frameworks integrated
<b>SR_FuncR_B205</b>	Yes	5.5	
<b>SR_FuncR_B206</b>	Yes	4.2 and 4.3	
<b>SR_FuncR_B207</b>	Yes	4.2.2 and 5.4.1	
<b>SR_FuncR_B208</b>	Yes	4.1.2	
<b>SR_FuncR_B209</b>	Yes	4.3.5	The visualizations are only possible in the Developers

#### D4.2: Advanced CDF Architecture and ICD

			Environment. On the target system they should be supported by the RCOS.
<b>SR_FuncR_B210</b>	Yes	4.3.2	
<b>SR_FuncR_B211</b>	Yes	4.2.2	

#### Performance Requirements (PerfR)

NA.

#### Interface Requirements (IntR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
<b>SR_IntR_B401</b>	Yes	5.2	

#### Resource Requirements (ResR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
<b>SR_ResR_B501</b>	Yes	4.1, 5.3	
<b>SR_ResR_B502</b>	Yes, but depends on DFPC	4.2	
<b>SR_ResR_B503</b>	Yes, not explicitly addressed	-	



#### D4.2: Advanced CDFF Architecture and ICD

---

##### Operational Requirements (OpR)

NA.

##### Product assurance and safety requirements (ProdR)

NA.

##### Configuration and implementation requirements (ConfR)

Req. ID	Compliance (yes / no / partial)	Section where it is addressed	Comments
SR_ConfR_B801	Yes	4.1.1, 4.2.1	