D2.1 Workload characterisation and Middleware Requirements

Requirements Document

Work package:	**WP2 Middleware Requirements from Workflows**	
Author(s):	Domokos Sarmany, Simon Smart, Teodor Nikolov, Julien Capul, Sebastien Morais, Francois Tessier	
Reviewer #1	Dirk Pleiter	JUELICH
Reviewer #2	Utz-Uwe Haus	CRAY
Dissemination Level	PU	
Nature	R	

DATA ORCHESTRATION IN HIGH PERFORMANCE COMPUTING
This project has received funding from the European Union's Horizon 2020 research and Innovation program through grant agreement 801101.

Date	Author	Comments	Version	Status
01/04/2019	Domokos Sarmany	Initial version	0.1	
29/04/2019	Domokos Sarmany Simon Smart	Updated partner requirements	0,2	
30/04/2019	Domokos Sarmany Simon Smart	Proof reading changes	0.3	
01/05/2019	Simon Smart	For internal review	1.0	
28/05/2019	Simon Smart	Updates from review	1.1	
29/05/2019	Domokos Sarmany Simon Smart	Updates from partners	1.2	

Contents

Executive Summary

The overall aim of the Maestro project is to design and build a novel memory- and data-aware middleware which will enable application authors to take advantage of a range of data-related workflow optimisations and a heterogenous mix of available hardware, both in the present and in the future.

In order for this novel middleware to demonstrate improvements in the application workflows, the project has been structured around a co-design approach between application authors and the authors of the middleware. This process has been carried out as Task 2.1 of Work Package 2 and has involved extensive collaboration between the partners involved in work packages 2, 3, 4 and 5 over a period of months. This has taken the form of an application requirements gathering exercise, followed by a number of consultation and design iterations.

This deliverable gives the output of the requirements capture process and provides an overview of the application-oriented input given to the co-design process. This takes the form of an overview of the workloads being run by the various project partners, a breakdown of these into their constituent activities and the requirements that these impose on the new middleware design.

This deliverable document is organised as follows:

- Section 1 provides a brief introduction to the requirements capture process and a glossary of terminology and abbreviations used throughout the document.
- Sections 2, 3, and 4 outline the usage scenarios, use cases and requirements in detail.
- Section 5 concludes the document.

1. Introduction

Within the Maestro project there are four application partners, each with their own domain of expertise:

- **ECMWF**: numerical weather prediction and climate re-analysis.
- **JUELICH**: global earth modelling.
- **CEA**: in-situ analysis in computational fluid dynamics
- **ETHZ**: electronic structure simulation

This document presents an analysis of the characteristics of these partners' workloads, and an extracted set of requirements for the development of the Maestro middleware.

We have used a three-level requirement capture process to generate and document the requirements associated with each application.

In the ***first level***, application partners give a high-level description of existing workflows. This analysis of the main characteristics of these workflows has a special emphasis on how data is used, described and transferred between and within applications. Each application has described one or more broad ***usage scenarios***. These document particular access patterns and ways that data is moved within and between workflow components and highlight the elements in the workflow that could benefit from the Maestro middleware. The usage scenarios are documented in section 0.

In the ***second level***, the usage scenarios are broken down into smaller, more detailed ***use cases***. These capture certain parts of each scenario in isolation and focus on specific ways that the application and workflows interact with each other and the storage system. In doing so, they flesh out both the opportunities for the Maestro middleware to provide benefit but also document the required spectrum of behaviour. They will form the basis of the application demonstrators to validate the Maestro middleware (WP6).

Finally, in the ***third level***, specific ***requirements*** are extracted from the discussed use cases. Each requirement is intended to be the smallest possible unit that is both meaningful as a design constraint and verifiable.

As the project progresses we anticipate that the requirements, use cases and even usage scenarios will become more fleshed-out by contact with the in-development middleware, and these changes will be added to the design discussion. An update on the state of the requirements and the API design will be available at month 18 (deliverable D2.3).

1.1 Glossary

CLM	Community Land Model
COSMO	A non-hydrostatic, limited area atmospheric prediction model
CSCS	Swiss National Supercomputing Centre
DFT	Density Functional Theory, a computational method for quantum mechanical simulation.
ECMWF	European Centre for Medium-Range Weather Forecasts
FDB	The Fields Database. ECMWF's domain-specific object store for meteorological data
GPU	Graphics Processing Unit. A processor optimised for display functionality.
GRIB	General Regularly-distributed Information in Binary form. A data format standardised by the World Meteorological Organisation for meteorological data.
HBM	High Bandwidth Memory
HPC	High Performance Computing
IFS	The Integrated Forecasting System
JURECA	Large supercomputer at the Jülich Supercomputing Centre
MARS	The Meteorological Archival and Retrieval System. ECMWF's perpetual archive for meteorological data.
MCT	Model Coupling Toolkit
MPI	Message Passing Interface
NetCDF	Network Common Data Format
NWP	Numerical Weather Prediction
OASIS3-MCT	Coupler between numerical codes representing different components of the climate system
ParFlow	An open-source, modular, parallel watershed flow model
SCRIP	Spherical Coordinate Remapping and Interpolation Package
TSMP	The Terrestrial Systems Modelling Platform, TerrSysMP.

2. Usage scenarios

Application workflows are composed of components, which (amongst other activities) produce, consume and move data into, out of, and between workflow components. A usage scenario describes a characteristic set of components and manner of use that is meaningful for a given application partner. This description is in the broadest possible terms that give the flavour of the particular use of the HPC system. These descriptions can then be used to generate, validate and assess the requirements for the Maestro architecture. They provide the context for a discussion of what a successful middleware design looks like.

To maximise the range of applications that benefit from the Maestro middleware (and thus the range of organisations), the usage scenarios aim to capture as much detail as possible across the widest range of different use patterns in the application partners' workflows. This is reflected in the applications that have been chosen. The usage scenarios focus on the data-movement requirements and access patterns associated with these workflows, with further ancillary details as possible.

The Maestro project accepts that the new middleware being developed is unlikely to be able to assist with all use patterns described in all usage scenarios. Nevertheless, having a full set of usage scenarios will enable the assessment of the different options available for the Maestro middleware architecture and API. A broad set of usage scenarios will also help ensure that choices made to optimise certain usages will not have deleterious impacts on required uses of the system, even if Maestro is not able to improve those cases.

2.1 Definition of usage scenarios

We expect usage scenarios to be primarily text-based descriptions of a given application workflow deemed relevant for the project. They are defined based on five fields.

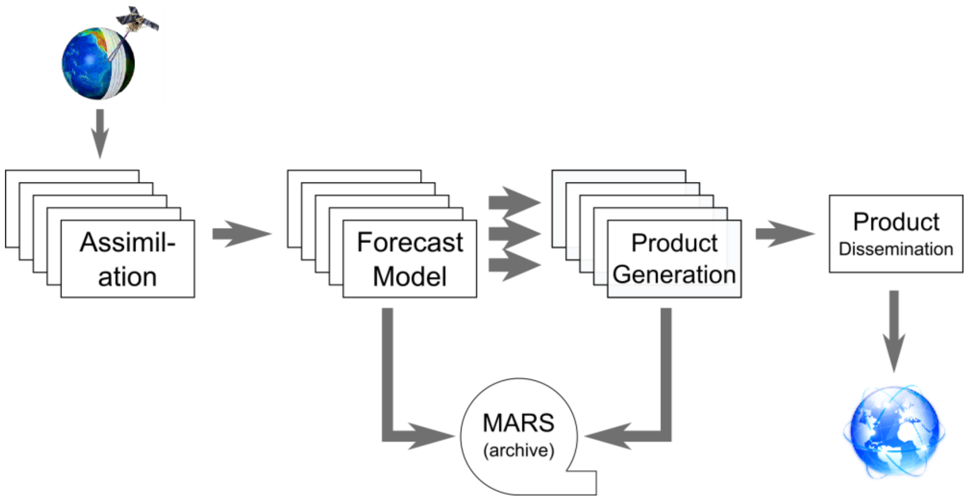
Number*	A unique identifier allocated to the scenario of the form US#.##
Name*	A short name that can be used to identify the scenario.
Owner*	The project partner responsible for the scenario.
Description*	A text-based description of the applications, systems, configurations and how they are used within a workflow to make up the scenario.
Opportunities	The opportunities envisaged within a usage scenario for which the Maestro middleware may provide benefit or enable new functionality or modes of operation.

The asterisk (*) symbol indicates that the item is required in every usage scenario.

2.1.1 Scenario description template

Number	**US1.1**
Name	
Owner	
Description	
Opportunities	

2.2 Identified scenarios

Number	**US1.1**
Name	**Operational weather forecast**
Owner	**ECMWF**
Description	The IFS is a global numerical weather prediction system used at ECMWF, which sits within a larger, time-critical operational workflow. This workflow consists of six main steps. • *Observations*: gather measurements from around the world and pre-process them. • *Data assimilation*: create an initial condition for the forecast run from the observations and past forecasts. • *Forecast model*: predict weather parameters for a sequence of given future time periods. • *Product generation*: create meaningful output data from the forecast output according to customer requirements • *Product dissemination*: make forecast products available to users. • *Archive*: preserve data for long-term use.  Assimil-ation Forecast Model Product Generation Product Dissemination MARS (archive) Main characteristics: • Large-scale parallel: ○ Uses around 25% of the whole machine – around 25,000 of 126,468 cores.

    - Product generation is highly I/O intensive.
- Time critical:
    - The operational workflow runs four times per day, with varying scope.
    - The workflow must complete within an externally contracted time limit, normally one hour.
    - Sufficient time leeway must be retained to handle errors.
    - Data assimilation, forecast and post processing run as ensembles:
        1. One deterministic (HRES) forecast.
        2. 51 Ensemble forecast members (at a lower resolution)
        3. One parallel product-generation task per output timestep.
- Domain-specific data objects:
    - The (three-dimensional) atmosphere is represented as a stack of two-dimensional slices which wrap around the earth and are called fields. During computations these fields are geographically distributed across the processes and nodes.
    - For transport and storage, fields are encoded as GRIB messages, which also contain an encoded metadata description such that GRIB messages are self-describing. These fields vary in size depending on content (for example wave-height data is smaller as it has no content over land), but most fields are around 20MiB in size.
- High-velocity and high-volume I/O
    - Up to 10 million fields are written per operational forecast, totalling 20TiB. In the next two years, this is expected to increase to 100TiB.
    - 10000-15000 fields are created per second.
    - Data is currently written to a domain-specific indexed object store, called the fields database (FDB). Data-consuming processes can then read the data from the FDB.
    - The data in the FDB is indexed according to its scientifically meaningful metadata. This takes the form of a dictionary of key-value pairs that globally and uniquely identify the object. This metadata matches the self-description in a GRIB message.
- I/O and I/O contention have a significant impact
    - Switching on I/O results in a 17% slowdown, despite I/O being asynchronous from computation via an I/O server.
    - Switching on the consumers of the output data (primarily product generation) causes further slowdown, resulting in a 26% total slowdown.
- Nature of the I/O pathway
    - I/O path is largely optimised to minimise the impact on the writing (computational) processes:
        1. Each distributed field is aggregated onto a single I/O server process. Fields are treated independently, so there is no requirement that any two fields should be aggregated on the same I/O server process.
        2. Field is written to the FDB. Each of the I/O servers in the model 52 instances writes to a distinct I/O output stream.
        3. Once per each of the 240 time steps, the I/O is flushed to disk, and indexing entries are atomically made available to reading processes.
        4. A product generation task reads across all the output streams (from the ensemble members) to produce products for that time step. A diagrammatic representation of this is presented in UC1.1.
        5. The generated products are disseminated to consumers
        6. Asynchronously and outside the time-critical window, the output

	forecast data is archived into the perpetual archive (MARS). o Write and read operations are all ***driven by*** scientifically meaningful metadata and occur simultaneously on the same indexed store. o The constraints of maintaining a consistent indexing of the data under many streams of simultaneous read and write, as well as performant access, significantly inform the system and workflow design. The resultant congestion in the metadata bottleneck is the limiting factor in the I/O pathway.
Opportunities	• Abstract the hardware details of new HPC systems away from the application developers. This is especially relevant for new systems with more complicated and heterogeneous memory and storage hierarchies. • Exploit more heterogeneous HPC environments to maximise throughput whilst minimising the cost, time and energy constraints. • Avoid unnecessary round-trips via persistent storage (currently spinning disk) between producers (IFS) and consumers (product generation). • Provide a consistent approach to monitoring and management of available memory and storage resources for the workflow. • Reduce the I/O induced slowdown of the forecast model. Even a small decrease in the 26% slowdown is of significant benefit.

Number	**US1.2**
Name	**Research experiment**
Owner	**ECMWF**
Description	Research experiments are largely made up of the same components as the operational workflow but run in a very different manner under different constraints. Most obviously, they do not run under time-critical constraints related to product-delivery to customers. Rather, the goal is to enable researchers to run as many experiments as possible in total, i.e. a throughput rather than a capacity problem. A typical experiment runs only a subset of the operational components. This may mean only running data assimilation, or the forecast model, with only a reduced subset of ensemble members. Normally, a significantly reduced set of fields is written from the forecast, enough only to inform a given research or development activity. A researcher may, for example, run new versions of the forecast system on (potentially very) old input data. Product generation to produce user-requested output is not run except as part of the process of producing a new version of the operational forecast. Main characteristics: • In aggregate, uses the bulk of ECMWF's HPC resources, 50-75% of the entire machine. • High-volume, consistent workload: o Workload is made up of many, potentially modified, standard components and run in a range of configurations with a range of input data. o Not run according to a preconfigured schedule. No pattern to the overall mix of experimental load. o The makeup of the workload in aggregate tends to evolve gradually over time according to the current research priorities. • Large I/O: o Very large aggregate volumes of I/O (the bulk of ECMWF's I/O and data written to the perpetual archive).

	◦ Amount of data per experiment varies wildly, from almost none to approaching the output of a full operational cycle. ◦ As post-processing is not normally carried out, there is much less read access to data. Read access varies according to the research investigation. A larger volume of unstructured and unpredictable manual access to data. ◦ Approximately 150TiB per day written to persistent archive, and several hundred TiB to and from spinning disk. ◦ Thousands of (aggregated) fields created per second. This varies wildly per experiment. Field sizes also vary significantly: the majority of experiments use lower resolutions, but some are working towards future resolution increases and so run at higher-than-operational resolution. ◦ Data is written to the domain-specific meteorological object store (FDB). • I/O bottlenecks: ◦ Multiple experiments may produce data, and read data, simultaneously causing congestion on the I/O systems. ◦ Read congestion is typically significantly lower than for operational workflows. But the read patterns are *extremely unpredictable*. ◦ Resources are shared to some degree with operational workflows. This increases the congestion observed by both research and operations, although prior reservations are used to mitigate this. ◦ Research produces a lot of data. Storing this until it is needed for analysis without incurring significant retrieval costs is difficult. ◦ Extremely large data sets are sometimes requested unexpectedly.
Opportunities	• Scheduling of requests according to the location of data. • Minimise the use of persistent storage, and slower layers of the storage hierarchy, for transfer of data between workflow components.

Number	**US2.1**
Name	**Multi-physics simulation pipeline**
Owner	**CEA**
Description	A traditional multi-physics simulation pipeline commonly run at CEA relies on systematic persistence of intermediate data described as Sim1 ⇨ persist ⇨ Conversion ⇨ persist ⇨ Sim2 ⇨ persist ⇨ Processing, where: • Sim1 and Sim2 are two distinct simulation codes that simulate different physics or scales, or operate on different types of dimensions and/or grids, • Conversion is an application that converts output data from Sim1 into input data for Sim2 (typically grid conversion or data projection) Our objectives are (i) accelerate such workflow (reduced elapsed time), (ii) improve the time resolution (higher frequency of transfer between stages), and (iii) reduce the amount of data persisted. This can be made possible by decoupling the persistence (still needed for potential further post-hoc processing) from the actual simulation pipeline and adopting a

streaming-like data exchange between applications of the main pipeline:

Sim1 ⇨ (filtered) Conversion ⇨ Sim2 ⇨ Processing

      ↳ (sub-sampled) persist   ↳ (sub-sampled) persist

This decoupling allows higher data flow rate in the main simulation pipeline (thus improving the time accuracy), while reducing the amount of data persisted through sub-sampling.

Data exchange between pipeline components would no longer occur through the persisted data, but through a producer-consumer streaming mechanism involving the Maestro middleware. This principle is also commonly referred as in situ or in transit workflow.

The persistence sub-branch of the pipeline would be handled by Maestro

Note that Sim1 produces many different physical quantities but the pipeline only consumes some of them (known ahead). There is an opportunity for Maestro to filter data in flight, according to user configuration.

The ***desired*** characteristics of the simulations in this usage scenario follow (note that ~ means “order of”):

- mid-scale parallelism: ~5000 cores per simulation
- Large-scale transfer of data between components along the main pipeline (from Sim1 through Sim2 to Processing):
    - Data transfer at each time step, with ~1 step/sec,
    - ~100GB written by Sim1 per step, ~50GB consumed by the pipeline per step (after filtering),
    - ~10,000 steps
    - No persistence. Only data transfer between applications
- Mid-scale I/O workload in the persistence “branch” of the pipeline:
    - Sub-sampling output of Sim1 to ~1/10 (1 write / 10s)
    - ~100GB per step, ~1000 steps
    - Data persisted
- Simultaneous execution of all pipeline applications (producer/consumer pattern, with multiple parallel consumers of the same data)
- Ability to filter which data goes through the pipeline (to only process a portion of the data produced by the first simulation)
- Ability to sub-sample data in some portion of the pipeline
- Applications will also regularly dump checkpoint data (not depicted in workflow diagram above) that will have to be managed by Maestro

Applications may perform time step rollback (and overwrite previously written data).

Opportunities

- Accelerate workflows and reduce load on parallel storage by providing fast streaming-like transport capabilities for intermediate data between workflow applications (producer/consumer patterns, caching/buffering, in memory staging, automatic deletion of consumed data)
- Improve simulation time-resolution accuracy by enabling higher data output

	frequencies from producer to consumer applications • Abstract away the details of the I/O subsystems (software and hardware) to simplify simulation and tool code bases and to improve flexibility and adaptation to new memory/storage technologies

Number	**US2.2**
Name	**Large-scale simulation and data processing**
Owner	**CEA**
Description	This scenario involves executing a large-scale parallel simulation with an intensive I/O workload and efficiently processing its data. This scenario can be viewed as a portion of the previous simulation pipeline scenario with a larger single simulation run: Simulation ⇨ Processing ⇘ (sub-sampled) persist The ***desired*** characteristics of the simulations in this usage scenario follow (note that ~ means “order of”): • Large-scale parallelism: ~50,000 cores • Large-scale transfer of data from Simulation to Processing: o Data transfer at each time step, with ~1 step every 5sec, o ~1TB per write, o ~1000 steps o No persistence. Only data transfer between applications • Mid-scale I/O workload in the persistence “branch” of the pipeline: o Sub-sampling output of Simulation to ~1/10 (1 write / 50s) o ~1TB per write, ~100 write steps o Data persisted • Simultaneous execution of Simulation and Processing (producer/consumer pattern, with multiple parallel consumers of the same data) • Ability to sub-sample data in some portion of the pipeline • Applications will also regularly dump checkpoint data (not depicted above) that will have to be managed by Maestro Applications may perform time step rollback (and overwrite previously dumped data).
Opportunities	• same opportunities as US2 • Allow for larger scale simulations to be executed, otherwise limited by the required I/O workload

Number	**US3.1**
Name	**Electronic structure calculation**
Owner	**ETHZ / CSCS**
Description	We consider a case of diagonalization based electronic structure methods for density functional theory (DFT) calculations. In this class of methods the Kohn-Sham wave-functions are represented as a linear combination of basis functions with a given set of properties such that the second-order differential Kohn-Sham equations of the DFT are transformed into a Hermitian eigenvalue problem. The eigenvalue problem is solved using direct or iterative subspace diagonalization methods. The resulting eigenvectors of the Kohn-Sham Hamiltonian represent the single-electron wave-functions which are used to construct the charge density of the system which is, in turn, used to update the effective potential and recompute the Hamiltonian. This procedure is repeated until a self-consistent density is found. The diagonalisation-based methods rely heavily on linear algebra operations. These are usually computationally intensive and are thus perfect candidates for GPU acceleration. The challenge is to design the GPU port of the code in a memory-efficient way, given that the matrices involved in the calculation are usually large and do not typically fit into the memory of a device. As a case study we consider the electronic structure library SIRIUS developed at CSCS. This library implements a plane-wave pseudopotential method and is designed to work with electronic structure community codes such as Quantum ESPRESSO. The library supports CPU and GPU backends and is written in C++ with CUDA, OpenMP and MPI programming models.
Opportunities	● Optimize the movement of data between host memory (CPU) and the HBM of the GPU, back and forth, based on the communication requirements (targeted memory, timings, transport layer). ● Insure the reusability of the allocated memory space.

Number	**US4.1**
Name	**Global Earth Modelling**
Owner	**JUELICH**
Description	TerrSysMP was developed to simulate the interaction between lateral flow processes in river basins and the lower atmospheric layer. The software is composed of three model components: COSMO, CLM and ParFlow and an external coupler, OASIS3-MCT. COSMO is a non-hydrostatic limited-area atmospheric model. CLM (Community Land Model) is used in the field of ecological climatology to study the effect of terrestrial ecosystems on climate. ParFlow is a hydrologic model for surface and subsurface flow. OASIS3-MCT is built on top of MCT (Model Coupling Toolkit) and provides fully parallel implementation of coupling field re-gridding and exchange. In TerrSysMP, CLM is coupled to both

	ParFlow and COSMO. ParFlow and COSMO are only coupled to CLM; they are not coupled with each other. Workflow: • Preprocessing (5 % of total time) ○ (re-)Generate SCRIP (Spherical Coordinate Remapping and Interpolation Package) files ○ Inspect field output for errors from previous cycle • Simulation (80 % of total time) ○ Start with field data from previous cycle (if any) ○ Run COSMO, CLM and ParFlow for 3h to complete a cycle • Postprocessing (15 % of total time) ○ Analyze data using the Climate Data Operators toolkit Pre/post-processing run as separate jobs on the data produced at each 3h simulation cycle. Overlap between the simulation and pre/post-processing is possible. Simulation at moderate resolution: • Scales up to 20 nodes (200 cores, 1 process per core) on JURECA (supercomputer equipped with 2x Intel Xeon E5 CPUs per node) • Memory usage per node is 20 GiB • Data is written to disk at the end of the simulation. For a three-hour run, the data produced is 200 GiB. • Per-coupling step 51 fields are exchanged between models. Fields are represented as 2D or 1D arrays of type double with size ~1.5 MiB (~180000 number of elements). ParFlow sends 20 fields to CLM, COSMO sends 12 fields to CLM, CLM sends 10 fields to ParFlow and 9 to COSMO. • Each field is exchanged between models at a coupling rate of 4s • ParFlow and COSMO take up between 80-100 processes each, while CLM occupies 15-30 processes. Features • Fields are distributed and exchanged via MPI across processes and nodes • Frequent row/column major order transformations • Frequent redistribution (remapping) of data • Frequent interpolation of data based on SCRIP • I/O based on netCDF4
Opportunities	• Dynamic load balancing of coupled models • In-transit transformations

3. Use cases

Use cases extract specific actions, functions or instances from the usage scenarios. A use case will take one aspect of a broader usage scenario and describe it in detail. It clarifies what the pre-conditions of the workflow and component are, and what system and middleware provisions are required to make it work. Use cases also give shape to the precise steps constituting the workflows, and what the desired outcome of interactions with the middleware are.

Use cases differ in their level of detail and focus. Some highlight a specific component of the workflow and describe it and its interactions in detail. Others will describe one aspect of the data interactions of the workflow in a manner which involves all components. In all cases, a use case refines some aspect of the associated usage scenario in sufficient detail to facilitate the generation of requirements.

Given the range of usage scenarios and partners, it is inevitable that there will be overlap of focus between some of the use cases. This is unsurprising and will help to highlight the areas with the most convergence on the workflows' needs for the Maestro middleware.

3.1 Definition of use cases

We expect use cases to be generated from usage scenarios. Use cases are defined based on the following eight fields:

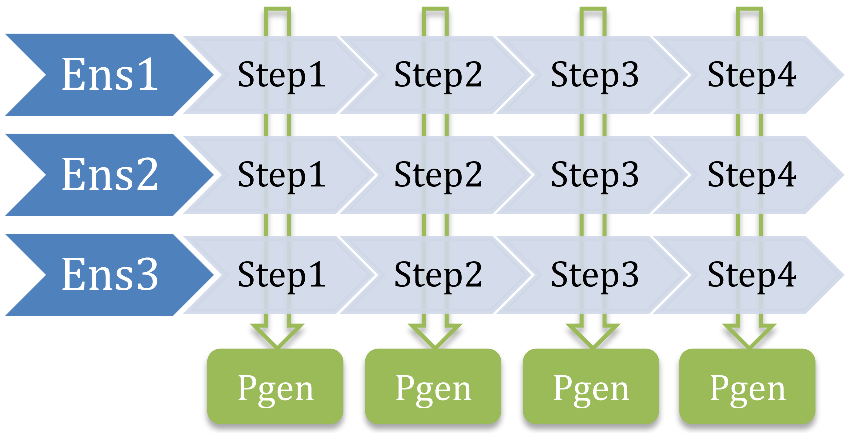
Number*	A unique identifier allocated to the use case of the form UC#.##
Name*	A short name that can be used to identify the use case.
Owner*	The project partner responsible for the use case.
Usage Scenario	The usage scenario from which the use case derives.
Description*	A text-based description of the use case.
Pre-conditions	The pre-conditions that need to be fulfilled for the use case to be valid in the context of the Maestro middleware.
Workflow	A typical workflow may describe the behaviour of the workflow in the context of Maestro.
Post-conditions	The post-conditions that need to be fulfilled for the use case to be valid in the context of Maestro.

The asterisk (*) symbol indicates that the item is required in every use case.

3.1.1 Use case description template

Number	**UC1.1**
Name	
Owner	
Usage scenario	
Description	
Pre-conditions	
Workflow	
Post-conditions	

3.2 Identified use cases

Number	**UC1.1**
Name	**Access semantically-related datasets**
Owner	**ECMWF**
Usage scenario	**US1.1**
Description	Within ECMWF's operational pipeline, *product generation* describes the workflow components that consume generated forecast output and post-process it according to customer-specified requirements. Each product generation task operates on a single time step's output. They consume data across all output streams required to satisfy its configured  Ens1 Ens2 Ens3 Step1 Step2 Step3 Step4 Pgen requirements.
Pre-conditions	• Data from the forecast system is available.
Workflow	1. Post-processing determines the set of meteorological objects required to

	satisfy its consumer-specified requirements. 2. Post-processing requests the objects from Maestro as a set. 3. Maestro retrieves the relevant data and makes it available in an efficient order. 4. Post-processing tasks are then able to iterate over the requested data as it becomes available.
Post-conditions	• All post-processing jobs have received their input data.

Number	**UC1.2**
Name	**High-velocity production of meteorological objects**
Owner	**ECMWF**
Usage scenario	**US1.1**
Description	During computation, forecast data is in the form of two-dimensional slices of the atmosphere geographically distributed across the compute processes. Once per step the pieces are sent asynchronously to dedicated I/O servers which assemble complete fields and pass these to the I/O subsystems, from where they become accessible to a range of consumers. Consumers and producers access the data simultaneously. ECMWF anticipate a future rate of 20,000 objects being produced or accessed per second.
Pre-conditions	• I/O servers receive the data slices from the forecast computation.
Workflow	1. The I/O server process assembles the meteorological field and attaches scientifically meaningful metadata. 2. It pushes this data to Maestro. 3. Maestro takes ownership of the data. Maestro will ensure it is made available in a timely manner to appropriate consumers. (Note that consumers request before is available in Maestro). 4. The I/O server can then move onto generating the next field from the data slices.
Post-conditions	• Data is made available to consumer tasks as soon as possible.

Number	**UC1.3**
Name	**Operators restart forecast from previously computed forecast data**
Owner	**ECMWF**
Usage scenario	**US1.1**
Description	The operators decide that the forecast must be restarted from the latest available checkpoint. That would entail restarting from previous forecast data and it would exclude the data assimilation workflow component. A typical reason for the need to restart would be catastrophic system failure occurs during operational forecast.

Pre-conditions	• Sufficient checkpoint data is available.
Workflow	1. Request checkpoint data. 2. Reinstate Maestro with latest available checkpoint data. 3. Re-start operational workflow from the ‘forecast run’ component. 4. Consumers make scientific use of the data.
Post-conditions	• Workflow is running as if interruption and restart had never occurred (as per other use cases).

Number	**UC1.4**
Name	**Operators monitor system characteristics**
Owner	**ECMWF**
Usage scenario	**US1.1, US1.2**
Description	At ECMWF, both operational and research workflows are continuously monitored by the operators. This is to be able to immediately detect and react to events including failures, heavy system load and unbalanced work distribution. During operational runs, operators monitor the system characteristics of the operational workflow as well as each workflow component. As they are very familiar with the (documented) behaviour of each component, they are able to provide a rapid response to unusual behaviour. Operators do not monitor individual research workflows, but they monitor the overall system, which research workflows are a part of.
Pre-conditions	Many workflows are running simultaneously, each producing and consuming meteorological data.
Workflow	1. Operator monitors the system. 2. Operator observes the system exceeding safe limits. 3. Operator takes documented action or contacts the person responsible for the workflow manager’s configuration.
Post-conditions	Information about system characteristics can be fed back into the workflow manager.

Number	**UC1.5**
Name	**Operators monitor workflows**
Owner	**ECMWF**
Usage scenario	**US1.1**
Description	Operators use ECMWF’s workflow manager, ecFlow, as well as other tools to monitor the operational workflow.
Pre-conditions	A workflow with well-understood expected workflow characteristics is running. This will typically mean the operational workflow.

Workflow	1. Operator monitors the operational workflow. 2. Operator observes unexpected behaviour, or failures, in one or more of the workflow components. 3. Operator takes documented action or contacts the person responsible for the workflow manager's configuration.
Post-conditions	

Number	**UC1.6**
Name	**Sustained high-volume production of meteorological objects**
Owner	**ECMWF**
Usage scenario	**US1.1, US1.2**
Description	Multiple large experiments run simultaneously. The runs are not time-critical and the per-experiment rate at which fields are produced is not typically problematic. However, the total volume of data pushed through the system is extremely large. Consumers may request arbitrary volumes of data in an unpredictable read pattern according to current research requirements.
Pre-conditions	• I/O servers receive the data slices from the forecast computation.
Workflow	1. The I/O server assembles the meteorological fields and attaches scientifically meaningful metadata. 2. It pushes the data to Maestro, defining a lifetime that is long relative to the workflow runtime. 3. Maestro takes ownership of the data. 4. Maestro manages and holds onto large quantities of data. 5. Consumers make scientific use of the data.
Post-conditions	• Data is available for consumers for the duration of the data's life time.

Number	**UC1.7**
Name	**Consumer applications request sets of objects**
Owner	**ECMWF**
Usage scenario	**US1.1, US1.2**
Description	Consumers in the workflows request sets of fields rather than individual ones. These sets are organised along different axes of the scientifically meaningful metadata (such as "all levels in the atmosphere"). Any one object may belong to several object sets.
Pre-conditions	Objects are available to consumer applications through Maestro.
Workflow	1. Consumer application requests a set of objects (say a given temperature field for all levels) according to scientifically meaningful metadata. (The sets of objects requested by multiple consumers may be disjoint, or not).

	2. The application may run asynchronously or block waiting for the data. 3. The corresponding data is returned to the application by the Maestro middleware.
Post-conditions	Consuming application(s) have received the objects that they have requested.

Number	**UC2.1**
Name	**Applications write/read Data Collections to/from Maestro**
Owner	**CEA**
Usage scenario	**US2.1, US2.2**
Description	Most simulation codes and tools at CEA use the in-house Hercule library to perform IO related tasks. Within Hercule data is conceptually organized into the following data container hierarchy: Collection > Records > Datasets, where dataset is a multi-dimensional array, record is an immutable group of related datasets and collection is a log-structured sequence of related records (see R2.1 for further explanation). Applications will use Maestro to write/read similar data collections. Note that write/read operations can be collective (ie. multiple processes coordinate themselves to write into the same dataset and record) or independent (ie each process will write its own datasets and records).
Pre-conditions	
Workflow	1. To write data, an application opens (or creates) a collection in Maestro specifying a unique ID/name, and provide some access properties. 2. The application use Maestro API to create a record, fills the record with named datasets and metadata, and closes it. 3. Upon close, Maestro takes ownership of the record and all contained data 4. The application closes the collection. 5. To read data, the application opens the collection in Maestro by specifying its ID/name along with access properties (e.g. read-only) 6. The application opens a record, reads the datasets and metadata it needs and closes it.
Post-conditions	

Number	**UC2.2**
Name	**“Streaming” data records between producers and consumers**
Owner	**CEA**
Usage scenario	**US2.1, US2.2**

Description	In a “live” pipeline execution mode in which applications are executed simultaneously, data needs to be streamed from producers to consumers. In such execution mode, exchange of data and synchronization should be based on publish/subscribe pattern and the use of iterators instead of random access.
Pre-conditions	
Workflow	1. User provide its pipeline configuration into Maestro prior executing it (collections (see UC2.1) used for exchanging data between applications, number of producers, number of consumers, etc.) 2. User launches the pipeline 3. Consumer applications declare to Maestro the ID/name of the data collection they want to subscribe to 4. Maestro provides an iterator to consumers which start to consume it (blocked while waiting for records to be produced) 5. Producer applications declare to Maestro the ID/name of the data collection in which they will publish/push new records 6. Producer applications start generating data and sink data into the pipeline (by pushing records to Maestro) 7. Maestro makes the data records available to waiting consumers
Post-conditions	

Number	**UC2.3**
Name	**Ownership and persistence of a data**
Owner	**CEA**
Usage scenario	**US2.2**
Description	Maestro takes ownership of the data and manages its persistence beyond the lifetime of the workflow that generated it. The data is made accessible to reader/consumer processes outside the workflow. The user declares in Maestro configuration that data produced in particular data collection (see UC2.1) will be persisted after the execution of the workflow.
Pre-conditions	The user has provided some configuration information to Maestro dedicated to her workflow and specifying which data collection will be persisted after the execution.
Workflow	1. The producing application generates and pushes a new data records from various data collections in Maestro 2. Maestro takes ownership of the data records and based on its configuration, proceeds with the necessary actions to ensure persistence of the data records belonging the data collection specified in configuration
Post-conditions	The persisted data collection is available through Maestro for future workflows.

Number	**UC2.4**

Name	**Simulation checkpoints spooling**
Owner	**CEA**
Usage scenario	**US2.2**
Description	An application uses Maestro to write/read its checkpoint data. By configuration, the user can instruct Maestro to persist only the N last checkpoints once the simulation/job allocation is over. Objectives for this use case are: • large number of checkpoints (up to each time step, ~1 step/sec) • ~100GB per checkpoint • The number of checkpoints in the spool to be user-configurable • accommodate time step rollback (overwrites previously dumped checkpoints)
Pre-conditions	The user has provided the relevant configuration information to Maestro.
Workflow	1. The simulation generates and pushes checkpoint dataset in Maestro 2. Maestro manages the spool of checkpoint datasets to keep only the last N 3. At the end of the job, Maestro to persist the last N checkpoints
Post-conditions	Last N checkpoints are available for the simulation to restart

Number	**UC2.5**
Name	**Time step rollback on persisted data**
Owner	**CEA**
Usage scenario	**US2.2**
Description	A simulation code uses Maestro to persist its output and checkpoint data. For each of these two collections (see UC2.1), a series of records indexed by their time step are produced. At some point during the execution, the simulation may “decide” to rollback and re-write new output and checkpoint data (possibly at different time steps). For persisted data, Maestro should detect this rollback and overwrite previously persisted data.
Pre-conditions	A simulation code is running and has already persisted some data records through Maestro.
Workflow	1. The simulation rolls back and pushes new records to Maestro with a time step prior to some time steps already persisted 2. Maestro detects the records history is being re-written 3. Maestro truncates the record history to the last record before the newly pushed record (and discard irrelevant records) and append the new record
Post-conditions	The persisted record history reflects the records produced by the simulation considering all rollbacks.

Number	**UC3.1**
Name	**Move data from CPU to GPU memory**
Owner	**ETHZ / CSCS**
Usage scenario	**US3.1**
Description	When the data cannot fit into GPU memory, allocation/deallocation is currently manually done by blocks, meaning that the memory management is fully controlled by the developer. This memory management is implemented in a dedicated library within SIRIUS. To give a coarse estimation of the data sizes created during the pseudopotential DFT ground state calculations let's consider a unit cell with N_a atoms. The largest memory consumer is usually an array of wave-functions $□_{i\mathbf{k}}(\mathbf{G})$ which is computed for the set of irreducible **k**-points. The number of **k**-points depends on the size of the unit cell and its symmetry and can be estimated as N_k = Int(1000/N_a + 1). The number of individual wave-functions is approximately N_{wf} = 10 * N_a and the number of plane-waves is typically N_G = 2000 * N_a. The total size of the wave-function array is then N_k * N_{wf} * N_G double complex elements. For the unit cell of 100 atoms this is approximately 30 Gb of data. There is also a work array for the Davidson iterative solver used to diagonalize the Kohn-Sham Hamiltonian. The size of the work array is 12 * N_{wf} * N_G which is approximately 36 Gb of data for a 100-atom unit cell. The wave-functions are searched independently for each **k**-point and this is the prime parallelization strategy for the DFT codes. However, the size of the work arrays for the Davidson iterative solver can be larger than the GPU memory and the second level of parallelization within a k-point is necessary. Both levels of parallelism are implemented in SIRIUS. The DFT ground state density is searched in the self-consistent fashion during the DFT iteration cycle. The number of DFT iterations depends on the system and can take up to 100 steps. During each step the wave-functions are recomputed using Davidson iterative method and then used to updated the new charge density. The operations involved during this phase are mostly complex matrix-matrix multiplications (zgemm) and fast Fourier transforms (FFT) which are accelerated by GPUs. The GPU memory management steps that the code performs during the iterative solver phase are: ● allocate work-arrays for Davidson iterative solver at a given k-point ● allocate memory for wave-functions at a given k-point ● copy wave-functions to GPU for the initial guess ● generate new wave-functions using iterative solver (involves zgemm + FFT operations) ● copy new wave-functions back to host memory ● deallocate wave-function and work arrays The GPU memory management steps that the code performs during the charge density summation are: ● allocate. memory for wave-functions at a given k-point ● copy wave-functions to GPU

	● generate new charge density (involves zgemm + FFT operations) ● copy new wave-functions back to host memory ● deallocate wave-function The data movements are synchronous. A GPU kernel cannot start until all the data is available on the device. It is also important to note that the source of the data is not in use during the transfer. In addition, the host memory always stores the correct and valid data arrays.
Pre-conditions	● Hardware requirements: multicore cluster with GPUs ● Dataset large enough to not fit into GPU memory
Workflow	1. A large dataset has to be moved from CPU memory to GPU memory 2. The dataset is pushed to Maestro as a set of objects fitting in GPU memory 3. Maestro manages movement of data between DRAM (CPU) and HBM (GPU) 4. Data object ownership is taken by the GPU computational kernel 5. Ownership of the data is given back to Maestro then to SIRIUS
Post-conditions	● All of the relevant data has been consumed.

Number	**UC3.2**
Name	**Management of memory resources**
Owner	**ETHZ / CSCS**
Usage scenario	**US3.1**
Description	As described in UC3.1, copying wave-function to the GPU memory for computation implies repeatedly allocating and deallocating GPU memory. The same behaviour is also observed on host memory. As the frequency of allocation and deallocation is correlated to the number of steps required for computation like described in UC3.1, a performance overhead is observed when the number of memory operations performed on both memories is high. A memory pool where allocated memory spaces are re-used is necessary to mitigate this overhead.
Pre-conditions	● Multiple allocation/deallocation has to be performed
Workflow	1. A memory space is required for a piece of data 2. Maestro retrieves an unused allocation memory space and offer it to SIRIUS 3. Memory space is locked, used and released
Post-conditions	● The memory space is queued in the Maestro memory pool

Number	**UC4.1**
Name	**Exchange 2D fields between climate models**
Owner	**JUELICH**
Usage scenario	**US4.1**
Description	During the execution of each climate model (COSMO, CLM and ParFlow), 2D fields are transferred, remapped and interpolated between the models.
Pre-conditions	Source model asynchronously sends distributed 2D field
Workflow	1. Map portions of the source array needed for interpolation into target processes based on the interpolation matrix generated by SCRIP 2. Transfer in parallel from source to target processes 3. Convert to/from row/column-major order if necessary (e.g. when transferring between Fortran and C climate models)
Post-conditions	Target model synchronously receives distributed 2D interpolated field

Number	**UC4.2**
Name	**Dynamic Load Balancing of Coupled Models**
Owner	**JUELICH**
Usage scenario	**US4.1**
Description	Currently load-balancing between climate models is done a-priori and heuristically, based on intuition and experience. Determining optimal configuration of the climate models over the available processes/cores requires extensive profiling. As a consequence, TerrSysMP is often run in suboptimal configuration. Maestro can be leveraged to implement a profile-driven workflow providing a close to optimal TerrSysMP configuration over the available resources.
Pre-conditions	COSMO, CLM and ParFlow are distributed according to an initial user-provided process distribution
Workflow	1. Run TerrSysMP for a user-specified number of coupling cycles, measuring idle time in each of the models 2. Analyze performance data for each model and feed it back to Maestro's workflow manager 3. Generate a new process distribution based on idle time in each model and redistribute the model data. 4. Repeat 1), 2) and 3) until a user-specified termination criteria (e.g. maximum number of iterations) 5. Run normal TerrSysMP simulation with the process distribution generated by 1) - 4)
Post-conditions	COSMO, CLM and ParFlow are re-distributed to reduce idle times compared to the initial, user-provided distribution

4. Requirements

Requirements aim to capture anything that should influence the design or implementation of the Maestro middleware. Given that there are time and resource constraints on the development of the middleware, and that requirements from different sources may come into conflict, the desirability of each requirement should also be expressed.

Requirements are to emerge from and impact all areas of the system. Some of them are complicated and have multiple interlocked components to them. In these cases, the overarching requirement is expressed first, and more precise details and constraints are added in further follow-on requirements that *relate to* the first.

Requirements aim to be finely granular, such that they can be clearly enumerated and checked against the proposed middleware designs.

4.1 Definition of requirements

Requirements are recording in tabular form according to the following six fields:

Number*	A unique identifier allocated to the requirement of the form R#.##
Desirability*	An indication of how important this requirement is to the Maestro project, as described below.
Use case(s)*	The use case(s) from which the requirement derives.
Relates to	Other requirements that are related to this one.
Description*	A text-based description of the requirement.
Justification	An explanation of why the use case poses this requirement on the design of the Maestro middleware.

The asterisk (*) symbol indicates that the item is required in every requirement.

Desirability levels follow the five-level scheme outlined in RFC 2119

Must have	Requirements that if not met would mean the developed middleware is of no use to the application.
Should have	Important requirements, but not strictly necessary for the success of the project. Design should avoid preventing these requirements from being satisfied in the future.
Optional	Requirements that are not critical to the project, but that could improve performance or add useful functionality.
Should not have	Constraints that if met would have a negative but not terminal impact on the success of the project.
Must not have	Design constraints that if met would result in failure of the middleware to meet the goals of the project.

4.1.1 Requirement definition template

Number	**R1**
Desirability	Must have/Should have/Optional/Should not have/Must not have
Use case(s)	
Relates to	
Description	
Justification	

4.2 Identified requirements

Number	**R1.1**
Desirability	Must have
Use case(s)	**UC1.1**
Relates to	**R1.2**
Description	Objects handled within Maestro can be described and handled according to user-defined domain-specific metadata.
Justification	Scientific software is written by scientists who think in scientifically meaningful terms. One of the goals of a good data-handling system is to abstract the locations and storage-handling mechanisms away from the application developers and to provide mechanisms which facilitate their work. The Maestro system is going to substitute some existing functionality within ECMWF's workflows, which already operate according to scientific metadata described according to the MARS language. If this cannot be supported, then the substitution cannot take place.

Number	**R1.2**
Desirability	Should have
Use case(s)	**UC1.1**
Relates to	**R1.1**
Description	User-defined metadata should take the form of a dictionary of key-value pairs, or equivalently as a set of arbitrarily named attributes.
Justification	The Maestro system is going to substitute some existing functionality within ECMWF's workflow, which already operate according to scientific metadata described according to the MARS language. This structure queries in these terms.

Number	**R1.3**
Desirability	Must have
Use case(s)	**UC1.1, UC1.7**
Relates to	**R1.1, R1.2**
Description	Object retrieval should support metadata queries that describe lists of values.
Justification	Consumer operations retrieve and operate on sets of fields.

Number	**R1.4**
Desirability	Optional
Use case(s)	**UC1.1, UC1.7**
Relates to	**R1.3**
Description	Object retrieval may support metadata queries that describe ranges of values, for example by automatically expanding it to a list.
Justification	Consumer operations retrieve data according to the MARS language which supports such ranges. An example would be post-processing operations requiring a given field from all 50 ensemble members. This expansion could be carried out on the client side if required so long as R1.3 is satisfied.

Number	**R1.5**
Desirability	Optional
Use case(s)	**UC1.1, UC1.7**
Relates to	**R1.4**
Description	Object retrieval may support wildcards, such as ‘all’ and ‘every’. This may be interpreted as all available data matching the request.
Justification	These types of requests occur at ECMWF when researchers request data for verification and/or analysis. The expansion could be carried out on the client side if required.

Number	**R1.6**
Desirability	Optional
Use case(s)	**UC1.1**
Relates to	
Description	Support describing a set of fields as ‘complete’.
Justification	Some consumers, in particular product generation, may want to iterate over ranges of data. When iterating over a sparse set of data that has been retrieved, it is important to know when the generation of all the data has been completed. This functionality could be implemented at an application level using sentinel objects.

Number	**R1.7**
Desirability	Must have
Use case(s)	**UC1.1**
Relates to	
Description	Once data objects handed over to Maestro, transport and access must no longer be the application’s concern. Maestro should take care of maintaining unique references to the data objects, and facilitate locating objects according their metadata.
Justification	Otherwise responsibilities Maestro is meant to substitute within ECMWF’s current workflow will have to be kept.

Number	**R1.8**
Desirability	Must have
Use case(s)	**UC1.2, UC1.6, UC1.7**
Relates to	
Description	Objects may have a minimum lifetime (possibly zero).
Justification	An unknown number of consumers may access a given field or set of fields, in a manner that is not known prior to runtime. Further, the operators may intervene and cause already completed tasks to run again. Maestro therefore must guarantee the existence of a field for a user-defined period.

Number	**R1.9**
Desirability	Optional
Use case(s)	**UC1.2, UC1.6, UC1.7**
Relates to	
Description	Objects may have a maximum lifetime (possibly infinite).
Justification	It is possible that an object is produced but no consumer ever requests it.

Number	**R1.10**
Desirability	Must have
Use case(s)	**UC1.3, UC1.4, UC1.5**
Relates to	
Description	When an error occurs in the Maestro software/system that impacts the ability of the Maestro system to correctly satisfy requests, it should be treated as a hard failure and be propagated to all other Maestro-dependent instances.
Justification	ECMWF’s operational system does not require best-effort to keep going at all costs. It is better to fail fast in a way that it is visible to the operators, and can be investigated by analysts quickly, than it is to stall by automatically retrying.

Number	**R1.11**
Desirability	Optional
Use case(s)	**UC1.4, UC1.7**
Relates to	
Description	Maestro may have further ‘soft’ failure modes, if these do not impact the ability to access data according to the specification. As an example, if a duplicate copy of an object is lost, this may impact performance and may violate resiliency guarantees, but it is not a hard error from the perspective of the workflow.
Justification	Not all errors impact the external behavioural correctness of the system. Although they may have performance impacts, it is not necessary to hard stop the entire system in this context – but it may be of use to report these errors. This is especially true if the error reports can be brought to the immediate attention of the operators. We anticipate that the implementation will default to always reporting hard errors, and softer failure modes will be added later in the development process.

Number	**R1.12**
Desirability	Should have
Use case(s)	**UC1.4, UC1.5**
Relates to	
Description	Errors propagate from Maestro to dependent workflow components when they next access the Maestro API.
Justification	The Maestro middleware failing hard mean that it should make no best-attempt to keep going, rather than that it needs to be aggressive in tearing the system down. This makes it unnecessary to build a low-latency push system to actively propagate errors.

Number	**R1.13**
Desirability	Should have
Use case(s)	**UC1.3**
Relates to	
Description	Maestro may have a mechanism to enable an out-of-band workflow component to inject knowledge and data into it outside of normal process.
Justification	If a workflow has to be (re)started partway through, especially after system failure, it is necessary to get the system into a state where it can resume from a position that is not the start of the workflow. This will likely involve seeding the workflow, and hence Maestro, with data and state information out-of-band.

Number	**R1.14**
Desirability	Should have

Use case(s)	**UC1.4**
Relates to	**R1.12**
Description	Maestro to record and communicate usage statistics – type of storage used and their load, write (production) rate, read (consumption) rate, etc. – to human operators or system monitoring/logging tools.
Justification	Operators can carry out this task with ECMWF's current system. Recorded telemetry information is extremely useful for diagnosing issues.

Number	**R1.15**
Desirability	Must have
Use case(s)	**UC1.2**
Relates to	
Description	Maestro to be able to cope with at least 20,000 object creations per second per workflow. This is sustained for most of the workflow duration.
Justification	Current load is in the region of 10,000–15,000 objects per second and is expected increase.

Number	**R1.16**
Desirability	Must have
Use case(s)	**UC1.6**
Relates to	
Description	Maestro to be able to cope with the creation of at least 150TiB data in an hour per workflow.
Justification	Current maximum load is around 100TiB in an hour per workflow and it is expected increase.

Number	**R1.17**
Desirability	Must have
Use case(s)	**UC1.2, UC1.6**
Relates to	**R1.4**
Description	Objects within Maestro are handled transactionally. No partial state must exist.
Justification	The reliability and consistency of the operational workflow is paramount. Data corruption must be avoided under all circumstances, even if this reduces efficiency. As a result, data-write and indexing must be atomic and consistent from the perspective of a reading process. Either an object is not available, or the entire correct object must be returned. This is especially important if workflow components are rerun, when either old *or* new data must be returned (and

	correctly identified), but it must be impossible to retrieve part-new and part-old data in a request.

Number	**R1.18**
Desirability	Should have
Use case(s)	**UC1.2, UC1.5**
Relates to	**R1.3**
Description	The way Maestro handles objects in case of errors within Maestro should be well-defined and consistent.
Justification	Software and processes need to be designed around the semantics of the system. The processes for dealing with errors may be used rarely, but they must be robust, and reliable.

Number	**R1.19**
Desirability	Must have
Use case(s)	**UC1.1**
Relates to	**R1.20**
Description	Objects within Maestro are handled consistently. No risk of undefined behaviour or data corruption.
Justification	Meteorological data must only be changed according to a pre-defined set of rules.

Number	**R1.20**
Desirability	Should have
Use case(s)	**UC1.7**
Relates to	
Description	If a set of objects is being iterated over by a set of consumers, and one of these consumers fails, the entire iteration should be considered to have failed.
Justification	If a set of consumers is iterating over data in a distributed fashion, as determined by the middleware, then from an external perspective if one of the consumers fails then the entire process has failed. It is likely that the entire process will need to be rerun.

Number	**R1.21**
Desirability	Should have
Use case(s)	**UC1.7**
Relates to	

Description	A single consumer must be able to iterate over a set of objects, even when this is too large to fit in memory at once.
Justification	If a set of objects is retrieved, and iterated over, it is desired to give the entire request to Maestro to handle in the most efficient manner possible. Maestro should provide data as fast as it can, but throttled according to the resource constraints on the consumer.

Number	**R1.22**
Desirability	Should have
Use case(s)	**UC1.7**
Relates to	
Description	Multiple consumers must be able to iterate over a set of requested objects simultaneously, either all of them consuming all the objects or distributed amongst the consumers as they are able to consume them.
Justification	This allows efficient task-based parallelism to be supported at the middleware level.

Number	**R1.23**
Desirability	Should have
Use case(s)	**UC1.7**
Relates to	
Description	Once consumers have obtained a data set and started iterating over the elements, Maestro must manage its resources so that it will not be out-of-memory-killed at least until the iteration completes.
Justification	It is the application's responsibility to ensure that it does not use too much memory. However, if the application is iterating over a large data set, it is not able to directly constrain Maestro's total resource usage. Maestro needs to throttle its retrieves according to resource availability on the consumer nodes.

Number	**R1.24**
Desirability	Must have
Use case(s)	**UC1.1**
Relates to	**R1.18**
Description	Once data objects are handed over to Maestro, they must be immutable.
Justification	Although data objects may have partial state during creation, once finalised they represent meteorological data that is unique. Accesses to this data must be consistent, and its appearance must be transactional. Any attempt to overwrite said data should produce a new (also immutable) object rather than modifying the original.

Number	**R1.25**
Desirability	Optional
Use case(s)	**UC1.2, UC1.7**
Relates to	
Description	Maestro may support iteration granularity larger than a single data object.
Justification	Within the ECMWF workflow, the granularity of the data required for processing is not necessarily a single field. The simplest example is that of wind data, where two fields are required. Another example would be time-series data. Multiple data elements must thus be provided together during iteration.

Number	**R1.26**
Desirability	Should have
Use case(s)	**UC1.7**
Relates to	
Description	Consumer applications should not be required to know the size of the data prior to requesting it.
Justification	That would be an unnecessary constraint on the application; Maestro should be able to manage memory resources. Or supply the required tools to enable the application to do so at the appropriate moment.

Number	**R2.1**
Desirability	Must have
Use case(s)	**UC2.1**
Relates to	
Description	Maestro must allow a conceptual data organization based on the following data container concepts: - array: a multi-dimensional typed array - record: a grouping of related datasets published at the same time (e.g. simulation output fields at a specific time step) - collection: a grouping of related records (e.g. checkpoint data collection, post-processing data collection)
Justification	Most simulation codes and tools at CEA use the in-house Hercule library to perform IO related tasks. To minimize effort to use Maestro, we intend to port Hercule on top of Maestro (using Maestro as a backend store). This is a fundamental organization of data within Hercule.

Number	**R2.2**
Desirability	Must have
Use case(s)	**UC2.1**
Relates to	
Description	Maestro must provide the ability to add (and query) named attributes (metadata) to data arrays, records and collections.
Justification	This seems a common requirement in for typical data storage system.

Number	**R2.3**
Desirability	Should have
Use case(s)	**UC2.1**
Relates to	
Description	Maestro should provide a way to hierarchically group related data arrays within a record (and query this group structure), similar to HDF5 groups.
Justification	This will assist us (CEA) in porting our Hercule I/O library on top of Maestro (though this hierarchical organization could be implemented in the data naming, having a native system for hierarchical grouping would be more efficient).

Number	**R2.4**
Desirability	Must have
Use case(s)	**UC2.2**
Relates to	
Description	A data collection must accommodate records produced by multiple independent producers. Hence, labelling of data records is at least a 2-tuple made of a sequence integer (the sequence number of a record with the list of records) and a producer ID integer. Maestro must provide an indexing or attribute system to be able to support this.
Justification	When one of our (CEA) simulation writes data with Hercule I/O library, each rank produces a record which is self-describing and contains all data and metadata produced by that rank. Unlike in other I/O libraries, these records are not combined to produce a global view of the domain (or a different domain decomposition). They are written as is (keeping the same domain decomposition). The recombination into a different domain decomposition is deferred until reading if needed. This is the default behaviour and is part of optimisation for writing. This principle implies that records are indexed by their issuing sequence (here the time step number) and by the producer id (here the MPI rank). Therefore, to allow porting Hercule to leverage Maestro, it must provide an indexing or attribute system compatible with this approach.

Number	**R2.5**
Desirability	Should have
Use case(s)	**UC2.2**
Relates to	
Description	In “streaming” mode, a data collection should accommodate multiple consumers organized into groups, similar to Kafka groups (or Redis Stream consumer groups). Records are broadcasted to all consumer groups. Consumers within a consumer group retrieve records from a queue.
Justification	This allows mixing two types of pattern: - all records: consumers registering in different groups receive (and process) all the records, e.g. a statistical processing that integrates data over time needs all records - fair-share: consumers registering in the same group split the work by concurrently consuming records from a queue (hence a record will be processed by only one consumer within the group), e.g. a stateless processing such as an image rendering can work independently on a record and does not need the whole sequence

Number	**R2.6**
Desirability	Should have
Use case(s)	**UC2.2, UC2.3, UC2.4**
Relates to	
Description	Maestro should provide a configuration system to describe properties of the workflow/pipelines for which it will manage the data exchange, where these properties are known in advance.
Justification	An example property that could be leveraged by Maestro at configuration time would be the identifiers of data that are known to be consumed by a consumer application. Let’s say a simulation code dumps many physical quantities, but for a particular workflow, the consumer application only requires a few of them. This is an information that Maestro could leverage to avoid transporting data from the producer application that will not be consumed at all.

Number	**R2.7**
Desirability	Should have
Use case(s)	**UC2.2**
Relates to	**R2.5**
Description	In “streaming” mode, a consumer should have the opportunity to declare in advance (in configuration) the data arrays it will consume.
Justification	In our (CEA) simulation pipelines, upstream simulation codes produce many

	arrays that are not necessarily consumed by downstream consuming applications. The arrays consumed is information that is often available ahead of time (e.g. the user knows that its consuming application only reads pressure fields). Maestro could leverage this knowledge to optimise data flow (filtering) between producer and consumer applications.

Number	**R2.8**
Desirability	Must have
Use case(s)	**UC2.2**
Relates to	
Description	Maestro must provide the ability to persist data beyond the lifetime of the workflow that generated it, and make it available to appropriate consumers.
Justification	Required if Maestro is to replace an existing I/O stack.

Number	**R2.9**
Desirability	Should have
Use case(s)	**UC2.2**
Relates to	**R2.8**
Description	When used for persisting data produced by an application, Maestro should provide the ability to configure which data collection is persisted.
Justification	An application may produce different data collections serving different purposes. The user should be able to instruct Maestro which collection needs to be persisted.

Number	**R2.10**
Desirability	Should have
Use case(s)	**UC2.2**
Relates to	**R2.8**
Description	When used for persisting data produced by an application, Maestro should provide the ability to specify a subsampling rate of persistence or filter.
Justification	This allows different branches of a pipeline to work on different data flow rates. In particular, the main simulation branch executed “in transit” can work on a higher data flow rate to have refined data processing.

Number	**R2.11**
Desirability	Must have

Use case(s)	**UC2.3**
Relates to	
Description	When used for persisting checkpoint data, Maestro must provide the ability to specify the number of recent checkpoint records that will eventually be persisted.
Justification	This is required for spooling the checkpoint records and saving persistent storage space.

Number	**R2.12**
Desirability	Must have
Use case(s)	**UC2.5**
Relates to	
Description	When used for persisting data, Maestro must be able to detect rollback on a collection based on the record sequence index number and proceed with rewriting the persisted record history.
Justification	This is a required behaviour by our (CEA) simulation codes.

Number	**R2.13**
Desirability	Optional
Use case(s)	**UC2.5**
Relates to	
Description	When used for streaming data between consumers and producers, Maestro should be able to detect time step rollback from producer applications and clean up the staging/buffering area from discarded time steps that have not yet been consumed by consumer applications.
Justification	This is would save space in Maestro data staging/buffering area.

Number	**R3.1**
Desirability	Must have
Use case(s)	**UC3.1, UC3.2**
Relates to	
Description	Grant ownership of data to Maestro
Justification	Maestro needs to be able to take ownership of data to orchestrate data movement.

Number	**R3.2**
Desirability	Must have
Use case(s)	**UC3.1, UC3.2**
Relates to	
Description	Indicate Maestro to move or allocate data to a particular tier of memory
Justification	Explicit usage of GPU and Cuda kernel is very limiting. Maestro can abstract some of the details.

Number	**R3.3**
Desirability	Must have
Use case(s)	**UC3.1**
Relates to	
Description	Require data owned by Maestro
Justification	An application may need to use all or parts of data owned by Maestro even if it's not the original producer.

Number	**R3.4**
Desirability	Must have
Use case(s)	**UC3.1, UC3.2**
Relates to	
Description	Maestro is able to manage memory resources (including allocation and deallocation) across multiple layers of the memory hierarchy.
Justification	The multiplicity of memory and storage tiers, usually with a dedicated software stack, on modern HPC systems make them hard to use. A memory abstraction is necessary for Maestro to performance, usability and extendibility purposes.

Number	**R3.5**
Desirability	Should have
Use case(s)	**UC3.1**
Relates to	
Description	Data can be moved synchronously or asynchronously
Justification	Compute and communication should overlap for improved performance, but block where required for correctness.

Number	**R3.6**
Desirability	Must have

Use case(s)	**UC3.1**
Relates to	
Description	Maestro can move data from one memory space to another
Justification	Moving or copying data from one tier to another can be challenging given the different characteristics of the memory and storage technologies. An example could be to move data from a byte-addressable to a block-addressable memory.

Number	**R3.7**
Desirability	Must have
Use case(s)	**UC3.1**
Relates to	**R3.1**
Description	Maestro can manage different memory spaces within a node
Justification	As we may want to move data between two node-local memory tiers, each memory type has to expose its own memory space.

Number	**R3.8**
Desirability	Must have
Use case(s)	**UC3.1**
Relates to	**R3.1**
Description	Data dependencies are preserved within Maestro
Justification	A calling process or application usually must deal with multiple chunks of a dataset that show interdependencies. A simple illustration is a set of tasks that needs to be performed sequentially. It is necessary to preserve those dependencies to keep data coherency and properly schedule data movements.

Number	**R3.9**
Desirability	Should have
Use case(s)	**UC3.2**
Relates to	**R3.1**
Description	Maestro should manage (re)use of the memory space with minimal overhead.
Justification	Allocation and deallocation of memory spaces can significantly decrease I/O performance. Memory space reuse can reduce this.

Number	**R4.1**
Desirability	Must have
Use case(s)	**UC4.1**
Relates to	

Description	Redistribution of parallel 2D fields
Justification	The 2D data has to be redistributed according to the distribution of the rows of the interpolation matrix (built from SCRIP) among processes.

Number	**R4.2**
Desirability	Should have
Use case(s)	**UC4.1**
Relates to	
Description	Row and column-major layout transformations of distributed arrays
Justification	Climate models in TSMP are written in C as well as Fortran, arrays exchanged between them often have to be rearranged.

Number	**R4.3**
Desirability	Optional
Use case(s)	**UC4.1**
Relates to	
Description	Interpolation of distributed arrays
Justification	Fields in TerrSysMP can have different grid sizes and can be distributed on different number of processes. A matrix representing the interpolation weights is either generated or read from a SCRIP file and used to map the distributed source array to the distributed target array. The SCRIP bilinear interpolation operator is used for COSMO variables and the SCRIP distance-weighted averaging operator for CLM variables. An MCT datatype describes the coupled system processor layout. MCT stores coupling field data in an object that supports arbitrary numbers of real- and integer-valued fields, indexed using string tokens. A domain decomposition descriptor (DDD) object uses a 1-D global index space (e.a. linearization) to represent multidimensional index spaces. Parallel communication schedules are computed automatically from source and destination DDDs. Parallel data transfer is accomplished by calling paired send/receive methods with data storage and communication schedule datatypes as inputs. MCT provides distributed storage for precomputed interpolation coefficients from which it derives communication schedules for parallel interpolation. This operation can optionally be constrained to give identical results on different numbers of processes.

Number	**R4.4**
Desirability	Must have
Use case(s)	**UC4.2**
Relates to	

Description	Spawn and terminate processes within the same job allocation
Justification	To redistribute TerrSysMP’s models over different processes, Maestro has to be able to spawn and terminate processes multiple times during a single job allocation.

5. Concluding remarks

This document presents a set of requirements for the design of the Maestro middleware, and the use cases and usage scenarios within which they have been identified. These requirements represent a broad range of data use within HPC systems and a range of challenges for a memory- and data-aware middleware to solve. These requirements have fed, and will continue to feed, into the design process for the Maestro middleware to ensure that it constitutes a real step forward in data use in HPC systems.

It will not be possible to satisfy all possible requirements collected from the applications within the project. The requirements capture process lets us evaluate the significance of each requirement and synthesise a coherent middleware design taking into account the revealed priorities.

This deliverable collects a baseline set of requirements. As the project progresses, this list will be updated and refined as appropriate. Updated requirements will be published as part of D2.3.