

Project Title	Online Data Intensive Solutions for Science in the Exabytes Era
Project Acronym	ODISSEE
Grant Agreement No.	101188332
Start Date of Project	01/01/2025
Duration of Project	36 Months
Project Website	odissee-project.eu

D2.1 – Composability Specifications Document

Work Package	WP2 – Composable Data Interfaces
Lead Authors (Org)	Damien Gratadour (OP & CNRS)
Contributing Author(s) (Org)	Vladimir Gligorov (CNRS), Christian Perez (INRIA), Chris Broekema (ASTRON)
Due Date	30.06.2025
Date	17.07.2025
Version	Final

Dissemination Level

<input checked="" type="checkbox"/>	PU: Public
<input type="checkbox"/>	PP: Restricted to other programme participants (including the Commission)
<input type="checkbox"/>	RE: Restricted to a group specified by the consortium (including the Commission)
<input type="checkbox"/>	CO: Confidential, only for members of the consortium (including the Commission)

Versioning and contribution history

Version	Date	Author	Notes
0.1	03.03.2025	Damien Gratadour	ToC and V0.1
0.2	16.05.2025	WP2 members	Version circulated to consortium
1.0	17.07.2025	WP2 members	Final version taking into account consortium feedback

Disclaimer

This document contains information which is proprietary to the ODISSEE Consortium. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to a third party, in whole or parts, except with the prior consent of the ODISSEE Consortium.

AI tools may have been used to produce this deliverable. The ODISSEE consortium is ultimately responsible for its content and ensured that every AI generated content has been proof-read and checked if necessary. AI tools have been used within the boundaries of existing laws, in particular regarding the respect of privacy, confidentiality and Intellectual Property Rights.

Table of Contents

Versioning and contribution history	2
Table of Contents	3
List of Tables.....	4
List of Figures	4
Executive Summary.....	5
1. Scope.....	6
2. Overview of targeted use cases	6
2.1. Radio-astronomy use-case in the context of SKA	7
2.1.1. High-level specifications for data processing	8
2.1.2. High-level specifications for data transport	11
2.1.3. High-level specifications for data emulation	12
2.2. HEP use-case in the context of LHCb	13
2.2.1. High-level specifications for data processing	15
2.2.2. High-level specifications for data transport	17
2.2.3. High-level specifications for data emulation	18
3. Synthesis of specification requirements.....	18
3.1. Outline of high-level specifications for the two use-cases	18
3.2. Hardware technology overview	19
4. Software Composability	21
4.1. Overview of Related Work	21
4.1.1. Overview of (Temporal) Dependency Composition	21
4.1.2. Example of Workflow management framework in Radio-Astronomy	22
4.1.3. Overview of Structural Composition.....	22
4.2. ODISSEE Composability specifications	23
4.2.1. Overview of the specifications.....	23
4.2.2. Examples	25
5. Bibliography	27

List of Tables

Table 1 – Specification requirements derived from the two use-cases high level specifications.	19
Table 2 – Specification requirements derived from the hardware technology overview.	21

List of Figures

Figure 1 The process of generating visibilities (credit N. Monnier, PhD thesis)...	7
Figure 2 Proposed architecture for the SKA SDP from Critical Design Review.....	7
Figure 3 Typical full calibration + imaging pipeline for radio-interferometers. The imaging module within the full pipeline is detailed on the bottom part.	8
Figure 4 Typical dataflows volumes for SKA LOW from science sensors to the telescope site-local SDP sub-systems.	12
Figure 5 LHCb data processing facility overview	14
Figure 6 LHCb real-time processing pipeline overview	15
Figure 7 Current LHCb software articulation	16
Figure 8 Vision for the future LHCb software articulation.....	17
Figure 9 LHCb offline data analysis pipeline overview	17
Figure 10: Example of a component type A with a port p1 that provides the type aType and a connector C with 2 roles r1,r2, that respectively provides and uses the type aType. Components and connector can have attributes.....	23
Figure 11: Example of a simple OCS assembly containing 2 components (a, b) linked by a connection (c).	24
Figure 12: A component can have several implementations, typically for different deployment scenarios.	24
Figure 13: A connector can have several implementations, for different deployment scenarios.....	24
Figure 14: SimpleDataStream connector definition in text and graphics.	25
Figure 15: Simple Producer / multiple Consumers example.	26
Figure 16: HTTP Specialization of the DataStream connector	26
Figure 17: Simple MPI connector.	26
Figure 18: Simple MPI assembly for 32 nodes.....	27

Executive Summary

This document outlines the initial specifications for software composability within the ODISSEE project, aiming to enhance data-intensive solutions for science. It details two primary use cases: radio-astronomy with SKAO and high-energy physics with LHCb, describing their data processing, transport, and emulation needs. The report explores various hardware technologies like CPUs, GPUs, FPGAs, and DPUs that will be leveraged, emphasizing the importance of flexible and scalable workflows. Finally, it introduces the ODISSEE Composability Specifications (OCS), a component model designed to simplify the assembly and interoperability of software components for complex scientific applications.

1.Scope

One of the major innovations proposed in ODISSEE is to provide solutions that enable a trade-off, through composability, between energy efficiency and the accuracy for the processing workflows including data emulation. This involves co-designing hardware and software implementations ranging from highly flexible software-defined solutions to highly efficient low-level hardware implementations. The ability to scale up the actual data volumes to the dimensioning of targeted experiments will be a key ingredient in the trade-off between energy and precision, as well as cost and portability across a variety of cyber-infrastructure. The objective of WP2 is to provide such solutions supporting data emulators in the absence of science sensors, smart data ingestion close to the sensors, efficient data transport within a distributed infrastructure and modular yet scalable workflows as unique tools for both designing future upgrades and preparing scientific exploitation, leveraging innovative HW solutions together with efficient AI tools and trading-off accuracy and energy efficiency.

This is related to ODISSEE's objective #2 recalled below, as from DoA.

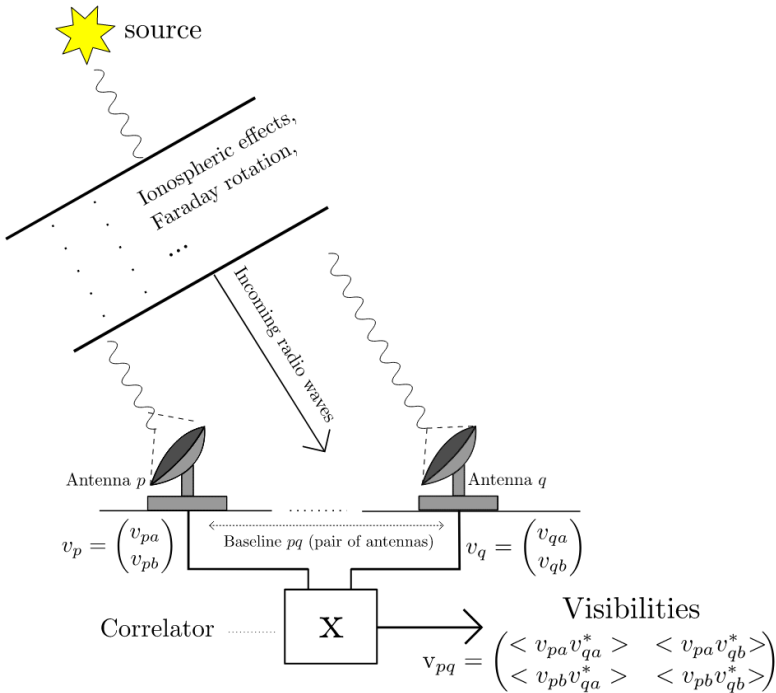
Objective 2: Provide critical components to enhance the science return of HL-LHC and SKAO with reduced system complexity and cost of ownership, through the joint development of composable data interfaces and high accuracy data emulators assessed and validated on SLICES.	
Description: ODISSEE will provide the means to maximize the expected science return of HL-LHC and SKAO while minimizing system design complexity and cost of ownership by delivering composable data interfaces covering the full spectrum of data transport needs from the ingestion of sensors data to the distribution of data products within the cyber-infrastructure. Such unique set of components shall be used to implement data emulators, leveraging AI techniques to enable higher resolution models of the physical world, with the goal to predict performance, prepare and validate their exploitation and design future generations of instruments.	
Achievement Indicators: <ul style="list-style-type: none"> • New data interfaces, based on reprogrammable technologies like FPGA, DPU and NextSilicon Maverick 2 processor realizing data transport at bandwidth as close as possible to 1Tb/s per board • Software-defined composability of these interfaces integrated within the applications development frameworks for SKAO and LHCb. • Realistic data emulators available for the typical ranges of operation of SKAO and LHCb sensors 	Means of Verification: <ul style="list-style-type: none"> • Boards available from partner vendors usable for implementing composable interfaces • A set of software tools to enable software-defined composability • Corpus of data used for emulation / simulation available in a database • Performance reports & publications from users community
Relation to the Work Programme: Enhanced global competitiveness and technological excellence of the EU and Associated Countries in an extremely fast-moving environment through investments into the development of forward-looking technical instruments and tools for European RIs.	

The objective of this document is to provide an overview of the specifications for composability across the infrastructure, including high-performance data interfaces together with configurable and scalable workflows, as well as specific onboard data processing and the type of patterns generated in order to simulate as closely as possible the reality of the data sent by future sensors using an emulator.

2.Overview of targeted use cases

We provide below an overview of the two main use cases in ODISSEE: radio-interferometry with SKAO and collision events reconstruction with LHCb.

2.1. Radio-astronomy use-case in the context of SKA



In a radio-interferometer, the radio waves emitted by the target objects are collected by an array of sensors (antennas or parabolas).

Distributed sensors are synchronised using accurate clocks to ensure each sensor samples the same wavefront. Pairs of signals from different sensors are correlated to form a set of “visibilities”, which constitute a measure of a Fourier component of the object structure in the time-frequency plane.

Figure 1 The process of generating visibilities (credit N. Monnier, PhD thesis)

After a first filter and calibration step, these visibilities are inverse Fourier transformed. This makes it possible to obtain a map of the region of the sky considered, under the form of 4 Stokes components in the image plane, from all this information collected in the time-frequency plane. While the SKA will produce a number of diverse data products, the computational challenge of producing an accurate image from the many visibilities while correcting for instrumental effects (such as signal delays) and computational artifacts (such as filter ripples) drive the requirements for the SKAO Science Data Processor (SDP) systems.

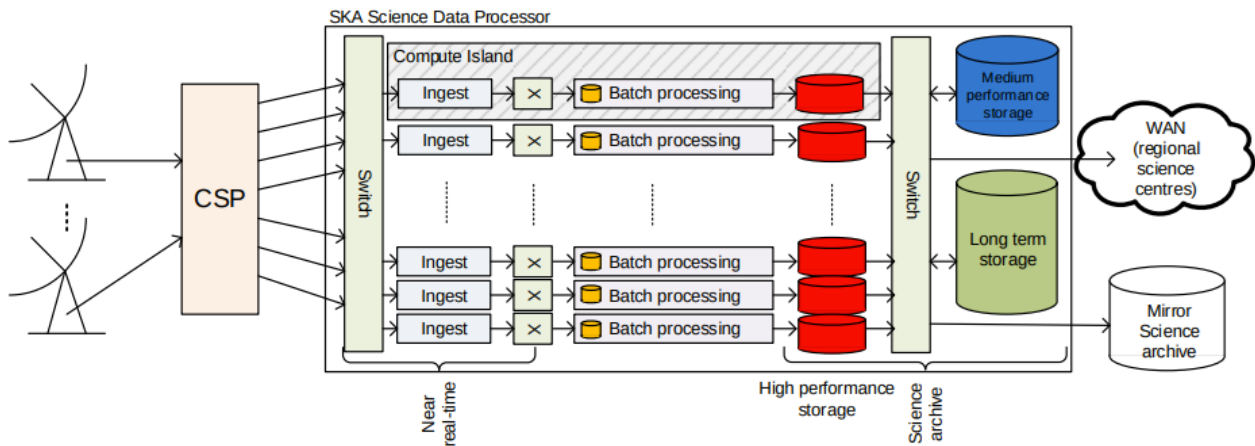


Figure 2 Proposed architecture for the SKA SDP from Critical Design Review

Imaging in radio astronomy is iterative and relies heavily on (fast) Fourier transforms and convolutions on, in the current state-of-the-art, double precision complex floating point data. The signal of interest is buried deep below the noise floor of the sampled signal. This makes accurate and precise computation essential. Furthermore, the calibration algorithms are not guaranteed to converge, due to incompleteness in our human understanding in the chain of physical phenomena, making it difficult to estimate required processing capacities.

One of the main characteristics of the compute systems in major radio telescopes is their requirement to continuously receive large volumes of data for processing. Only very few parts of the processing pipeline in these experiments require data to be at that initial very high resolution. To save storage and processing capacity, data is generally selected and compressed before being stored for additional processing. By moving these functions into the network and applying them on the data as they arrive, we can immediately reduce the resolution of the data for further processing, decreasing the required storage and processing capacities significantly.

2.1.1. High-level specifications for data processing

Considering the system architecture above, and focusing mainly on a typical set of reduction pipeline, an example of a full calibration and imaging pipeline is presented below, providing some kind of worst case scenario in terms of computing capacities dimensioning. These pipelines generate diverse data products, and their operation encloses both real-time and iterative processing. The generated data products span from calibrated visibilities to image data cubes.

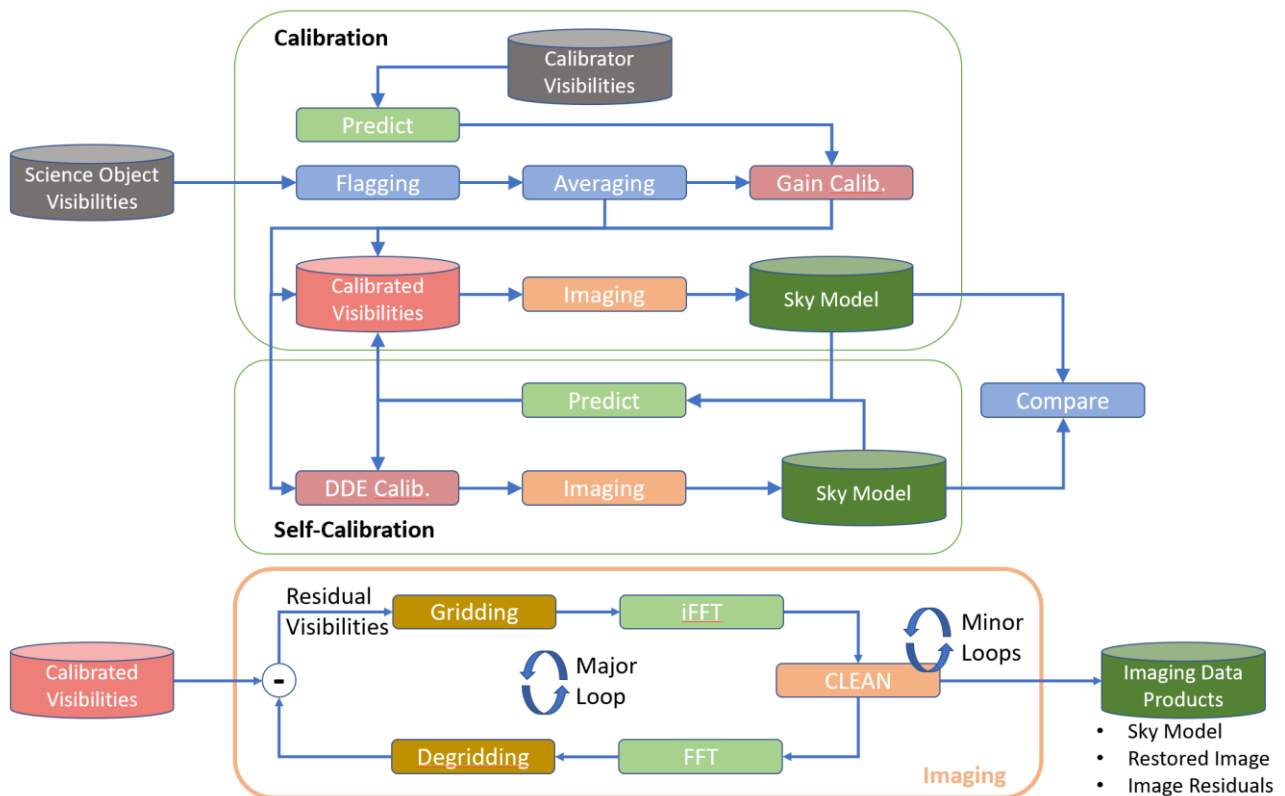


Figure 3 Typical full calibration + imaging pipeline for radio-interferometers. The imaging module within the full pipeline is detailed on the bottom part.

One of the first processing functions done on data arriving in the central processor of a radio telescope is the suppression of interference. Radio Frequency Interference (RFI) is a major limiting factor in modern radio astronomical observations and its removal is of critical importance to the success of any radio telescope. To identify and flag narrow-band, intermittent interference with as much accuracy as possible, this is done on high-resolution data. Moving this part of the processing pipeline into the network, implemented on programmable network controllers with significant compute resources could provide significant benefits. While this is a new approach and could in theory allow immediate reduction of data volumes after mitigation, it is still unclear whether this

approach is sufficient to detect required levels and scales of RFI to accomplish a given science goal.

The ability to detect RFI at an early stage using a variety of established and novel algorithms, together with other parts of the processing pipelines, in particular real-time calibration for aperture array stations¹, may be considered for in-network processing and could result in a significant ability to reduce data volumes immediately afterwards resulting in a potential cost reduction.

The calibration pipeline typically begins with the prediction of high resolution visibilities of a calibrator, a known radio source used to calibrate the data obtained from the radio telescope. This is often referred to as first generation calibration (1GC). The calibrator serves as a reference source for the telescope's sensitivity (through the gains of sensors) and response (through the phase of sensors). Gain calibration is performed by comparing the predicted visibilities of the calibrator to the observed visibilities obtained from the telescope after flagging/averaging and applying specific coefficients.

Gain calibration is the process of ensuring that the amplitude and phase of the signals received from the sky are accurately calibrated to the sensitivity of the telescope. The comparison of the predicted and observed visibilities is carried out iteratively through a process known as self-calibration. During this process, the pipeline updates the sky model used for predicting the visibilities, and the process continues until a stopping criterion is met. The stopping criterion is typically a threshold on the difference between the predicted and observed visibilities or when a certain number of iterations is reached. Once this criterion is met, the self-calibration loop stops², and the calibrated visibilities or data products are produced. Notably, the imaging process does not directly produce images in the calibration pipeline. Rather, the imaging process generates sky models, representing the distribution of radio sources in the sky, that are compared to the observed visibilities to terminate the self-calibration loop. During the self-calibration process, the sky models are iteratively updated. This is often referred to as second generation calibration (2GC).

The imaging step starts with an empty sky model. Firstly, the inversion step generates a dirty image, followed by the detection of one or more bright sources in the image using a deconvolution algorithm such as CLEAN. Then, a model image is created, and corresponding visibilities are predicted. The predicted visibilities are subtracted from the measured and calibrated visibilities, producing residual visibilities. This process eliminates strong sources from the measurements, revealing weaker, more interesting sources. The step is repeated until convergence is achieved, and the sky model is then used to generate the sky image. The inversion and prediction steps involve 2D FFT and gridding or degriding.

Some previous work has suggested that smaller word-sizes may be used for much, but not all, of the processing required, although it is not clear if AI-specific half-precision data structures offer sufficient accuracy for this.

Moreover, there is a direct relation between the resolution of the image and the geographical distance between the two outermost sensors used in an observation. For long baseline imaging, sensors pairs cannot be assumed to be in one plane, due to the curvature of the earth. Similarly, for wide fields of view, the sky cannot be approximated as a flat plane. Together, these effects must be corrected for, and require large

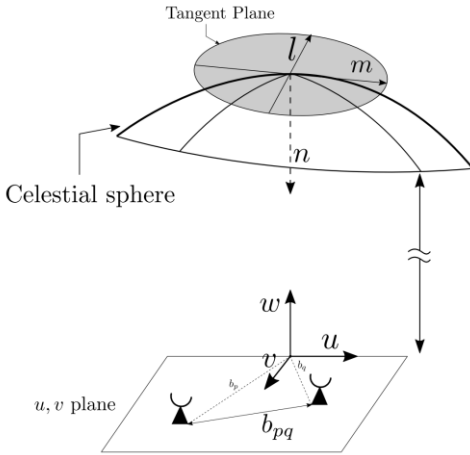
¹ SKA-Low will consist of 131,072 log-periodic dipole antennas distributed across 512 aperture array stations of 256 antennas each

² In de Jong et al. (in prep.) a new model to enable AI to handle this stop/continue criterion leading to more automation and less waste on e.g., processing cycles has been created and will be considered in the project.

convolution kernels during gridding and degriding. Gridding the longest baselines then becomes computationally very expensive.

We provide below, a description of a set of State-of-the-Art (SotA) software for radio-interferometric calibration and imaging.

DDFacet³ is an imager implementing various deconvolution algorithms that take direction-dependent effects (DDE) into account. Specifically, it is a facet-based imager and produces a single tangential plane. It does full polarization DDE correction from any generic time-, frequency-, baseline-, or direction-dependent effect (beam, ionosphere, calibration solution, etc) and such effects are described as being piecewise constant in a facet-based imager. It has the disadvantage of that approximation, but is highly flexible and no physical modeling is needed to describe the DDE. For efficiency, it runs completely in parallel and does on-the-fly Baseline Dependent Averaging (in general $\sim 90\%$ of the data can be compressed). In addition, it takes variable PSFs



into account (DDE, smearing/decorrelation), guaranteeing continuity in the deconvolved sky. This is especially important for the large surveys such as the LOFAR Two-metre Sky Survey (LoTSS⁴) wide/deep. DDFacet has imaged about 10 PByte of compressed visibilities for LoTSS. Which, with its current 3 million sources for 20% completion, is the largest radio survey to date. It has also been successfully used for imaging with VLA, ATCA, WSRT, ASKAP, MeerKAT and GMRT data.

DDFacet has been bundled, together with KillMS⁵ (for direction-dependent calibration) into the ddf-pipeline⁶, an automated data reduction pipeline designed for the LOFAR radio telescope, primarily used for processing wide-field continuum imaging data to a science-ready state with minimal human intervention. The pipeline is essential for large surveys such as the LoTSS and is optimized for high-fidelity, high-dynamic-range imaging across the full field of view.

DP3⁷ (Default Pre-Processing Pipeline, for calibration). Sometimes referred to as DPPP, DP3 is a specialized software package developed primarily for the preprocessing of radio interferometric data, especially for large-scale telescopes like LOFAR (Low-Frequency Array). DP3 can average raw visibility data to reduce data volume while preserving scientific information, which is crucial for handling the massive datasets produced by telescopes like LOFAR and the SKA. The software includes robust tools (e.g., AO-flagger) for identifying and flagging corrupted or unwanted data, such as radio frequency interference (RFI), ensuring cleaner datasets for downstream analysis. DP3 supports various calibration routines, including the correction of both direction-independent and direction-dependent effects (DDEs). This is essential for high-fidelity imaging, as DDEs-such as ionospheric distortions or antenna beam errors-can vary across the field of view and must be corrected on a per-observation or per-facet basis.

³ <https://github.com/saopicc/DDFacet>

⁴ see e.g. <https://arxiv.org/abs/2202.11733>

⁵ <https://github.com/saopicc/killMS>

⁶ <https://github.com/mhardcastle/ddf-pipeline>

⁷ <https://dp3.readthedocs.io/en/latest/>

QuartiCal⁸: a Python-based radio interferometric calibration package designed to address the growing computational demands of modern radio telescopes like MeerKAT and the JVLA. As the successor to CubiCal, it implements fast calibration routines using complex optimization with Wirtinger derivatives, enabling efficient handling of large datasets that scale quadratically with the number of antennas. It supports arbitrary combinations of parameterized gain terms (e.g., G, DE, K, B Jones matrices) for direction-dependent calibration while leveraging Dask for parallel task scheduling and Numba for just-in-time compilation, allowing deployment on systems ranging from laptops to cloud clusters. It uses Dask-MS and XArray for flexible data ingestion, chunking, and grouping by spectral window, field, or scan. QuartiCal excels at 3GC (third-generation calibration) tasks like facet-based calibration and simultaneous direction-dependent calibration, addressing challenges such as RFI mitigation and dynamic range limitations. Its performance has been validated through MeerKAT observations, demonstrating efficient calibration of both continuum and spectral line data.

WSClean⁹: a widely used software package in radio astronomy designed for fast and efficient wide-field imaging of interferometric data. Its name stands for "w-stacking clean," referring to its core imaging algorithm, which is an alternative to the traditional w-projection method. Unlike w-projection, which convolves uv-samples with a w-term correcting kernel before the Fast Fourier Transform (FFT), w-stacking applies the w-term correction as a multiplication after the FFT. This approach generally speeds up the imaging process, especially for large datasets, by making the correction computationally cheaper, though it requires more memory and more FFT operations. As such, WSClean is typically an order of magnitude faster than CASA's w-projection algorithm on data from telescopes like the Murchison Widefield Array (MWA). It can handle very large images (e.g., 10,000 x 10,000 pixels) that may exceed the memory capabilities of other imaging software. The software features advanced deconvolution options, including auto-masked multi-scale cleaning and parallel cleaning, which improve image quality and speed.

Finally, the SKA science processing facilities produce a variety of science-ready data products that are distributed to a network of regional science centres. These generally require further processing into final science products for analysis. Scientists interact with their data on these regional science centres, no scientist access is expected on the site-local SKA science processing facilities (CSP and SDP). Data products range in size from a few to tens of terabytes, making local processing of data on personal laptops infeasible. There is a need to create increasingly higher-resolution images of tens of thousands of pixels squared, consuming hundreds of Gigabytes per image.

2.1.2. High-level specifications for data transport

Considering the system architecture above, and focusing mainly on the SDP workflows, we can derive a set of high-level specifications to be addressed by the data acquisition solution to fit the needs of radio-astronomy. Although the main focus will be the components at the "edge", we will be looking at the continuum (data center) as well.

- Scaled-up bandwidth: in the case of SKA and given the overall requirement of ingesting up to 10 Tb/s per SDP system, the smart data interface (for ingestion and also transport within the infrastructure) should provide typical bandwidth of 100 Gb/s or multiples of that.

⁸ <https://quartical.readthedocs.io/en/latest/>

⁹ <https://wsclean.readthedocs.io/en/latest/>

- Support for direct memory access (DMA) on accelerators: here again, we expect to rely on a cluster of heterogeneous servers coupling host CPUs and accelerators, most of the workload residing on the accelerators. An efficient DMA mechanism, supporting a wide range of possible accelerators is a key requirement.
- In-Network computing features: as stated above, we expect significant gains, in terms of load and cost reduction, by realizing part of the data reduction pipeline "on-the-wire", i.e. as data arrive on the network controller. Extended and flexible in-network computing features are thus to be included in the data interface specifications. This support should include "classical" HPC workflows (e.g. simple arithmetics or linear algebra) together with more advanced concept such as DNN.
- Full integration in programming models and/or execution frameworks for efficient load balancing: in the case of SKA, the complexity of the SDP system together with the need to optimize cost and energy efficiency will require an advanced load balancing strategy across the whole infrastructure. For that purpose, a full integration of the data transport and processing in programming models is a key feature to enable scaled up performance as well as optimized energy consumption when realizing the whole pipeline.

Labate et al.: Highlights of the Square Kilometre Array Low Frequency (SKA-LOW) Telescope

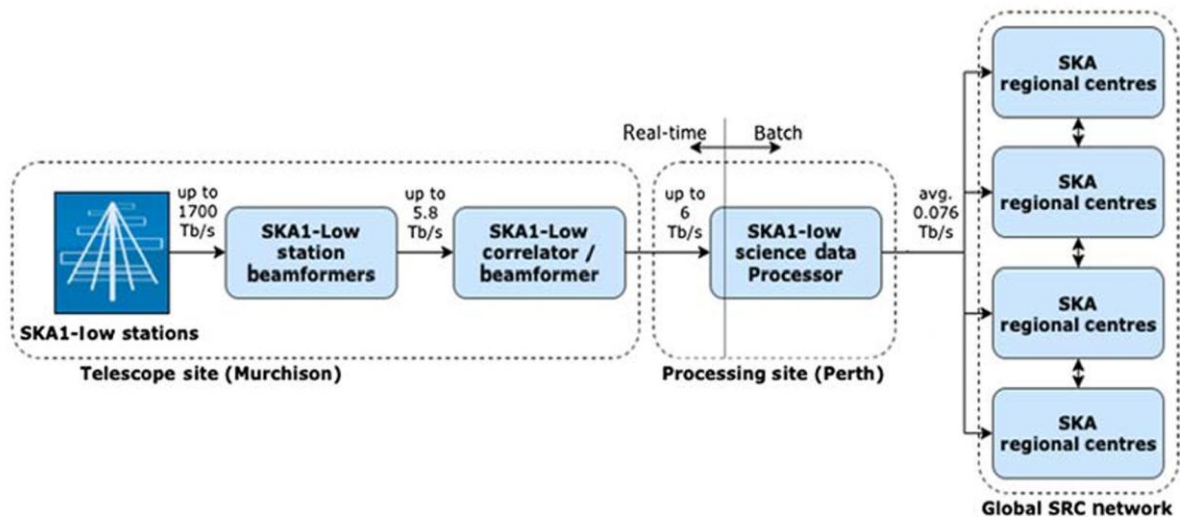


Figure 4 Typical dataflows volumes for SKA LOW from science sensors to the telescope site-local SDP sub-systems.

2.1.3. High-level specifications for data emulation

In order to prepare the community to leverage this unique infrastructure, and maximize the science yield, it is mandatory to develop a realistic data simulator, integrating physical models of scientific targets and their radio emission together with instrumental and operational effects so as to validate the ability to extract actionable measurements from SKA observations. Codebases already exist to simulate efficiently visibilities datasets such as the OSKAR¹⁰ (Oxford SKA Radio Interferometer) simulator, a specialized tool designed for radio interferometry simulations, particularly for the SKA project. It enables precise modeling of radio telescope arrays, sky models, and observational parameters to generate synthetic visibility data. OSKAR's flexibility and

¹⁰ <https://gitlab.com/ska-telescope/sim/oskar>

GPU acceleration make it a cornerstone for SKA-related studies, particularly in beamforming, calibration, and large-scale data processing. Its integration with high-performance workflows ensures relevance for future SKA operations. Its main applications include:

- **Visibility Data Generation:** OSKAR simulates visibility data for radio telescopes, especially those with aperture arrays like SKA-Low
- **Beamforming Simulations:** It enables the investigation of digital beamforming strategies for SKA aperture arrays, including hierarchical beamforming and station-level calibration¹¹
- **Sky and Telescope Modeling:** OSKAR allows users to create and manipulate sky models (source distributions) and telescope models (station layouts, antenna types, and beam patterns)
- **Direction-Dependent Effects Studies:** The simulator is used for investigating various direction-dependent effects on the sky, such as ionospheric distortions³.
- **Imaging and Analysis:** OSKAR includes an imager for creating dirty or residual images directly from simulated data, facilitating quick analysis without the need for external imaging software
- **Performance Testing:** It is used to evaluate the scalability and performance of data processing pipelines, particularly with GPU acceleration for large-scale simulations
- **Algorithm Development:** OSKAR serves as a platform for testing and developing new algorithms for beamforming, calibration, and other radio astronomy techniques

Other works are being led such as ML-PPA¹² within the Intertwin Horizon Europe project¹³ which could be considered in ODISSEE, depending on the level of maturity.

In addition to emulators we will where appropriate make use of real data from relevant SKA precursors and pathfinders.

2.2. HEP use-case in the context of LHCb

The LHCb detector records data at the Large Hadron Collider at CERN. This data is produced when beams of particles collide at the LHCb detector's interaction region, which is located inside the LHCb detector volume around 100 metres below ground. Most of the recorded data is a product of proton-proton collisions, however LHCb also records Pb-Pb data as well as a variety of other processes in which a beam of LHC particles passes through a region enriched with different kinds of gases (for example Helium, Neon, Argon, ...) and interacts with them. From the point of view of the detector data collection and processing of interest to ODISSEE there are only minor differences between these various types of data and therefore we will simply discuss "LHCb data" in what follows. The detector records data in 25 nanosecond windows, called "events", which correspond to one crossing of the LHC beams.

The LHCb detector consists of eight different "subdetectors" which are responsible for measuring different properties of the particles produced in the LHC collisions: their momenta, energies, and information which helps to identify the types of particles (e.g. pion, kaon, muon, ...) present in each event. The data is sent via optical fibres from the front-end, radiation-hard, electronics of each subdetector to "backend" FPGA boards housed in "event-building" x86 servers in a custom above-ground data centre. Some

¹¹ <https://ieeexplore.ieee.org/document/5613289>

¹² <https://gitlab.com/ml-ppa>

¹³ <https://www.intertwin.eu/intertwin-use-case-a-digital-twin-to-simulate-noise-in-radio-astronomy>

subdetectors zero-suppress their data already at the front-end, while others do not. A global clock is distributed by a set of dedicated FPGA control boards and allows each data packet to be associated to the correct event. However when the data arrives at the backend boards, it is time-stamped but not time-ordered. Moreover each backend FPGA board receives only a small fragment of an overall LHCb event, corresponding to the optical links connected to it. The first processing task carried by the event-building servers is therefore to group all data packets corresponding to one event, and then to group order $1e4$ such events together into a “multi-event packet” (MEP) in order to minimize overheads associated with data transport further down the chain.

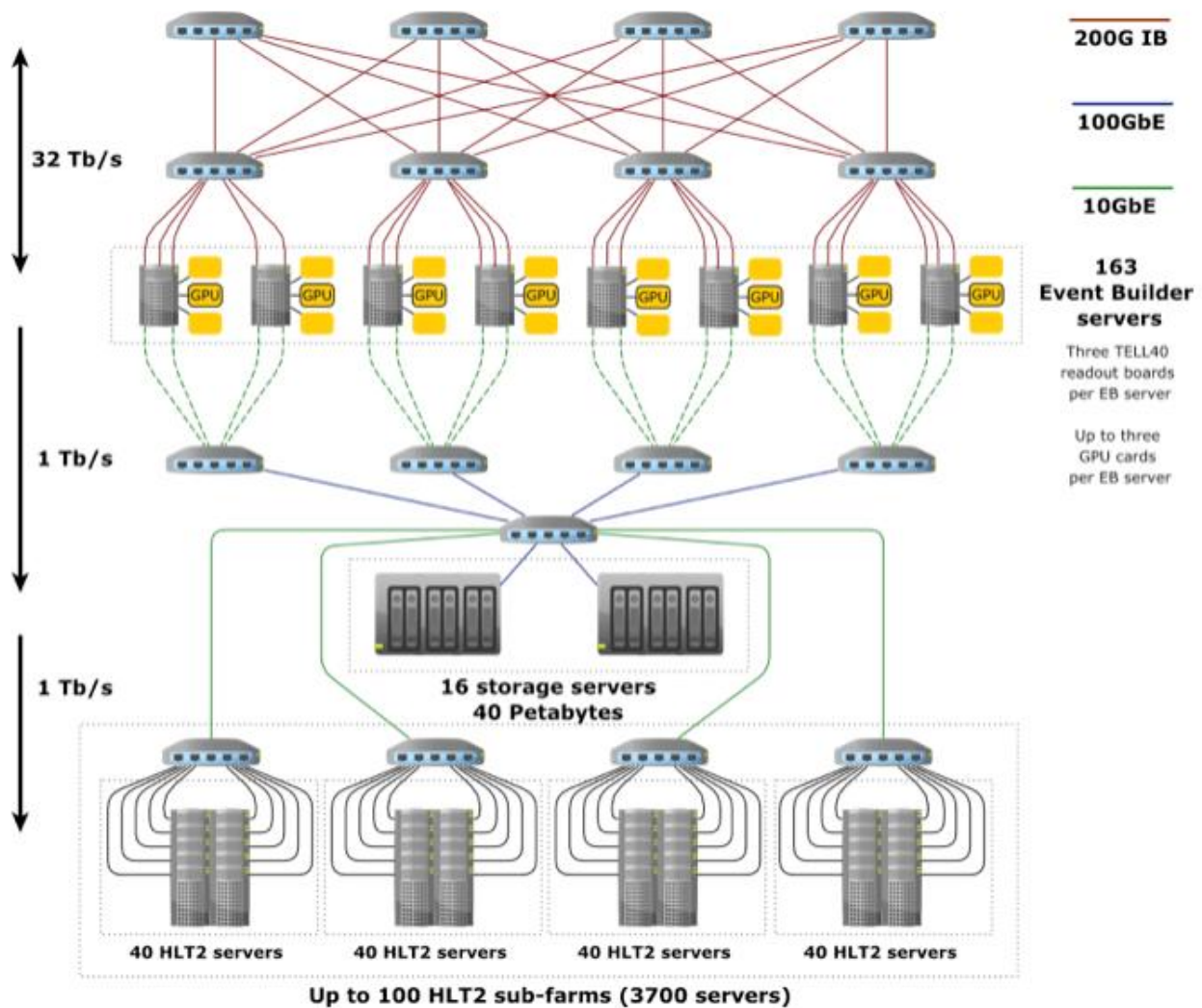


Figure 5 LHCb data processing facility overview

With the MEPs assembled, the overall data rate consists of around 30 million events per second, with each event roughly 70 kiloBytes in size, or around two Terabytes¹⁴ of aggregated physics data per second. This data volume needs to be reduced to 10 Gigabytes per second which can be written to permanent storage for physics analysis, a reduction of around two and a half orders of magnitude. Because the FPGA cards are hosted in x86 servers whose memory (512 Gigabytes per server) serves as a sufficient

¹⁴ The system was designed for a rate of 5 Terabytes/second maximum, including generous margins since the data volumes recorded by a collider detector depends on the beam and material-related backgrounds and cannot always be reliably predicted upfront.

operational buffer, latency is not a relevant system parameter and instead event throughput controls whether the system can take data without losing packets or not. In practice packet loss is limited to <1%.

2.2.1. High-level specifications for data processing

In traditional collider experiments most beam crossings (events) contain nothing of interest, and the real-time processing serves in order to reject those based on general topological criteria which efficiently retain a wide range of the (rare) signals of interest. Since only one in a thousand or even ten thousand events is kept, the full “raw” detector data can be kept for this event and there is no inherent need to perform a physics-analysis-quality reconstruction of this data in real-time. This strategy is not applicable to LHCb because at the data rates collected since 2024, more than 5% of the events¹⁵ contain topologies of interest for at least one physics analysis. LHCb is therefore obliged to perform a full physics-analysis-quality reconstruction of the detector data in quasi-real-time, and record to disk high-level physics features of interest to analysis for a large number of events while throwing away the underlying detector data for most events. This also implies performing a physics-analysis-quality alignment and calibration of the detector in real-time. The figure below provides an overview of the various stages of data processing within the LHCb real-time pipeline.

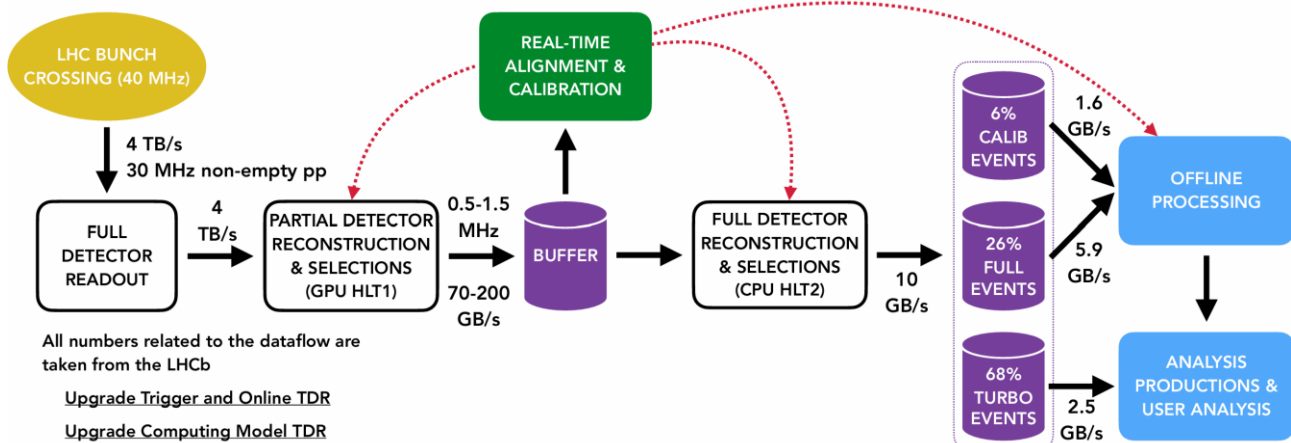


Figure 6 LHCb real-time processing pipeline overview

The typical features of the 3 main building blocks are listed below.

High Level Trigger 1 (HLT1):

- 30 MHz input
- Not latency bound
- O(100) algorithms to maintain
- O(10) developers
- 125k lines of code
- Processed by Allen software¹⁶ on O(500) GPUs

High Level Trigger 2 (HLT2):

- 1 MHz input
- Not latency bound
- O(2000) algorithms to maintain
- O(100) developers
- 2M lines of code

¹⁵ <https://inspirehep.net/literature/1920705>

¹⁶ <https://allen-doc.docs.cern.ch/>

- Processed by Moore software¹⁷ on CPUs

Alignment & Calibration:

- Determine detector position based on reconstructed quantities
- RICH mirror alignment & refractive index calibration
- Calibrate calorimeter

As of the time of writing this document, the real-time data analysis pipeline for LHCb described in Figure 7 is supported by a software bundle composed of two main frameworks Allen and Gaudi articulated according to the picture below.

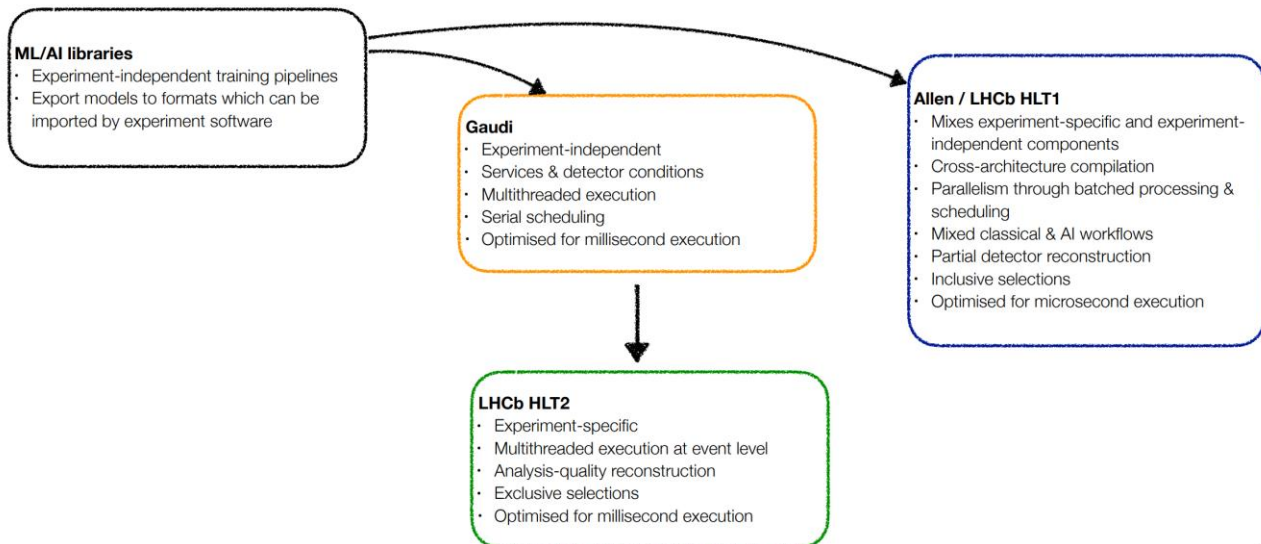


Figure 7 Current LHCb software articulation

One of the key challenges faced by the LHCb community today is to make the 3 main components above (Allen, Gaudi and ML/AI libraries) fully interoperable and most importantly composable to realize modular pipelines that can be mapped efficiently on the heterogeneous architecture depicted in Figure 6 (offline processing excluded). In particular, a few key developments are required that will benefit from the implementation of a composability framework including: enabling Gaudi services in Allen, extending Gaudi's monitoring infrastructure to Allen or the ability to explore the use of emerging architectures such as ARM processors or new technologies such as Maverick from NextSilicon. On top of this, the connection between Allen/RTA and AI/ML is becoming a strategic development. This can be outlined with the figure below.

¹⁷ <https://gitlab.cern.ch/lhcb/Moore>

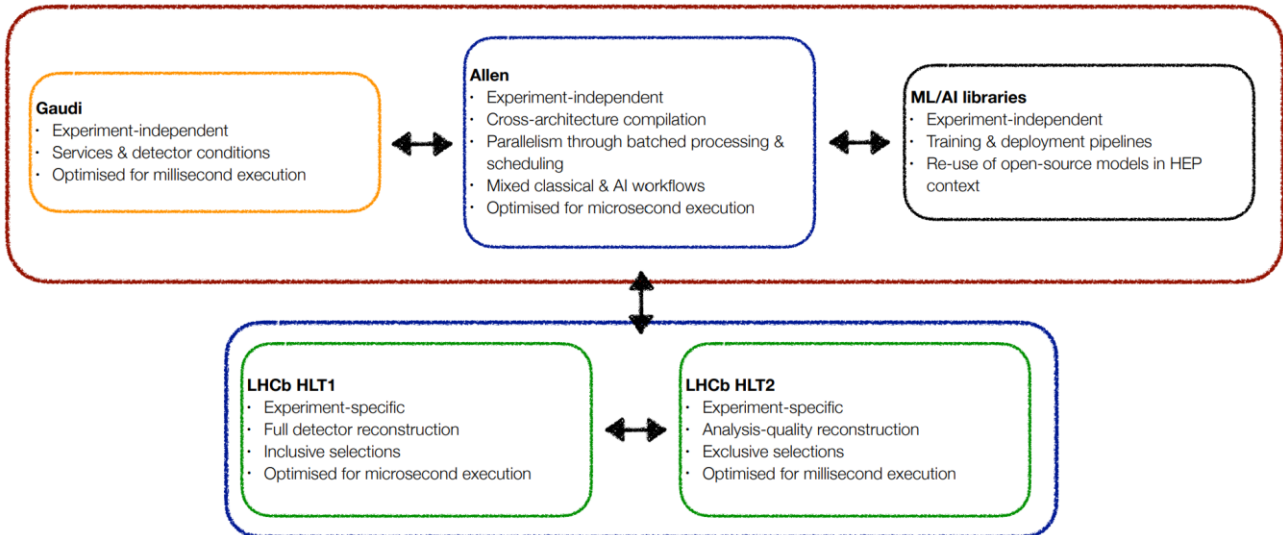


Figure 8 Vision for the future LHCb software articulation

For a complete description of the data processing steps, up to results publication (although not considered in this WP), the typical LHCb offline data analysis pipeline, mostly executed on the WLCG (Worldwide LHC Computing Grid¹⁸) is represented in the figure below.

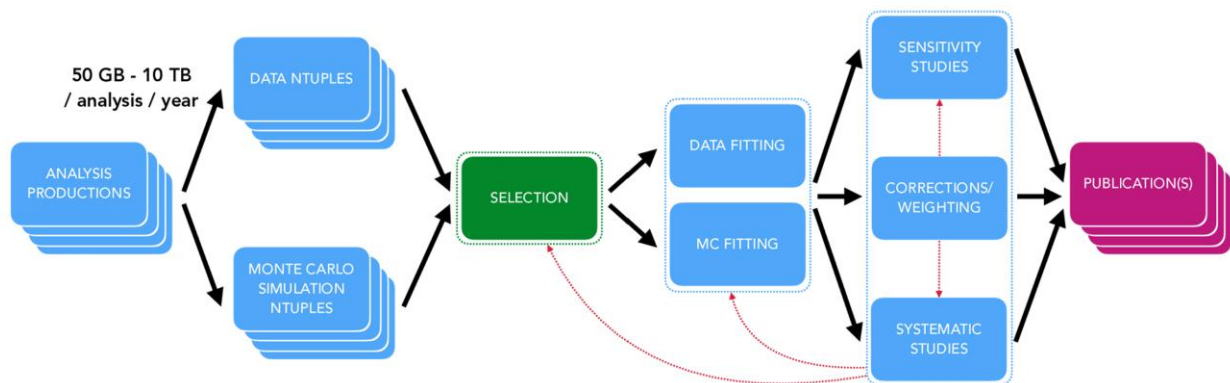


Figure 9 LHCb offline data analysis pipeline overview

2.2.2. High-level specifications for data transport

Once the events have been built, the LHCb system is in principle fully flexible in terms of architecture, and is optimised for the cost of the processing hardware, not including power but including components used to move data around, buffer data while being processed, and so on. The energy efficiency of LHCb's data processing¹⁹ is an important consideration but until now has been secondary in terms of operational costs to other parts of the overall instrument. There is however no in-principle reason why future LHCb upgrades cannot optimise for total cost of ownership, including energy costs, or even directly for energy efficiency. In practice the current LHCb detector is processed using a two-stage system. The first stage, implemented on GPU processors, ingests MEPs and reduces the data volume by roughly a factor 20 using the kinds of inclusive topological selections discussed earlier. This data is subsequently recorded to a 30 Petabyte disk buffer while the detector is aligned and calibrated with physics-analysis-quality. The

¹⁸ <https://home.cern/science/computing/grid>

¹⁹ <https://inspirehep.net/literature/1868467>

data transport between the GPUs and disk servers is primarily limited by the total number of disk servers and the single-disk read/write speed, since SSDs are unaffordable. A second stage, implemented on x86 servers, then reduces the data volume by a further order of magnitude before it is written to permanent storage, available for being distributed to WLCG for physics analysis.

2.2.3. High-level specifications for data emulation

All algorithms used in LHCb's real-time processing are developed and tested using a detailed simulation of the detector. This simulation is based on the Pythia event generator which emulates the underlying LHC collisions, GEANT4 for transporting particles through the detector material, and custom LHCb-specific software for emulating the digitisation of particles trajectories into hits in the detectors based on detailed detector geometry maps defined using the DD4HEP package. ODISSEE will be provided with a sufficient quantity of simulated events of various kinds in order to develop and benchmark the solutions discussed in this document. A set of dedicated servers, with and without GPU cards, are used to reproduce as closely as possible the real production environment and make these tests as realistic as possible.

3. Synthesis of specification requirements

3.1. Outline of high-level specifications for the two use-cases

Work led in WP1 will define and specify new approaches for radio-astronomical visibilities generation, calibration and image synthesis as well as LHCb real-time data processing and prepare validation methodologies in comparison with SotA approaches on a corpus of simulated and real data. These new components shall be used to implement data processing solutions, leveraging new AI techniques to enable higher resolution models of the physical world, with the goal to predict performance, prepare and validate their exploitation and design future science programs. Although, mainly focused on the workflows running at the edge infrastructures of radio astronomy and LHCb, many of the approaches developed in ODISSEE are also relevant to workflows running at the data centers in the continuum. Hence, investigating the developed approaches in relation to the continuum is within scope of ODISSEE. The high-level specification requirements from both use cases can be outlined as follows.

Name	Description
Multiple codebases support	The proposed solutions should at high level be able to support the multiple codebases mentioned above for both use cases.
Codebases interoperability support	The proposed solutions should support interoperability between multiple components from different codebases for each use case (independently)
Support for data simulators integration	The proposed solutions should support the integration of raw data simulators at the input of the pipeline, in place of the raw data ingestion interface.
Support for sequential as well as pipelined execution	The proposed solution should support both pipelined and sequential execution models to cope with constraints from both use cases

Support multiple compute hardware technologies	The proposed solutions should where appropriate support mainstream technologies such as x86 CPUs, GPUs and possibly FPGAs, as well as emerging technologies, in particular those promoted by the project, namely SiPearl Rhea processor (based on ARM) and NextSilicon Maverick.
Support multiple networking technologies	The proposed solutions should support mainstream networking technologies (COTS network interconnect and associated protocols based on Ethernet or Infiniband) as well as emerging technologies such as converged DPUs (network controllers and GPUs on the same board) as well as custom FPGA solutions (e.g. in the context of the European Exascale Interconnect project : NET4EXA ²⁰)

Table 1 – Specification requirements derived from the two use-cases high level specifications.

3.2. Hardware technology overview

The goal in ODISSEE is to demonstrate applications support on multiple hardware architectures (e.g., CPU, GPU, FPGA), assess their efficiency in the context of the two main use cases described above and assemble proofs of concepts enabling portable, optimized and efficient implementations of the corresponding core workflows capable of exploiting the full potential of the European HPC ecosystem. Composability plays a critical role here, and as such, one high level specification for the proposed composability solution(s) is to support the following list of hardware technologies.

It has to be noted that considerations on supporting specific hardware technologies in the following categories (e.g. for DPUs, which in some cases are still emerging and for which long term availability is not predictable today) will be limited to solutions already available within SLICES-RI (i.e., partners hosting experimental facilities such as eX3 at Simula or Grid'5000 in France). The actual implementation of this support will be limited to solutions selected for the hardware demonstrators in WP3 of ODISSEE, still under definition at the time of writing this document (see deliverable D3.1, in preparation).

Name	Description
Networking technologies	
NIC	Typical high bandwidth Network Interface Controllers with several ports supporting multiples of 100Gb/s and protocols such as Ethernet and Infiniband. These solutions can typically support advanced features for data transport (e.g. DMA) but won't support advanced in-network computing solutions (e.g. compression). This technology should be natively supported by the composability solution in ODISSEE.
DPU	Data Processing Units (DPUs) are a new generation of reprogrammable high-performance processors linked to high-performance network interfaces. DPUs can embark advanced features for both data transport and processing and represent the SotA for in-network computing solutions. Multiple vendors offer solutions with various

²⁰ <https://net4exa.eu/>

	levels of maturity including Nvidia/Mellanox ²¹ , Intel ²² or AMD ²³ among the mainstream players. One specification for composability in ODISSEE is to provide support for at least one of these solutions for data ingestion / transport.
Custom FPGA	Field-Programmable Gate Arrays (FPGAs) are built from a grid of configurable logic blocks (such as Logic Elements or Adaptive Logic Modules), programmable switches, embedded memory (RAM blocks), and specialized processing blocks (e.g., DSPs). They can process data with minimal delay, making them well-suited for real-time and latency-sensitive applications, such as network packet processing. In the context of networking applications, these solutions are totally flexible. They can support advanced data transport features (e.g. multiple protocols handling, complex data routing, DMA, etc ..) as well as advanced in-network computing features (e.g. compression, filtering, DNN inference, etc ..) at the cost of complex development cycles. One specification for composability in ODISSEE is to provide support for data ingestion / transport with custom FPGA interfaces, when existing logic and networking blocks are available.
Computing technologies	
CPU	Beyond standard support for high-end x86 CPUs from AMD and Intel, a key specification in ODISSEE is to ensure support for the Rhea processor developed by SiPearl. Rhea incorporates HBM2E memory with a large number of high performance Arm Neoverse V cores for a processor targeting balanced operation for supporting memory bound codes.
GPU	Graphical Processing Units (GPU) are specialized processors designed for highly parallel tasks and consisting of thousands of cores, enabling them to process many operations simultaneously. The two main vendors at the time of writing this document are Nvidia (90% market share on high-end GPUs) and AMD. Beyond some architectural differences, the key discrepancies lie in the software ecosystem, with the ROCm platform for AMD being less mature than Nvidia's CUDA. Since both use cases will leverage this technology at some level, one key specification for composability is to enable support (at least partial) for both ecosystems.
NextSilicon Maverick	Maverick-2 Intelligent Compute Architecture (ICA) is a novel software-defined approach that uses real-time optimization to learn your application and accelerate just the hot spots. On top of this, data can be streamed via the experimental on-die 100Gb Ethernet NIC directly to the

²¹ <https://www.nvidia.com/en-us/networking/products/data-processing-unit/>

²² <https://www.intel.com/content/www/us/en/products/details/network-io/ipu.html>

²³ <https://www.amd.com/en/products/data-processing-units/pensando.html>

	reconfigurable compute grid to provide more efficiency. a key specification for composability in ODISSEE is to ensure support for Maverick.
FPGA	FPGAs can also support a high level of parallelism as well as custom hardware acceleration through reprogrammability and can consume less power than CPUs and GPUs for parallelizable tasks. However, their adoption is tempered by programming complexity and integration challenges. One specification for composability in ODISSEE is to provide support for FPGA acceleration on some well-defined specific tasks, when existing logic and processing blocks are available. The goal is not to support full pipelines execution on FPGAs available to the project.

Table 2 – Specification requirements derived from the hardware technology overview.

4. Software Composability

4.1. Overview of Related Work

Software composability [NATO68] can be defined as the property of being able to compose code (software) to improve code reuse, allowing software systems to be assembled from modular, interchangeable parts. It addresses the challenges of improving separation of concerns and code reuse, both critical aspects for managing the complexity of hardware and software environments.

For this document, we can distinguish two major trends in software composability. A first branch of work is based on the idea of (temporal) dependencies. It contains in particular workflow and dataflow models. A second branch is more focus on the structural composition and it contains in particular the work done in the context of component-based software engineering (CBSE). An orthogonal trend, based on algorithmic skeleton [COL89], exists but it is not considered in this document as it is less relevant at this stage of the project.

4.1.1. Overview of (Temporal) Dependency Composition

A first category of previous work usually refers to data flows, control flow/workflow, and pipeline to express dependency composition. Dataflow and workflow models are two classes of models to describe (temporal) dependencies between computing tasks. Data flow models differ from workflow models as data flow only expresses data dependencies between tasks while control flow adds elements to control the execution flow. Typically, in a data flow model, a task is fireable²⁴ as soon as all its input data are available. Once the task has finished, all its output data are set and thus dependent tasks can be considered to be fired.

Workflow models typically orchestrate the flow of execution by specifying the order of execution of tasks, and they usually contain control flow elements such as conditional branching or loops. Hence, the dependencies between tasks are at the burden of the programmers. Some workflow models combine control flow and data flow. This is

²⁴ A fireable task is a task that has all its dependencies satisfied and it is thus to be considered to be executed when the scheduler decides it, typically when enough resources to execute it are available.

particularly interesting to increase the level of concurrency in case of parallel execution and to optimize resource management.

There are many models of data/workflow. One of the difference is the execution environment such as parallel or distribution environment. For example, COMPS [COMPS] is a task-based programming model which aims to ease the development of applications for distributed infrastructures, such as large High-Performance clusters (HPC), clouds and container managed clusters.

Note that there is a particular case of data/workflow model when the inputs/outputs data are streams and not just a set of data. In that case, the literature references it as pipelines or stream computing depending of the domain of consideration.

4.1.2. Example of Workflow management framework in Radio-Astronomy

An example of workflow management framework in radio-astronomy is Stimela²⁵, a containerized framework designed to facilitate the creation, management, and execution of data reduction workflows for radio astronomy, particularly in radio interferometry. It is not a fixed pipeline but rather a flexible environment that enables users to construct reproducible and portable pipelines by integrating diverse radio astronomy software tools. It does not impose a specific workflow but provides the tools and standards to create, manage, and execute pipelines for radio astronomy data processing. Workflows are described using linear, concise, and human-readable YAML-format "recipes". These recipes define the sequence of data reduction tasks and their dependencies in an intuitive way. Atomic data reduction tasks (e.g., binary executables, Python functions, CASA tasks) are encapsulated in "cab definitions". Each cab definition, also in YAML, specifies the schema for the task, detailing its inputs and outputs.

Another early example of the use of workflow manager in radio astronomy is still in active use in the LOFAR telescope. Software components in LOFAR are combined into pipelines using Common Workflow Language (CWL) and distributed over nodes using the TOIL workflow manager. A similar approach is currently being pursued by the SKAO software development programme. However, while this programme has not made a formal decision on the use of work distribution framework, it is currently investing in the Dask framework, which is much more modern and better supported than CWL/TOIL. To avoid duplication of effort and divergence of software, the International LOFAR telescope is considering moving to Dask as well.

4.1.3. Overview of Structural Composition

According to [HT01, SZY02], component-based software engineering (CBSE) is a style of software engineering that aims to construct a software system from components that are loosely-coupled and reusable. This emphasizes the separation of concerns among components. It led to many works these last decades. For example, in sequential and distributed systems, some well known systems include (Distributed) Component Object Model (DCOM), or the CORBA Component Model (CCM).

In the context of HPC, there have also several works such as the Common Component Architecture [CCA], Low Level Component Model [L2C], and more recently COMET [OBCOR17, COMET]. CCA targeted in particular interoperability between components developed by different teams across different institutions though the use of a dedicated language to describe interface between component — Scientific Interface Definition

²⁵ see <https://www.sciencedirect.com/science/article/pii/S2213133725000320?via%3Dihub>

Language or SIDL. It supported FORTRAN, complex numbers, as well as multi-dimensional data arrays and SPMD-like execution models. L2C is a lower component model that does not add overhead between components as it does not support language interoperability. Supported connectors between components are C/C++ function/method invocation, CORBA remote method invocation, and also MPI communications. COMET is an extension of L2C that proposes partitioned data flow composition and that generates the OpenMP code that submits the associated task graphs.

Some models, like CCM, support several implementations for a particular component definition as a mean to support hardware/operating system heterogeneity but also to provide implementations with various performance trade-offs.

Some models enable to have an assembly as an implementation of a component, such as Fractal [FRA09]. Such components are usually called composite components. HLCCM [HLCCM10] increases the level of flexibility by also enabling composite connectors to connect components but also parametric components and connectors. However, the model does not provide any hint to simplify the computation of a concrete assembly (i.e., an assembly with only primitive components and connectors).

4.2. ODISSEE Composability specifications

4.2.1. Overview of the specifications

The ODISSEE Composability Specifications (or OCS) aim to address the targeted use cases by proposing a model as simple as possible. These specifications will be revisited if necessary during the lifetime of the ODISSEE project. There are inspired from HLCCM [HLCCM10].

Let define OCS as a component model made of computing elements, named components, and relationships between those components, named connectors as illustrated in Figure 10. A component or a connector is an abstract definition, meaning that its implementation(s) will be provided later through a specialization mechanism. We will use the terms component types and component instances to distinguish a type from an instance while we will use connector for type and connections for instances of a connector. This document uses the term component (without type or instance) when the context enables to disambiguate between the two cases..

Component A:

Port[provides] aType p1;

Attributes:

aType anAttribute;

Connector C;

Role[provides] aType r1;

Role[uses] aType r2;

Attributes:

aType anAttribute;

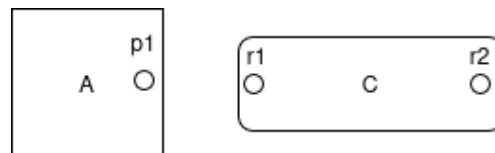


Figure 10: Example of a component type A with a port p1 that provides the type aType and a connector C with 2 roles r1,r2, that respectively provides and uses the type aType. Components and connector can have attributes.

A component (type) is in particular defined by a list of named ports and a connector is in particular defined by a list of named roles. In Figure 10, the component (type) A has

a port named p1 and the connector C has two roles r1 and r2. A port or a role is associated to a type as well as with a kind. In the initial version of OCS, two kinds are defined: *provides* and *uses*. Attributes (e.g., a set of key/value) can be added to components (types and instances) and to connectors and connections.

An assembly is a set of component instances and connections as illustrated in Figure 11.

Assembly:

Instances:

A a;

B b;

C c;

Connections:

a.p1 — c.r1; // the order is irrelevant

c.r2 — b.p2;

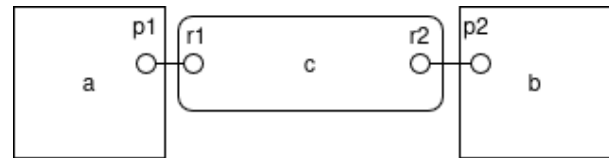


Figure 11: Example of a simple OCS assembly containing 2 components (a, b) linked by a connection (c).

As illustrated in Figure 12, a component (or a connector) can have one or several implementations, each implementation provides its own set of metadata (defined through attributes) that will be later used to actually select the right implementation in function of the execution environment. The implementation of a component can be either an executable form of the component, e.g. a binary, an executable shared library or an assembly, i.e., a set of components and connections. In this latter case, the component or the connector is said to be composite. The implementation of a connector is either a primitive connector, i.e., provided by some external mechanism such as a TCP connection or an MPI library, or an assembly.

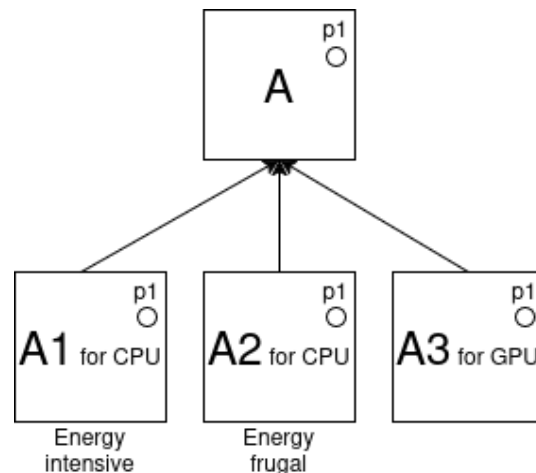


Figure 12: A component can have several implementations, typically for different deployment scenarios.

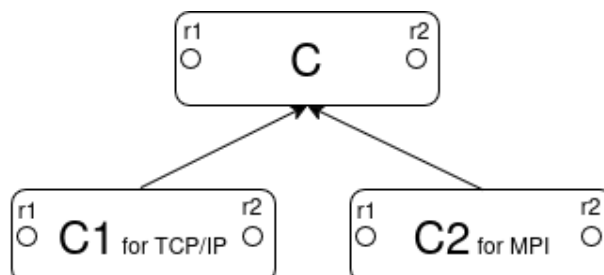


Figure 13: A connector can have several implementations, for different deployment scenarios

A connector aims to abstract the communication medium between components. The classical Sender/Receiver connector is similar to the connector *c* of Figure 10 with the two roles *sender* and *receiver* whereas an MPI connector will typically contains only one role to gather all participants to a MPI communicator. Attributes can be associated to further capture the nature of the communication, e.g. a single piece of data or a stream of data.

Roles have a cardinality, by default set to 1, as well as constraints to define the ports that can be connected to it. For example, a role of a connector with a type *T* could be only associated with a port of a component compatible with type *T*. In the case of an MPI connection, the cardinality is typically the number of process participating to the MPI communicator (defined as "+" in OCS if not limited).

4.2.2. Examples

This sections contains a set of examples to illustrate OCS.

Data Stream Connector

Figure 14 represents a connector that transports a (generic) data type *T* from one source to one or several sinks. An attribute *isStream* is added to indicate whether it is just one piece of data or a stream of data to proceed.

Connector *DataStream*<*T*>:

Role[provides] <*T*> source:

// exactly one source required

Cardinality : 1

Role[uses] <*T*> sink:

// at least one sink, multiple sinks are allowed

Cardinality: +

Attributes:

Boolean *isStream*; // false: just one data, true: multiple data to transport



Figure 14: *SimpleDataStream* connector definition in text and graphics.

Simple Stream Producer/Consumer Example

Figure 15 describes a simple (abstract) assembly made of three components and an instance of the *DataStream* connector presented previously.

Component Producer:

Port[provides] *aDataType* portP;

Component Consumer:

Port[uses] *aDataType* portC;

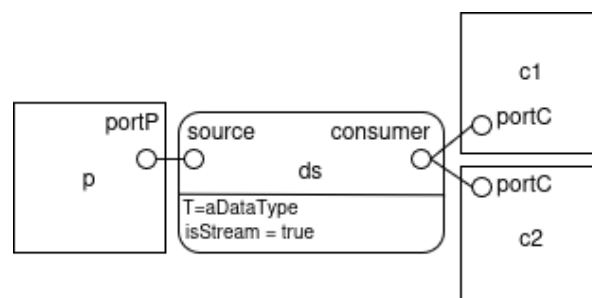
Assembly exampleAssembly:

Instances:

Producer *p*;

Consumer *c1*, *c2*;

SimpleDataStream *ds*;



Connections:

p.portP — ds.source<aDataType>; // the value of the generic type T
 c1.portC — ds.sink<aDataType>; // of ds could be automatically derived
 c2.portC — ds.sink<aDataType>; // from the types of connected ports.

Figure 15: Simple Producer / multiple Consumers example.

Specialization of the Simple Data Stream Connector

Figure 16 presents a possible specialization of the DataStream connector to an HTTP transport layer.

Generator DataStreamHTTP<T> **specializes** DataStream<T>:

Instances:

HTTPsender s;
 HTTPReceiver r;
 HTTPtransport http;

Connections:

this.source — s.sink; // this represents the type being specialized.
 s.source — http.sink;
 http.source — r.sink;
 r.source — this.sink;

Attributes:

Boolean enable SSL; // false for plain http and true for http/SSL connections (https)

Figure 16: HTTP Specialization of the DataStream connector

MPI Connector

Figure 17 displays a basic MPI connector for one communicator. As all the participants to the communicators are similar, a single role (with a multiple) cardinality is enough. An implementation can typically returns the MPI communicator to the participants that they can used in the MPI calls.

Connector MPI

Role[provides] member:
 // one or several participants
Cardinality: +

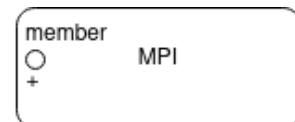


Figure 17: Simple MPI connector.

MPI Assembly

Figure 18 describes an assembly made of 32 components and an instance of the MPI connector presented previously.

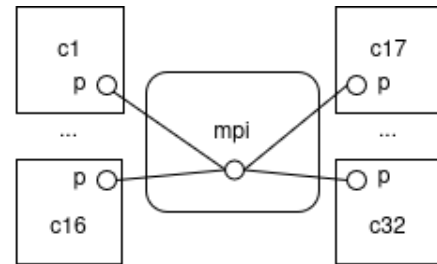
Component MPISolver:**Port**[provides] MPI p;**Assembly** exampleMPIAssembly:**Instances:**

MPISolver c[32];

MPIConnector mpi;

Connections:

c[1..32].p — mpi.member; // connect port p of the 32 c instances
 // to the member role of an MPI connection.

*Figure 18: Simple MPI assembly for 32 nodes*

5. Bibliography

- [NATO68] Mcllroy, M.D. Mass Produced Software Components. Proceedings of NATO Software Engineering Conference, Garmisch, Germany, October 1968, 138-155.
- [COL89] Murray Cole, Algorithmic Skeletons: Structured Management of Parallel Computation, Mit Pr, 1 January 1989, ISBN 0262530864.
- [HT01] George T. Heineman, William T. Councill (2001). Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Professional, Reading 2001 ISBN 0-201-70485-4
- [ZSY02] Clemens Szyperski, Dominik Gruntz, Stephan Murer (2002). Component Software: Beyond Object-Oriented Programming. 2nd ed. ACM Press - Pearson Educational, London 2002 ISBN 0-201-74572-0
- [FRA09] Blair, G., Coupaye, T. & Stefani, JB. Component-based architecture: the Fractal initiative. Ann. Telecommun. 64, 1–4 (2009). <https://doi.org/10.1007/s12243-009-0086-1>
- [COMPS] Lordan, F., E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia, ServiceSs: an interoperable programming framework for the Cloud, Journal of Grid Computing, March 2014, Volume 12, Issue 1, pp 67–91,, DOI: 10.1007/s10723-013-9272-5
- [CCA] Bernholdt D.E., Allan B.A., Armstrong R., Bertrand F., Chiu K., Dahlgren T.L., Damevski K., Ewasif W.R., Epperly T.G.W, Govindaraju M., Katz D.S., Kohl J.A., Krishnan M., Kumfert G., Larson J.W., Lefantzi S., Lewis M.J., Malony A.D., McInnes L.C., Nieplocha J., Norris B., Parker S.G., J. Shende Ray, T.L. S. Windus, and S Zhou. A Component Architecture for High Performance Scientific Computing. International Journal of High Performance Computing Applications, May 2006.
- [L2C] Julien Bigot, Zhengxiong Hou, Christian Pérez, Vincent Pichon. –A Low Level Component Model Easing Performance Portability of HPC Applications. – Computing, Vol. 96, Issue 12, page 1115-1130, December 2014 (Date: 23 Nov 2013), Springer Vienna, ISSN 0010-485X, DOI 10.1007/s00607-013-0368-3
- [COMET] Jérôme Richard. Conception d'un modèle de composants logiciels avec ordonnancement de tâches pour les architectures parallèles multi-coeurs, application au code Gysela. PhD, Université de Lyon, 2017. In French.

[OBCOR17] Olivier Aumage, Julien Bigot, Hélène Coullon, Christian Pérez, Jérôme Richard. Combining Both a Component Model and a Task-based Model for HPC Applications: a Feasibility Study on GYSELA. 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)., May 2017, Madrid, Spain. <10.1109/CCGRID.2017.88>. <hal-01518730>

[HLCM10] Julien Bigot. Du support générique d'opérateurs de composition dans les modèles de composants logiciels, application au calcul à haute performance.. Modélisation et simulation. PhD. INSA de Rennes, 2010. In French.