

## IS-ENES3 Deliverable D8.3

### WP8/Task4: XIOS development

*Reporting period: 01/01/2022 – 31/03/2022*

Author: Yann Meurdesoif (CNRS-IPSL), Marie-Pierre Moine (CERFACS),  
Gaëlle Rigoudy (Meteo-France/CNRM)

Reviewer(s): Jean-Christophe Rioual (MetOffice), Sophie Valcke (CERFACS)

Release date: 21/10/2022

D8.3 “XIOS development” describes the work achieved within IS-ENES3 on the rewriting of the XIOS server, as well as on the extension of the Dr2xml tool which automatically writes the XML XIOS configuration files. Since the two codes are independent from each other, the deliverable is presented in two separate parts:

- **PART A:** XIOS new release ([p. 2](#))
- **PART B:** Extension of the Dr2xml package ([p. 36](#))



# IS-ENES3 Deliverable D8.3 – Part 1

## WP8/Task4: XIOS development – PART A: XIOS new release

*Reporting period: 01/01/2022 – 31/03/2022*

Author: Yann Meurdesoif

Reviewer(s): Jean-Christophe Rioual (MetOffice), Sophie Valcke (CERFACS)

Release date: 18/10/2022

The work presented here forms the first part of the IS-ENES3 deliverable D8.3: "XIOS development" describing the rewriting of the XIOS server itself, while the second part deals with the extension of the Dr2xml tool developed to automatically write the XML XIOS configuration files.

The main elements of the new version of the server, XIOS-3.0, resulting from 3 years of rewriting, are described in details. These developments lead to: improved code readability, reliability and evolutivity; new client/server protocols for increased performance; rationalisation and improvement of the grid distribution for lower memory consumption; and a new HPC service-oriented infrastructure opening the door to future climate modelling, ensuring improved support of large ensembles, the arrival of neural networks and artificial intelligence methods. This deliverable is linked with the release of a pre-stable beta version available at <http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS3/stable/xios3.0-beta>, which can now be considered as the reference version for models testing. After being validated in production for several ESMs, it will be tagged as a stable "XIOS-3.0" version, targeted for end of 2022.

| Dissemination Level |   |   |
|---------------------|---|---|
| PU                  | Public  | X |
| CO                  | Confidential, only for the partners of the IS-ENES3 project |   |

| Revision table |            |                 |  |
|----------------|------------|-----------------|--|
| Version        | Date       | Name            | Comments                                   |
| 0.1            | 29/07/2022 | Yann Meurdesoif | Initial version submitted to the reviewer. |
| 0.2            | 05/10/2022 | Yann Meurdesoif | Revisions to answer the reviewers remarks  |
|                |            |                 |  |

## Table of contents

|   |    |
|---|----|
| 1. Introduction .....   | 5  |
| 1.1 Review of 10 years of XIOS development .....                            | 5  |
| 1.2 Lessons learned from the CMIP6 exercise .....                           | 8  |
| 2. Deliverable D8.3, IS-ENES3 WP8 .....                                     | 9  |
| 2.1 The original deliverable .....  | 9  |
| 2.2 The final deliverable .....   | 10 |
| 3. Details of the developments .....  | 12 |
| 3.1 Robustness and reliability .....  | 12 |
| 3.2 Development of new transfer protocols .....                             | 17 |
| 3.3 New concepts to rationalise and improve the distribution of grids ..... | 22 |
| 3.4 Development of a new internal HPC service-oriented infrastructure ..... | 27 |
| 4. Validation .....   | 34 |
| 5. Discussion and conclusion .....  | 34 |

## Executive Summary

The work presented here forms the first part of the IS-ENES3 deliverable D8.3:"XIOS development" describing the rewriting of the XIOS server itself, while the second part deals with the extension of the Dr2xml tool developed to automatically write the XML XIOS configuration files. The two parts of this deliverable were carried out separately because they are independent each other and are therefore presented in two separate documents.

In the current document, the main elements of the new version of the server, XIOS-3.0, resulting from 3 years of rewriting, are described in details. The previous version, XIOS-2.5, constituted of 110,000 lines of code in C++ was, and still is, heavily used with success by a large community in France as well as in Europe through the NEMO consortium but also at the Met-Office and at the Barcelona Supercomputing Centre, in particular for the CMIP6 exercise. However, that version, resulting from nearly 10 years of diverse developments, showed specific problems in terms of performance, memory use and upgradability. This is why time was dedicated to review the whole code instead of adding new functionalities, as originally planned in IS-ENES3. We decided to focus, on the one hand, on robustness, performance improvement and memory footprint reduction and, on the other hand, on the development of a new more general "in-situ" service infrastructure unifying I/O and model coupling, more adequate for the future evolution of climate simulations and supercomputing. These developments result in: improved code readability, reliability and evolutivity; new client/server protocols for increased performance; rationalisation and improvement of the grid distribution for lower memory consumption; and a new HPC service-oriented infrastructure opening the door to future climate modelling, ensuring improved support of large ensembles, the arrival of neural networks and artificial intelligence methods.

Currently the short-term priority is the stabilization of these functionalities. This deliverable is therefore linked with the release of a pre-stable beta version available at <http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS3/stable/xios3.0-beta>, which can now be considered as the reference version for models testing. After being validated in production for several ESMs, it will be tagged as a stable "XIOS-3.0" version, targeted for end of 2022.

The developments described in the initial deliverable, which have been postponed because considered to be of lower priority than the code refactoring, will be taken up and finalised before the end of the IS-ENES3 project.

## 1. Introduction

### 1.1 Review of 10 years of XIOS development

Given the regular increase of the output of climate models that we are asked to produce, in particular during CMIP exercises, in terms of volume (resolution), number and complexity (workflow), we realised very early at IPSL that this would be one of the main bottlenecks for our future simulations. At the beginning of 2009, in a first prototype written in Fortran (XML-IO-SERVER), we laid the foundations and fundamental ideas on which future versions of XIOS would be built. We wanted to relieve the models of any internal description of the outputs in order to simplify and mutualise the coding in Fortran. Instead, we exposed the data to be written through a minimalist interface in which is used only the address of the variable to be output, associated with an ASCII identifier:

```
CALL xios_send_field("field_id", field).
```

The description of how the variable will be output, on which mesh, in which file, at what frequency, with what time integration, etc., is externalised in an XML file that controls all the model outputs. As this file is analysed at runtime, it is possible to modify the model outputs without changing the model Fortran sources and therefore without having to recompile, which provides great flexibility of use. Furthermore, as writing files in "NETCDF" format to disk takes up a significant amount of computing time and affects performance, we have implemented a client-server approach in which the clients (models) send their data to be written completely asynchronously (via the MPI library) to the servers, which are then responsible for writing the data (see Figure 1). This approach allows both the sending of data to the servers and the writing of data to disk to be covered by the clients' computation and thus, theoretically, reduces the output cost to zero.

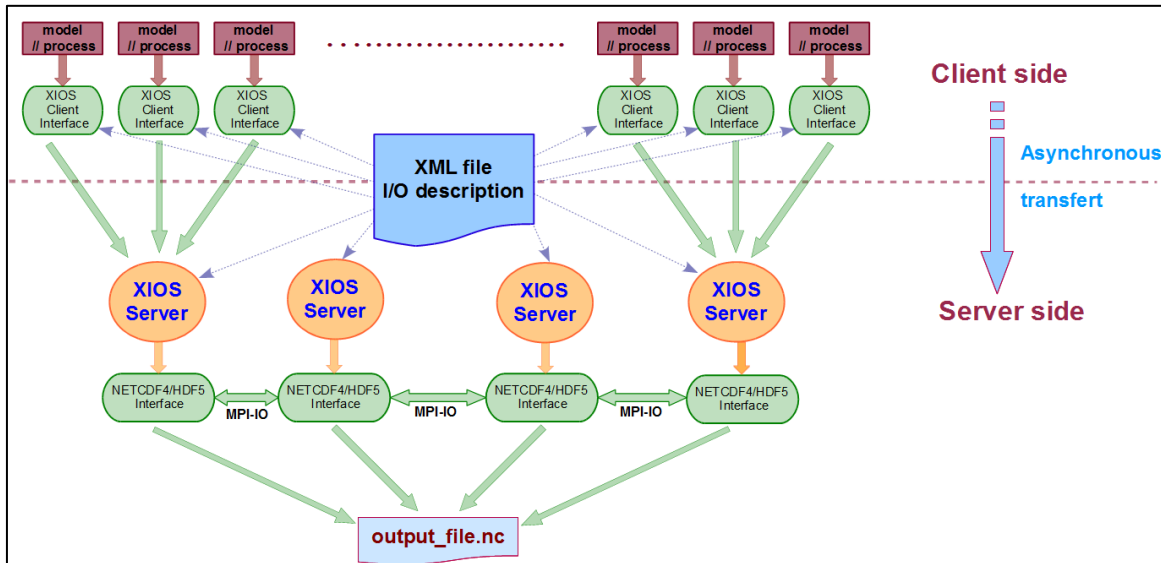


FIGURE 1 XML-IO-SERVER SCHEMA

The flexibility of this approach quickly seduced the NEMO consortium who was the first to adopt it. However, for practical reasons, the temporal integration (averaging) was still done on the server side and required sending the model data at each time step, which posed a scaling problem. Subsequently, with the help of dedicated resources from the IS-ENES-1 project (12 PMs), we decided to completely recode the tool in C++, a language much more suitable for this kind of tool, to benefit from the power of object-oriented programming and its standard library (STL). During this re-coding, the time integration was switched to the client side, which greatly improved scalability on supercomputers, making it a truly efficient tool. This transition also resulted in a contraction of the name "XML-IO-SERVER" to "XIOS" in early 2012. This first version of XIOS was then quickly integrated into the various IPSL models: NEMO (ocean), LMDZ (atmosphere), ORCHIDEE (land use) and INCA (atmospheric chemistry). An interface with the OASIS coupler was built and subsequently allowed XIOS to be used in the framework of IPSL coupled Earth System Model (ESM).

In the following years, the development was continued through the European projects IS-ENES-2 and ESiWACE-1. Inputs were added following the same philosophy as output, via the interface: `xios_rcv_field("field_id", field)`, as well as support for unstructured and Gaussian-reduced grids. New functionalities have gradually been introduced to act on the fields processed by XIOS. These transformations can be divided into 3 main categories:

- Temporal integration : allowing the accumulation or averaging of fields over time.
- Arithmetic operations: combination of one or more fields of the same mesh from any arithmetic expression and/or functions.
- Spatial transformation: allows a field to be transformed from one mesh to another through a specific transformation rule: extraction, zooming, reduction, reordering, interpolations, etc. (e.g. : Figure 2).

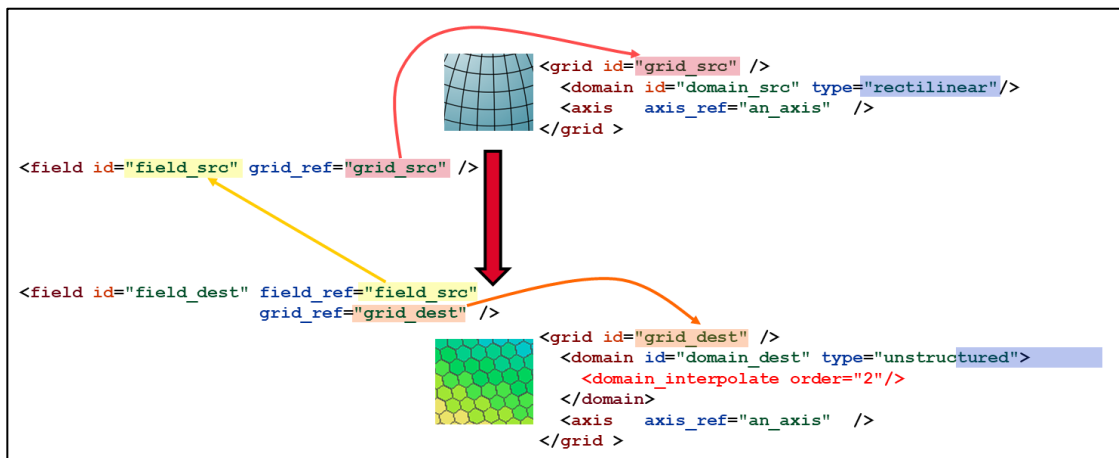


FIGURE 2 : EXAMPLE OF THE IMPLEMENTATION OF A SPATIAL INTERPOLATION FROM A CARTESIAN GRID TO AN UNSTRUCTURED GRID

These transformations are translated internally into filters which can be chained to lead to the execution of a workflow which calculates "in-situ" diagnostics, sometimes costly, before writing them to disk. As the workflow is executed in parallel on all the simulation cores throughout the execution, it allows scaling and greatly simplifies or even renders unnecessary the management of post-simulation processing (Figure 3).

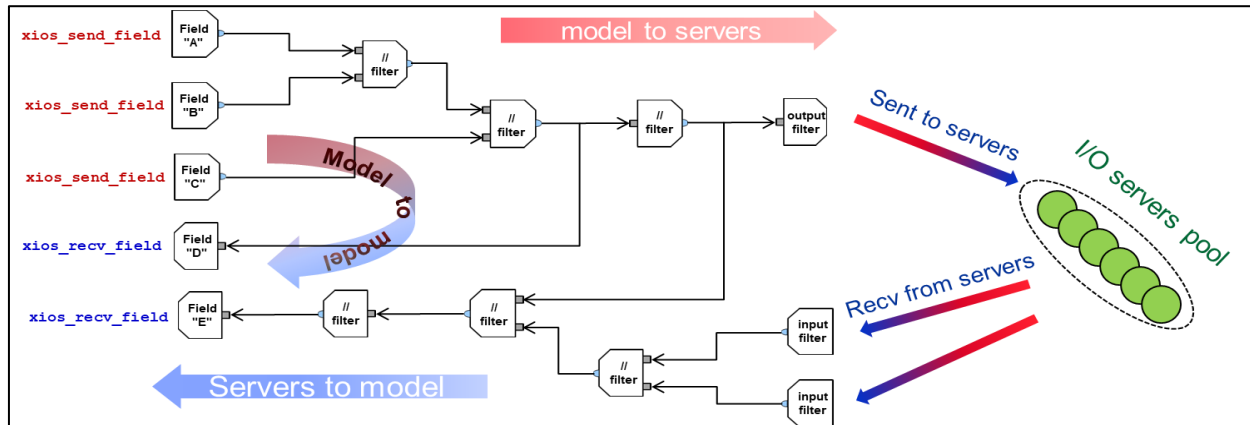


FIGURE 3 : DIAGRAM SHOWING THE CHAINING OF DIFFERENT FILTERS TO ACHIEVE A COMPLEX WORKFLOW, EITHER AS OUTPUT OR INPUT.

Note that the XIOS workflow is “reentrant”, which means that a series of filters can be applied to a field without necessarily calling on the servers, and then returned to the model after transformation. These features are present in the stable version of XIOS-2, available from June 2018 under SVN: <http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS2/branches/xios-2.0>.

The development continued through the preparation of the CMIP6 exercise with the objective of completely eliminating the post-processing phase, therefore distributing the data on the ESGF grid as they are produced at the end of the simulations. CNRM and IPSL pooled their resources to develop a common XIOS workflow. On this occasion, CNRM ESM was adapted so that the model outputs could be managed by XIOS (S. Sénési). CNRM has developed the "DR2XML" tool, allowing the automatic translation of the CMIP6 "Data Request" into XML that can be understood by XIOS; recent developments on DR2XML form PART B of this deliverable. For its part, IPSL developed the functionalities that XIOS lacked to address the CMIP6 workflow. This shared work was successful, as CNRM IPSL were the first groups to deposit their CMIP data on the ESGF grid, and to date, IPSL is the group that has published the most CMIP6 data in number and volume. This validated the strategy employed of performing post-processing "in-situ" and demonstrated the capacity of XIOS to handle extremely large and complex workflows. As an illustration, for a single CMIP6 deck, DR2XML generates almost 90,000 lines of XML to describe the associated XIOS workflow. Without this automation, the post-processing of the data would have required an enormous amount of human resources and computing time. The version used for the CMIP6

exercise is available as a stable XIOS-2.5 branch under SVN: <http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS2/branches/xios-2.5>.

At that stage, XIOS comprised nearly 110,000 lines of code in C++ (and Fortran for the interface with the models) (Figure 4) developed over nearly 10 years, which makes it, in size, the equivalent of a component of an ESM model. For comparison, NEMO 3.6 is also approaching 110,000 lines of Fortran code.

|   |                       |                           |
|---|-----------------------|---------------------------|
| SLOC  | Directory             | SLOC-by-Language (Sorted) |
| 97689   | XIOS/src              | cpp=66200, f90=31487      |
| 9204  | XIOS/extern/remap/src | cpp=9204                  |
| Totals grouped by language (dominant language first):     |                       |                           |
| cpp:  | 75404 (70.54%)        |                           |
| f90:  | 31487 (29.46%)        |                           |
| Total Physical Source Lines of Code (SLOC)                |                       | = 106,893                 |
| Development Effort Estimate, Person-Years (Person-Months) |                       | = 27.00 (324.05)          |
| Total Estimated Cost to Develop                           |                       | = \$ 3,647,862            |

FIGURE 4 : SLOCCOUNT TOOL REPORT ON XIOS-2 SOURCES (TRUNK)

XIOS is used by a large community in France (IPSL, CNRM, IFREMER, LGGE, MERCATOR) as well as in Europe through the NEMO consortium but also at the Met-Office (LFRIC) and at BSC (NEMO+OpenIFS).

## 1.2 Lessons learned from the CMIP6 exercise

The development of XIOS from 2010 to 2020 relied mainly on external resources from various French (Convergence) and European (IS-ENES1, IS-ENES2, ESIWACE1) projects. These resources represented the full-time equivalent of one engineer over the period, compared to the internal resources made available by IPSL, representing approximately 1/3 of a full-time position. This imbalance, as well as the high turnover of non-permanent engineers leaving us taking away their acquired know-how and experience, has resulted over the years in increasingly intricate developments and a deterioration in the readability of the code and its infrastructure. The deadlines imposed by the CMIP6 exercise led to emergency developments to fill in missing functionalities or certain bugs. Due to the lack of robustness and automated testing procedures, these new corrections led to regressions that were not detected until much later. In addition, the departure of engineers who came to the end of their contract at the beginning of the exercise and who had largely contributed to the development of the code, left us helpless in the face of the urgency of the situation. We realised that for the long-term sustainability of the project, it was necessary to reinforce the share of permanent engineers working on the project. Nevertheless, despite these difficulties, the production of the CMIP6 exercise data by XIOS was a success. But this was at the expense of several considerations:

- Loss of control over the code: overly intricate and urgent developments, lack of permanent staff to control over the developments made during the projects, weakness of continuous integration and lack of robustness. These difficulties considerably slow down new developments of the code, making them less efficient and more fragile. It is necessary to start again on a healthier basis for future developments.



- Loss of performance: during the exercise we focused primarily on functionality and less on performance. The complexity of the workflow in the coupled model led us to make some compromises on performance. The overhead of the standard CMIP6 workflow is in the order of 10%-15% but can rise to 40% for heavier experiments. Even if these times remain reasonable for our model resolutions (200-100km), we note that they increase as the resolution increases and that scaling up is difficult. With this workflow complexity the maximum acceptable resolution is around 25km global. However, with simpler outputs, simulations with global NEMO at 6km (C. Taillandier, IMMEDIAT project) have shown that it was possible to use XIOS at these resolutions while covering writing and calculation, but at the cost of an in-depth analysis of performance and configuration tuning. With the increase in model resolution and the arrival of GPU computing, it has become urgent to refocus on performance aspects.
- Memory overuse: the XIOS workflow was extremely memory-consuming. Until then, given the low resolutions of our global models, these aspects were not a priority at IPSL. But when scaling up to other communities, this problem became prohibitive and it became necessary to depopulate the nodes of the XIOS servers or to cleverly mix the different processes in order to pool the available memory. This is also one of the critical points on which it was necessary to act.

In view of these observations, IPSL has decided to consolidate the development of XIOS by the arrival of a permanent engineer at the beginning of 2021. Moreover, as these problems were not anticipated when the IS-ENES3-WP8 D8.3 deliverable was proposed more than 5 years ago, we have decided to redefine the developments that would be carried out to produce the new version of XIOS (XIOS-3).

## 2. Deliverable D8.3, IS-ENES3 WP8

The original deliverable was focused on adding a number of features. We decided to push back some of these objectives to focus on robustness, performance improvement and memory footprint reduction on the one hand, and on the other hand on the development of a new infrastructure unifying I/O and model coupling as well as its extension to a more general "in-situ" service infrastructure, more adequate to the future evolution of the use of simulations and supercomputing.

### 2.1 The original deliverable

In the original deliverable, we proposed several developments for this future version of XIOS, some of which have been included in the final deliverable and others pushed back to the year 2023:

- Robustness and reliability; implementation of a continuous integration suite => included and greatly enhanced.
- Model coupling => included, and extended to a more general "in-situ" service-oriented infrastructure.
- Addition of new spatial transformations => postponed
- Restartability added => postponed

- Temporal interpolation => postponed

## 2.2 The final deliverable

The main objective of this deliverable is to regain control over the development of the sources, to considerably improve the reliability and robustness aspects well beyond the initial proposal, to address the performance and memory consumption problems, and to redevelop a new internal code infrastructure for more evolution. Given the extent of the infrastructure/performance/robustness entanglements, it was decided to rewrite the XIOS internal computation engine almost completely to arrive at a new coherent, fully mastered and easily extensible code structure that will now allow us to accelerate future developments with greater reliability. The work to achieve this deliverable was considerable and spread over more than 3 years (beginning of March 2019). During this period, nearly 514 files were modified through 244 SVN commits, which represented changes to nearly 60,000 lines of code (see Figure 5), i.e. the major part of the internal XIOS core.

```
irene171 work~/XIOS3/src>svn diff -r 1749 | diffstat -m -s
514 files changed, 36824 insertions(+), 19441 deletions(-), 5209 modifications(!)
```

FIGURE 5 : RESULT OF THE STATISTICS OF THE "SVN DIFF" COMMAND SINCE THE BEGINNING OF THE DEVELOPMENT OF VERSION 3

Several branches of development followed one another to reach this version, which can be followed under SVN:

```
XIOS2/trunk          =>          XIOS2/dev/dev_ym/XIOS_ONE_SIDED          =>
XIOS2/dev/dev_ym/XIOS_SERVICES  =>  XIOS2/dev/dev_ym /XIOS_COUPLING  =>
XIOS3/trunk
```

The new version is available on the IPSL server:

<http://forge.ipsl.jussieu.fr/iomserver/browser/XIOS3/trunk>.

This branch, although functional, is not yet fully stabilised (trunk branch), as at this stage it has only been validated on the IPSL ESM model and needs to be tested by other models. A stable branch will then be created before the end of the year 2022, but for now a pre-stable beta branch is available and functional for testing:

- <http://forge.ipsl.jussieu.fr/iomserver/browser/XIOS3/stable/xios3.0-beta>.

For this update, for more consistency and clarity in relation to other XIOS branches and versions, we have completely reorganised the XIOS SVN repository (Figure 6).



FIGURE 6 : NEW XIOS SVN REPOSITORY TREE

This new update contains many developments that are difficult to describe in detail. Nevertheless, the following is a summary of the main points that have been addressed and will be discussed in more detail later in this document.

- Robustness and reliability
  - Implementation of a suite of non-regression test cases for continuous integration: at each SVN commit, automatic test of the compilation on several machines and compilers. Automatic execution of a series of test cases to detect possible regressions. Results can be viewed through a web browser (IS-ENES3 milestone M8.1).
  - Representation of XIOS workflow execution in the form of graphs, viewable through a web browser: help in setting up the workflow and debugging in the event of errors.
  - Output of the workflow software stack in case of a crash: debugging aid.
  - Additional internal output timers at the end of the simulation to refine the performance profiling of the workflow.
  - Internal tracking of the memory used by the XIOS workflow to diagnose memory leaks and overall consumption.
- Development of new client/server transfer protocols by adding passive "one-sided" MPI communication: protocol fluidity, performance improvement.
- Addition of new concepts of 'views' and 'connectors' for distributed management of workflow grids
  - High reusability of developed classes, more clarity and flexibility, fewer lines of code
  - Improved performance thanks to centralization of workflow computation in connector transfer functions
  - Reduction of the memory footprint by applying tensor product properties to the different elements (domain, axis, scalar) making up a grid

- Rewriting of flux transfer engine between model <-> client and client<->server, spatial transformations and disk reads/writes (via netcdf) using these new concepts.
- Rewriting of the engine allowing to chain spatial transformations: more readability, more robust and better-defined workflow construction, suppression of redundant calculations
- Development of a new internal HPC service-oriented infrastructure
  - When a simulation is launched, part of the resources (processes) is allocated for XIOS services. The services are started dynamically on the free resources according to the requirements specified in the XML configuration files.
  - The functionalities of version 2.5, integrating level 1 and level 2 servers, have been recoded in terms of services: gatherers service (level 1), writers service (level 2, writing), readers service (level 2, reading).
  - Models are technically seen by XIOS as services that provide or receive data.
  - XIOS handles the interconnection of data flows between the various services using the parallel and asynchronous transfer protocol. Transfers can thus be chained from service to service.
  - Flexibility in service management: each XIOS context, linked to a model, can have its own dedicated I/O service, whereas previously it was common to all models.
  - Allows model coupling: streams are exchanged via the XIOS middleware and remapping from the source grid to the destination grid is performed through the workflow (horizontal and/or vertical interpolation filters)
  - Prospectives: opens the way to the development of new services:
    - Offload service (in progress): costly diagnostics, previously performed by the client workflow, can be offloaded to dedicated servers and performed asynchronously before sending the data to be written to the 'writer' service.
    - Ensemble service: service dedicated to the management of ensemble simulation data: statistical reduction of data "in-situ" and writing.
    - AI service: Integrate AI learning "in situ" from data produced by the models. Symmetrically integrate "in situ" inference from trained neural networks.
    - ...

### 3. Details of the developments

#### 3.1 Robustness and reliability

##### a) Unit test case suite

This suite of unit test cases has been developed to detect regressions when the XIOS sources are modified. This development has already been the subject of a milestone document (M8.1). This suite is automatically launched after each SVN commit of the XIOS sources. It consists of testing a specific XIOS functionality over a whole set of parameters: type of model to be emulated, number of clients, number of servers, duration, type of grid, 2D grid, 3D grid, masked or not, etc. We have developed a single executable (generic test case) allowing all these aspects to be emulated according to 2 parameter files (param.def and iodef.xml).

| Param.def  |  | iodef.xml  |
|--|--|--|
| <pre>&amp;params_run duration='1d' nb_proc_atm=10 nb_proc_oce=5 nb_proc_surf=1 /</pre> |  | <pre>&lt;context id="atm"&gt;   &lt;variable_definition&gt;      &lt;variable id="timestep"&gt; 1h &lt;/variable&gt;     &lt;variable id="domain"&gt; lmdz &lt;/variable&gt;     &lt;variable id="domain_mask"&gt; true &lt;/variable&gt;     &lt;variable id="axis_mask"&gt; false &lt;/variable&gt;     &lt;variable id="init_field2D"&gt; academic &lt;/variable&gt;     &lt;variable id="ni"&gt; 36 &lt;/variable&gt;     &lt;variable id="nj"&gt; 18 &lt;/variable&gt;     &lt;variable id="nlev"&gt; 10 &lt;/variable&gt;     &lt;variable id="pressure_factor"&gt; 0.10 &lt;/variable&gt;     &lt;variable id="mask3d"&gt; false &lt;/variable&gt;     &lt;variable id="domain_proc_frac"&gt;3&lt;/variable&gt;     &lt;variable id="axis_proc_frac"&gt;2&lt;/variable&gt;     &lt;variable id="axis_proc_n"&gt;2&lt;/variable&gt;     &lt;variable id="ensemble_proc_n"&gt;2&lt;/variable&gt;    &lt;/variable_definition&gt; &lt;/context&gt;</pre> |

FIGURE 7: EXAMPLE OF A PARAMETER FILE FOR THE GENERIC TEST CASE

The functionality we wish to test is then described in XML through a piece of XIOS workflow: e.g. disk output, spatial transformations, etc. It is therefore possible to test this functionality in an automated way according to a set of parameters provided by the two files. This makes it very easy to add a new feature to be tested.

The execution of the unit test case chain proceeds as follows:

- For each commit, it first tests the compilation of the sources on several compilers (intel, pgi, gnu, llvm), with different optimisation modes (prod, debug) and on several machines (Jean-Zay and Irene Tiers1 supercomputers). The results are displayed in a synthetic way on a web page of the XIOS website:

[https://forge.ipsl.jussieu.fr/ioserver/chrome/site/XIOS\\_TEST\\_SUITE/xios\\_report.html](https://forge.ipsl.jussieu.fr/ioserver/chrome/site/XIOS_TEST_SUITE/xios_report.html)

Choose a revision number to show compile and test results :

All available revisions ...





-  : compile failed / test failed
-  : test result initialized
-  : compile passed / test passed
-  : no NetCDF files generated

Table of XIOS Compile status









| Revision | Jean-Zay        |   | Irene          |   |   |
|----------|-----------------|---|----------------|---|---|
| 2384     |                 | prod  |                | prod  | debug   |
|          | X64_JEANZAY_PGI |  | X64_IRENE      |  |  |
|          | X64_JEANZAY     |  | X64_IRENE_GNU  |  |  |
|          |                 |   | X64_IRENE_LLVM |  |  |

FIGURE 8: CONTINUOUS INTEGRATION SUITE BUILD REPORT, DISPLAYED ON THE XIOS WEBSITE

- Then, for each set of parameters, the non-regression tests are executed automatically on both supercomputers. Comparisons with reference outputs indicate whether the test is successful or not. More than a hundred unit test executions are thus carried out at each launch of the suite. The results are then presented in a synthetic manner with a drop-down menu on the same web page as the compilation.

| Table of XIOS unit tests results |                  |                   |  |  |  |  |  |  |  |
|----------------------------------|------------------|-------------------|--|--|--|--|--|--|--|
| Revision                         | Jean-Zay         | Irene             |  |  |  |  |  |  |  |
| 2304                             | X64_IJANZAY_PROD | X64_IJANZAY_PROD  |  |  |  |  |  |  |  |
|                                  |                  | test_scalar_algo  |  |  |  |  |  |  |  |
|                                  |                  | test_grid_algo    |  |  |  |  |  |  |  |
|                                  |                  | test_dynamic_algo |  |  |  |  |  |  |  |
|                                  |                  | test_nemo_algo    |  |  |  |  |  |  |  |
|                                  |                  | test_domain_algo  |  |  |  |  |  |  |  |
|                                  |                  | test_function     |  |  |  |  |  |  |  |
|                                  |                  | test_axis_algo    |  |  |  |  |  |  |  |
|                                  |                  | test_memory       |  |  |  |  |  |  |  |
|                                  |                  | X64_IJANZAY_PROD  |  |  |  |  |  |  |  |
|                                  |                  | X64_IJANZAY_PROD  |  |  |  |  |  |  |  |
|                                  |                  | X64_IJANZAY_PROD  |  |  |  |  |  |  |  |
|                                  |                  | X64_IJANZAY_PROD  |  |  |  |  |  |  |  |
|                                  |                  | X64_IJANZAY_PROD  |  |  |  |  |  |  |  |

FIGURE 9: EXTRACT FROM THE CONTINUOUS INTEGRATION SUITE EXECUTION REPORT AS DISPLAYED ON THE XIOS WEBSITE

This continuous integration suite, which is easily extensible, has helped us considerably to improve the robustness of XIOS developments by enabling us to detect very early the introduction of new bugs that lead to regressions.

#### b) Graphical representation of the XIOS workflow

The development of a complex XIOS workflow in XML is a delicate task and source of many errors. Faced with the problems encountered, users are often lost, due to the lack of tools for effective debugging. In particular, errors when combining temporal data streams (tagged with a specific date) can lead to "dead-locks", with the workflow waiting for a stream that will never be sent. In order to detect these situations, we have developed a tool that allows the workflow to be visualised graphically, displaying the data flows entering the workflow, their passage through the various filters, whether temporal, arithmetic or spatial, until they leave the workflow, i.e. are sent to the servers for writing. The implementation is very simple; in the XML configuration file, it is a matter of adding the attribute "build\_workflow\_graph="true"" to a field which trace we want to follow. All dependencies, whether upstream or downstream of the field, will also be taken into account.

```
<file id="atm_output" output_freq="4ts" type="one_file" enabled="true">
  <field field_ref="field3D" name="field_interp" grid_ref="grid3d_interp" build_workflow_graph="true" operation="average" />
</file>
```

FIGURE 10: THE "BUILD\_WORKFLOW\_GRAPH" ATTRIBUTE IS USED TO ACTIVATE THE GRAPHICAL OUTPUT OF THE XIOS WORKFLOW ON THE RELEVANT FIELD

To avoid generating too many graphs, it is possible to limit the tracing period using the attributes: "build\_start\_graph" and "build\_end\_graph". A file in "json" format is then generated post-mortem, which can be viewed through any web browser thanks to an application that we have developed in javascript:

[https://forge.ipsl.jussieu.fr/ioserver/chrome/site/XIOS\\_TEST\\_SUITE/graph.html](https://forge.ipsl.jussieu.fr/ioserver/chrome/site/XIOS_TEST_SUITE/graph.html).

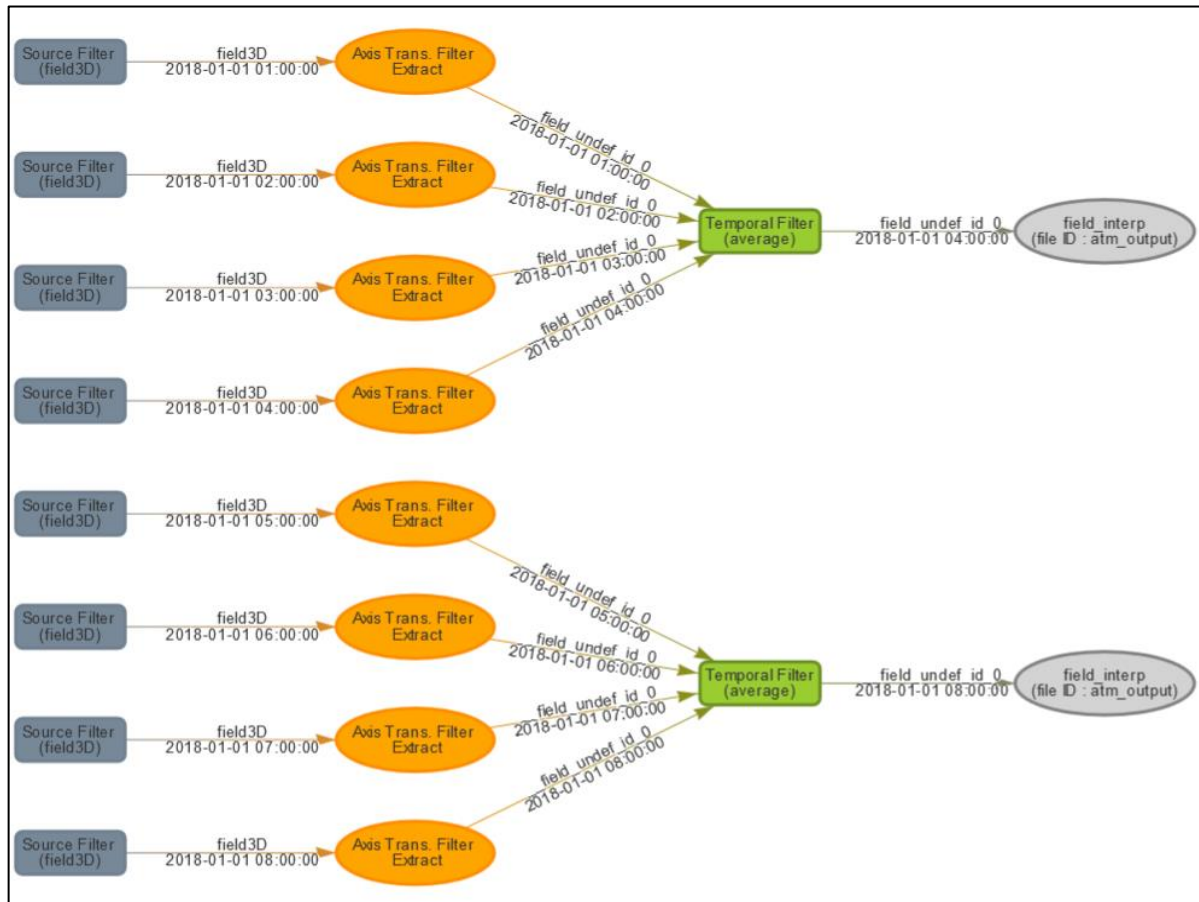


FIGURE 11 : EXAMPLE OF THE XIOS WORKFLOW VISUALISATION.

In the example of Figure 11, we see the graphic representation of the flow of the "field3D" field, which is taken from the model at each time step, to which a spatial transformation ("Axis Trans. Filetr Extract") is applied and then a temporal average before being output in the "atm\_output.nc" file every 4 time steps. This very easy to use tool saves many hours of work when setting up a complex workflow, especially for less experienced users.

### c) Improved error diagnostics

When running the XIOS workflow, errors are usually detected and an error message is generated by the routine that raised the exception. But often, although the error message is clear and the routine is well identified (name and line number in the code), it is not indicated in which context it applies because it is necessary to go back up the call stack to find this information. To help users to debug their workflow more efficiently, we have developed a new exception handler capable of going up the call stack and displaying various information (class names, object identifiers, related attributes, line numbers, etc.) to better identify the parts of the workflow that are affected by the



error generated. These new diagnostics are automatically output when an error is generated. They allow users to better target the origin of the errors generated, and thus make the link with the construction of the workflow in XML.

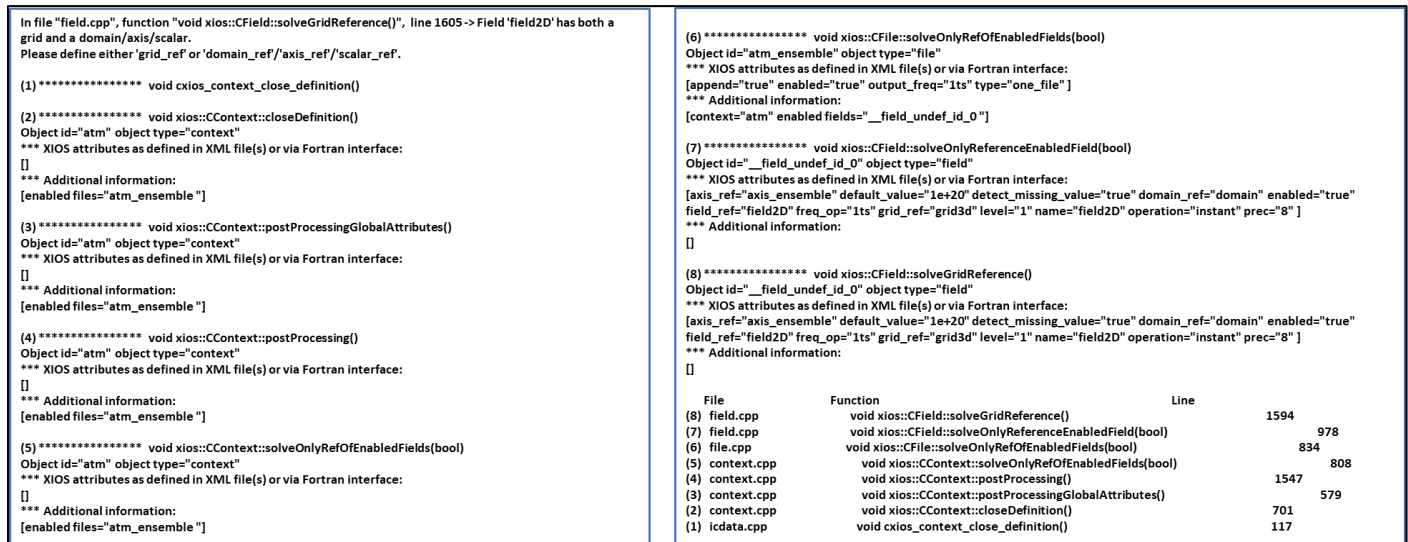


FIGURE 12: EXAMPLE OF HOW THE XIOS RUNTIME STACK IS DISPLAYED IN THE EVENT OF A CRASH.

#### d) Improved code profiling

To help optimise the workflow, determine the optimum number of level 1 and level 2 servers and their placement on the compute nodes in relation to the clients, it is necessary to have as accurate a performance report as possible to be able to determine how much time is spent in each part of the code. We have implemented C++ classes (CTimer) that allow us to measure the time spent in a region of code in a very simple way:

CTimer::get("timer\_id").resume(); as section input

CTimer::get("timer\_id").suspend(); in section output

These timers are placed at key points in the code in order to obtain a very fine performance profile. At the end of the execution, if the requested output level is sufficient, all timers are automatically exited (Figure 13).

In addition, the functionality of these timers has been made available to models through the Fortran interface and can be used to perform model profiling. It is used in Fortran in a very similar way to C++ :

CALL xios\_timer\_resume("timer\_id") at section entry.



CALL `xios_timer_suspend("timer_id")` at section output.

|  |   |
|--|---|
| <pre> -&gt; info : CContextServer: Receive context &lt;atm&gt; finalize. -&gt; report : Memory report : Context &lt;atm&gt; : server side : memory used for buffer of each connection to client +) With client of rank 0 : 10000000 bytes -&gt; report : Memory report : Context &lt;atm&gt; : server side : total memory used for buffer 10000000 bytes -&gt; report : Memory report : Context &lt;atm_server&gt; : client side : memory used for buffer of each connection to server +) To server with rank 0 : 10000000 bytes -&gt; report : Memory report : Context &lt;atm_server&gt; : client side : total memory used for buffer 10000000 bytes -&gt; info : Closing File : atm_ensemble -&gt; info : CContext: Context &lt;atm_server&gt; is finalized. -&gt; report : Memory report : Context &lt;atm_server&gt; : server side : memory used for buffer of each connection to client +) With client of rank 0 : 10000000 bytes -&gt; report : Memory report : Context &lt;atm_server&gt; : server side : total memory used for buffer 10000000 bytes -&gt; report : Memory report : Context &lt;atm&gt; : client side : memory used for buffer of each connection to server +) To server with rank 0 : 10000000 bytes -&gt; report : Memory report : Context &lt;atm&gt; : client side : total memory used for buffer 10000000 bytes -&gt; info : CContext: Context &lt;atm&gt; is finalized. -&gt; info : Client side context is finalized -&gt; report : Performance report : Whole time from XIOS init and finalize: 0.359765 s -&gt; report : Performance report : total time spent for XIOS : 0.327634 s -&gt; report : Performance report : time spent for waiting free buffer : 9.28231e-05 s -&gt; report : Performance report : Ratio : 0.025801 % -&gt; report : Performance report : This ratio must be close to zero. Otherwise it may be useful to increase buffer size or numbers of server -&gt; report : Memory report : Minimum buffer size required : 5267 bytes -&gt; report : Memory report : increasing it by a factor will increase performance, depending of the volume of data wrote in file at each time step of the file </pre> | <pre> -&gt; report : Timer : Blocking time --&gt; cumulated time : 9.28231e-05 Timer : Context : close definition --&gt; cumulated time : 0.138143 Timer : Field : rcv data --&gt; cumulated time : 0.026445 Timer : Field : send data --&gt; cumulated time : 0.03075 Timer : Files : close --&gt; cumulated time : 0.00188847 Timer : Files : create headers --&gt; cumulated time : 0.00926207 Timer : Files : get data infos --&gt; cumulated time : 0.000444779 Timer : Files : open --&gt; cumulated time : 0.00804241 Timer : Files : writing data --&gt; cumulated time : 0.00190237 Timer : Files : writing time axis --&gt; cumulated time : 0.0018695 Timer : Process events --&gt; cumulated time : 0.03559 Timer : Process request --&gt; cumulated time : 0.000316099 Timer : XIOS --&gt; cumulated time : 0.327634 Timer : XIOS close definition --&gt; cumulated time : 0.139521 Timer : XIOS context finalize --&gt; cumulated time : 0.00226334 Timer : XIOS finalize --&gt; cumulated time : 0 Timer : XIOS get variable data --&gt; cumulated time : 0.000234546 Timer : XIOS init --&gt; cumulated time : 0 Timer : XIOS init context --&gt; cumulated time : 0.00179636 Timer : XIOS init/finalize --&gt; cumulated time : 0.359765 Timer : XIOS send field --&gt; cumulated time : 0.176479 </pre> |
|--|---|

FIGURE 13 : EXAMPLE OF THE PERFORMANCE REPORT AND CODE PROFILING DISPLAY.

## 3.2 Development of new transfer protocols

The transfer protocol between clients and servers is based on non-blocking asynchronous MPI communications in order to guarantee the overlapping of data transfers with computation on the client side and writing on the server side. Each client sends its data to one or more servers, and each server receives data from one or more clients. To guarantee asynchrony, for each connected client-server pair, it is necessary to use buffers on both client and server sides. The messages sent from a client to a server are of diverse types and self-describing, i.e. they contain all the structural information allowing, when they are received on the server side, to unpack all stored data whatever its type and to call the C++ method corresponding to the requested action. Each message therefore has a different size, and some can be very small, such as sending attributes or updating the calendar. We therefore reserve the possibility of concatenating several messages in the client buffer before sending them in a single call to the server, so as not to overload the MPI stack with too many calls, in order to optimise latencies.

### a) [The transfer protocol of XIOS 2.5](#)

In that version of the protocol, which is still available in XIOS-3, see below, the client side has a double buffer of fixed size. Buffer-1 contains the messages being sent to the servers. As for each client-server pair, it is forbidden to have several MPI requests being sent, as long as this request has not been transferred to the server, the following messages are concatenated in buffer-2. For each new message to be sent, which is concatenated in buffer-2, we test whether the request being transferred (buffer-1) has been completed using the non-blocking `MPI_Test` function. If it is, we send all the concatenated messages from buffer-1 to the server using the asynchronous non-blocking function `MPI_Issend`. The roles of buffers 1 and 2 are then swapped, with the new messages now concatenated in buffer-1 and so on (Figure 14)...

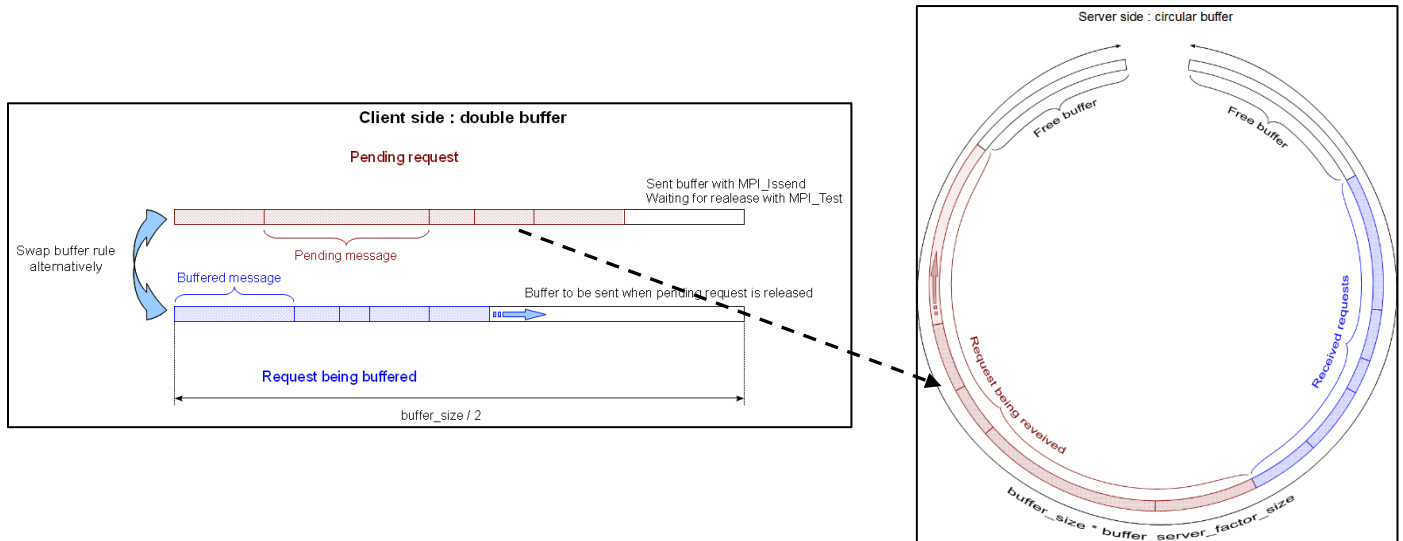


FIGURE 14 : ILLUSTRATION OF THE TRANSFER PROTOCOL OF VERSION 2.5.

On the server side, there is a circular buffer, part of which is free, part of which is occupied by messages already received from the paired client and part of which is occupied by a request currently being received (Figure 14). The server will prioritise the reception of requests from clients in order to free up space in the client-side buffers. This is only possible if the free space in the server buffer is sufficient to accommodate the entire MPI request. If the buffer has no more requests to receive, it will try to unpack the messages already received (in blue in the figure). The set of client messages corresponding to the same action is called an "event". It is only possible to unpack a message if an event is complete, i.e. if all the client messages making up the event have been received. In this case, the server unstacks the corresponding messages, calls the C++ method associated with the event and then releases the associated memory space in the buffer.

When the memory space in the server buffer becomes insufficient to receive a new MPI request, it refuses any reception until the space is freed. To do this, it must process events to free some space. In this case, on the client side, the request being sent (in red) remains pending and messages are concatenated in the other buffer (in blue). When the buffer is completely full, we enter in "blocking" mode, which means that XIOS does not return until this request has been received and enough space has been freed up in the server buffer. This situation is a sign that the server does not have enough time to process all the events sent to it by these clients: too many events, too many writes, etc. In this case, it may be necessary to increase the number of servers to reduce the load.

That protocol has been proven for many years, and it does indeed overlap the transfer times between clients and servers and also the file reading/writing of the servers, with the model computation. However, when the servers become overloaded, blocking appears on the client side. Due to a traffic jam effect, these blockings are sometimes multiplied: one client has to wait, then it is the turn of another, etc. As clients are often synchronised in the models by collective communications, even

if the blocking time per client is low, the overall time can increase significantly. These situations are quite difficult to diagnose and can lead to very poor performance in the case of server saturation. In addition, client blocking can also lead to a general blocking (dead-lock) in very rare situations. In fact, a client can stay blocked because its corresponding server buffer is full, and the server in question cannot free up memory space. The event that must be processed to allow this release is not complete, because one of the messages making up the event remains stored in another client's buffer and has not been sent. This client cannot send its buffer either, because it is itself stuck in a collective communication, waiting for the first client to release the server's buffer; the snake biting its own tail... To avoid such situations, it was necessary to impose a criterion on the maximum number of messages that could be concatenated in a client buffer. Unfortunately, for certain configurations, such as NEMO ones for example, this number can be quite low, of the order of a few units. We therefore lose a good part of the effect of asynchronism, which should allow to smooth out load peaks; this results in "chopping" the protocol and increasing the phenomenon of contention described in the previous paragraph, thus leading to poor performance during client-server transfers.

That version of the protocol requires fixed buffer sizes, as they are not expandable at runtime, and must be defined before the first send. Their final size must therefore be estimated very early on during initialisation. In the new version 3 of XIOS, in order to evaluate these buffer sizes more precisely, it is necessary to have prior exchanges with the server, otherwise the evaluations will not be very precise and may lead to an over- or under-evaluation of the size. In the first case, more memory is consumed than necessary, in the second case, this impacts performance, or even causes the code to stop if the buffers are too small to contain a message to be sent.

This lack of flexibility of the protocol, whose buffer sizes cannot be reallocated, and the drop in performance when traffic becomes less fluid motivated the development of new transfer protocols in the XIOS-3 version.

#### b) XIOS-3 protocol extension - adding passive one-sided communications

The main cause of the performance degradation of XIOS 2.5 protocol is the inability of the servers to retrieve a missing message from the client buffer without the intervention of the client. Thus, to avoid dead-locks, the number of messages stored on the client side must be limited, which increases contention. To circumvent this problem, we added the possibility for servers to fetch messages directly from the client buffer using passive "one-sided" MPI communications. This is done by exposing the memory of the client buffers through MPI windows (MPI\_Win). The local memory address of the buffers is then sent to the servers, which can then retrieve the contents of the buffer in memory at any time without the clients' intervention, using passive communication (MPI\_Win\_lock / MPI\_Get / MPI\_Win\_unlock) (Figure 15).

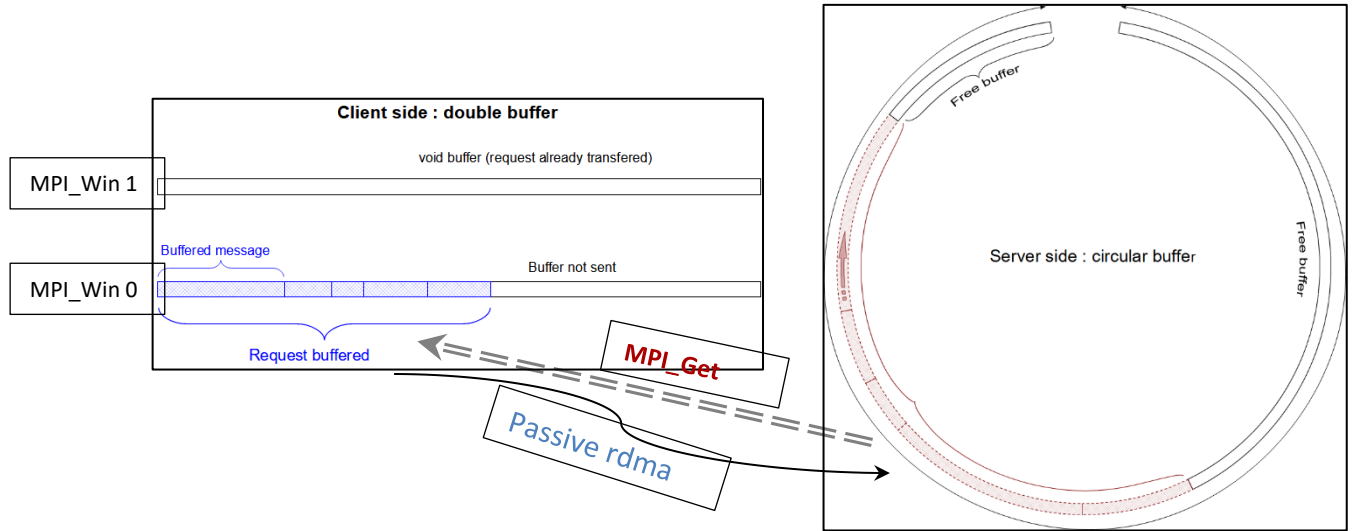


FIGURE 15 : ILLUSTRATION OF THE EXTENSION OF THE VERSION 3 TRANSFER PROTOCOL, BY ADDING PASSIVE "ONE-SIDED" MPI COMMUNICATION

The protocol is still mainly based on the asynchronous `MPI_Issend`, `MPI_Irecv` primitives of the original protocol. However, when the server detects a potential "dead-lock" situation, i.e. when it no longer receives anything from the clients, and it has no more events to process because they are incomplete, it implements "one-sided" communications to fetch the missing messages from the client buffer. The weakness of this approach lies in the fact that the server does not know which client to go to for the missing messages, so it must successively check the buffer of each client with which it is connected. On the other hand, transfers with `MPI_Get` are not asynchronous for the server, so it cannot in this case cover this type of transfer with writes. So this approach is to be used exceptionally to unblock critical situations.

We also took advantage of this development to improve the asynchronous P2P `MPI_Issend`/`MPI_Irecv` protocol by using the "Matching Probe" functions with the use of `MPI_Improbe` and `MPI_Imrecv`. This allows better orchestration of receptions between different clients and smooth the traffic flow. In addition, we have also made the management of buffers more flexible by allowing them to grow dynamically. In addition to improving the memory management of buffers, this introduces fewer constraints on development and more concise code.

For these new features, we based ourselves on the recent MPI 3.1 (2015) standard for dynamic use of MPI windows (`MPI_Win_create_dynamic` / `MPI_Win_attach`) and the "matching probe" functions (`MPI_Improbe`/`MPI_Imrecv`).

### c) New pure "one-sided" protocol

To further improve the transfer protocol, it is advisable to base it solely on passive "one-sided" communications. In this case, the clients only deposit their data in memory and do not intervene during the transfers. It is the servers that choose to initiate the transfers of client messages that are necessary for the completion of an event. The servers therefore have complete control over the orchestration of message transfers for optimum traffic flow. However, the servers must have knowledge of the address and sizes of messages that are made available in the clients. To do this, we continue to send the message header using a point-to-point protocol (MPI\_Isend/MPI\_Irecv) but with very small MPI requests, only a few bytes (size and address). The message is copied into the client buffer and will be downloaded later by the server after receiving the headers. As the server has received the memory address of the message and its size, it is able to perform a random memory access to copy it into its own buffer. As we want the transfer to be asynchronous, we use a new function "MPI\_Rget" from the MPI 3.1 standard, similar to MPI\_Get but which allows asynchronism. For more flexibility, the client and server buffers are now both circular buffers, which better optimise memory management (Figure 16).

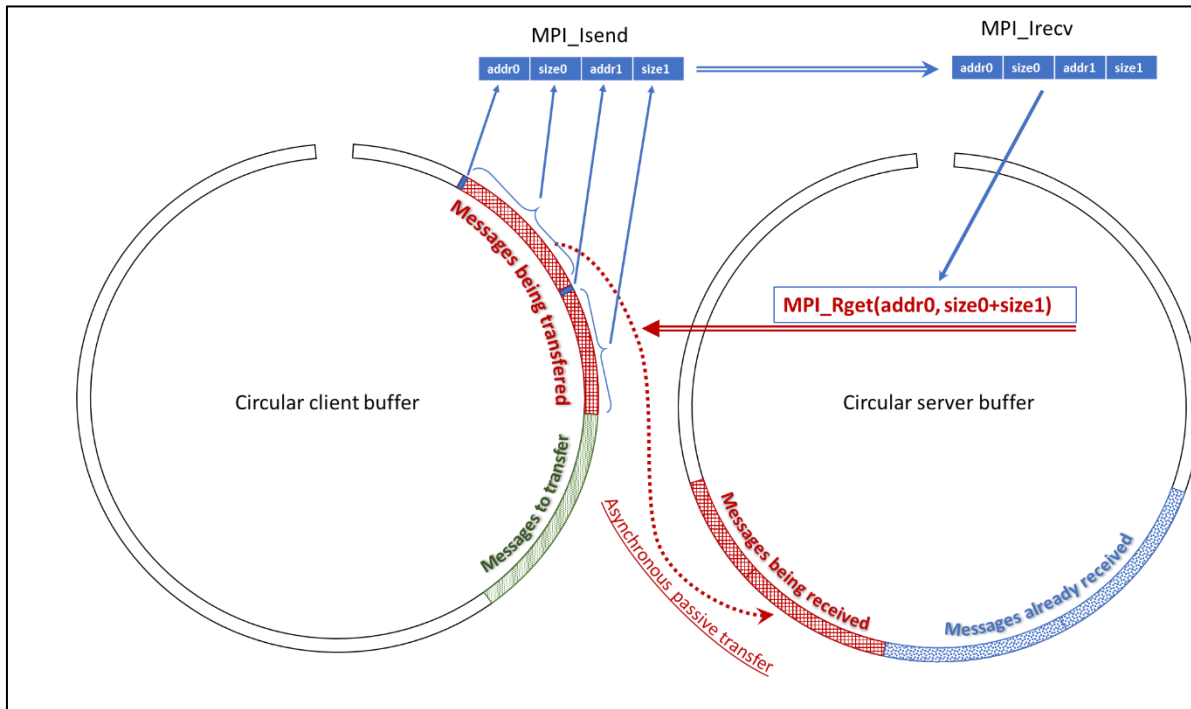


FIGURE 16 : ILLUSTRATION OF THE NEW, PURE "ONE-SIDED" TRANSFER PROTOCOL

As we are using fairly recent features of the MPI library that are not very well tested, we have observed a lot of problems when implementing the new protocols. To avoid users ending up with crashes on their machines, we decided to make several protocols cohabit within the XIOS sources by virtualising the classes. It is now possible to choose one or other of the protocols during an

execution through the "transport\_protocol" parameter in the iodef.xml file. For the moment, 2 choices are possible:

- transport\_protocol = "legacy": mixed protocol P2P + "one sided"
- transport\_protocol = "one\_sided": pure "one sided"

In the future, other specific protocols will be developed to allow better matching of client-server transfers according to the type of service expected: write service, read service, code coupling, etc.

### 3.3 New concepts to rationalise and improve the distribution of grids

XIOS is a tool for managing and exchanging data based on scalability: each component of the code, whether it implements transfers, workflow, transformations, writes, etc., acts on data that is distributed over all the calculation cores ; all the algorithms developed are therefore parallel. These data flows are defined in relation to their associated underlying geometric objects, i.e. their grid. The management of the grid parallelism and the algorithms for switching from one grid to another are the keystone of the XIOS edifice. In version 2, these complex algorithms are scattered throughout the code with little reusability, making the coding considerably cumbersome and difficult to read and optimise. This slows down new developments as specific algorithms have to be redeveloped each time. Furthermore, methods and algorithms used in version 2 are very memory consuming, which causes major problems when scaling up. In version 3, we decided to conceptualise these algorithms and implement them in reusable classes, in order to obtain a more readable code, easily extensible, improving performance and decreasing the memory footprint.

#### a) The elements

##### *(i) The concept of element and the "CLocalElement" class*

The grids on which the data fields are defined are obtained by aggregating 3 elementary types of objects:

- Horizontal domains `< domain />`: geometrically describes a 2-D horizontal layer on the surface of the sphere
- Axes `< axis />`: describes a vertical axis or an additional dimension (1-D).
- Scalars `< scalar />`: a geometric object of zero dimension (0-D).

Grids are constructed from these elements by applying the properties of the tensor product. For example, a grid defined by "`domain ⊗ axis ⊗ axis ⊗ scalar`" is 4-D. These basic elements, although very different in terms of their geometric properties, are described in a similar way in terms of their distribution. At the global level, they are described by a series of global indices ranging from 0 to `n_glo-1`, `n_glo` being the global size of the element. Locally, as the element is

distributed, it will be described by a subset of the global indices corresponding to the local distribution (Figure 17). This distribution is encapsulated in the "*CLocalElement*" class.

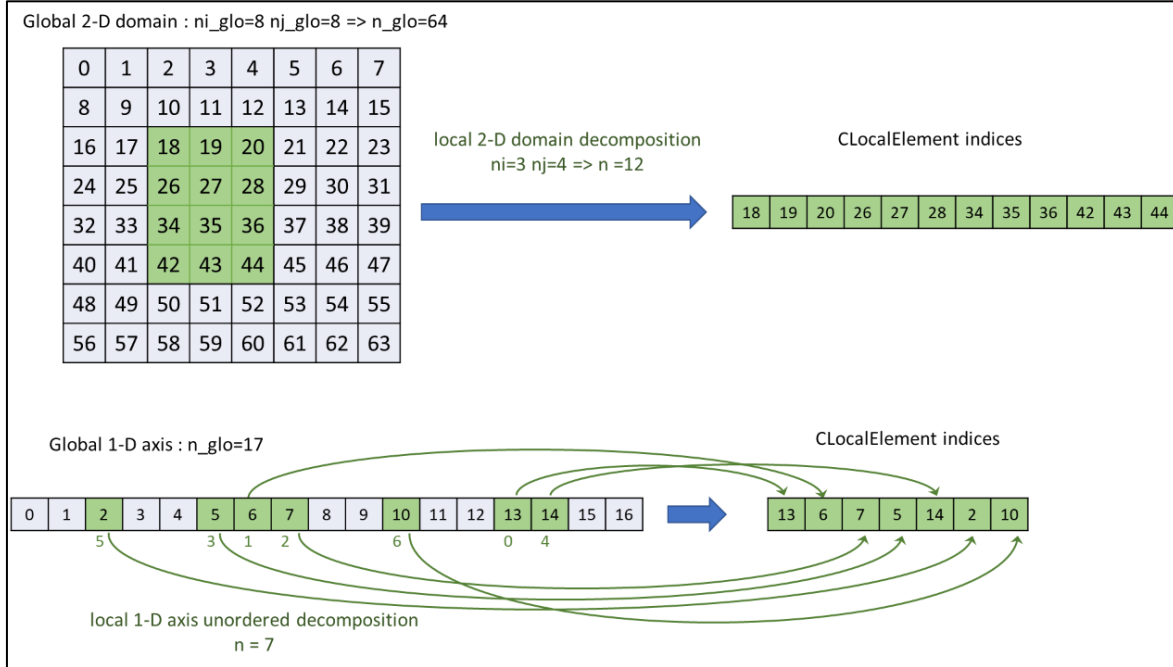


FIGURE 17 : EXAMPLE OF A REPRESENTATION OF THE LOCAL DISTRIBUTION OF ELEMENTS FOR A 2D-DOMAIN AND A 3D-AXIS

### (ii) The *CDistributedElement* class

The *CLocalElement* class describes the local distribution of an element. However, it is necessary to have local knowledge of the global indices distributed on other clients or servers. The *CDistributedElement* class therefore encapsulates several sets of global indices of the element, each of which is associated with a distinct rank (Figure 18). For example, this information may be needed to determine which parts of a local field to send to different servers.

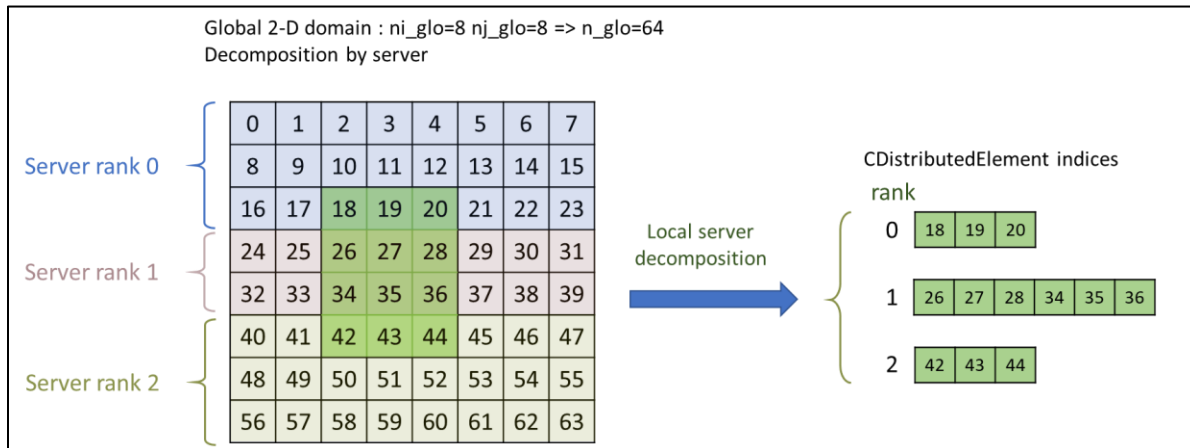


FIGURE 18 : EXAMPLE OF A REPRESENTATION OF A DISTRIBUTED ELEMENT.



(iii) *The CGridLocalElement and CGridDistributedElement classes*

The *CGridLocalElement* class encapsulates the list of objects of type *CLocalElement* corresponding to the definition of the grid composed of several elements (domain, axis, scalar) by application of the tensor product. It therefore describes the local distribution of a multidimensional field. The *CGridDistributedElement* class is its equivalent to describe a distribution on different ranks.

b) The concept of "view"

The concept of "view" defines how an element is represented in memory. The same element can have a different representation in different parts of a model or code. For example, to represent a field with its overlapping halos, its representation in memory may be larger than the local grid on which it is defined. On the other hand, if the grid is compressed, as in the case of land use models for which only the land points are calculated, its representation in memory will be smaller. In the XIOS workflow, only the unmasked points are extracted from the model, so it will have a more compressed memory representation. Finally, when we wish to write the fields to a file, they must be written in an uncompressed form (but with "missing values" for the masked values). Views therefore define different memory representations of the same element. Although it is possible to add specific views, 3 types of commonly used views are predefined:

- The "model view": the view defining the memory representation on the model side, taking into account the mask and the indexes
- The "workflow view": the view defining the memory representation in the XIOS workflow for which the data is compressed.
- The "full view": the view defining the memory representation which includes also masked values (non-compressed view), identical to the definition of the element to which it is attached.

A data flow will initially be defined on a model view and then evolve, as the workflow progresses, on different views before finally being written or returned to the model.

(i) *The CLocalView class*

This class encapsulates the notion of a view for the local distribution of an element. The information is stored in the form of indices referring to the global indices of the local element (see Figure 19). If  $n$  is the number of points composing the local element, the view referring to it will have indices varying from 0 to  $n-1$ . Values  $<0$  or  $>n$  indicate invalid memory areas that will not be retrieved.

(ii) *The CDistributedView class*

This class encapsulates the notions of view in a similar way to the *CLocalView* class but for *CDistributedElement*.



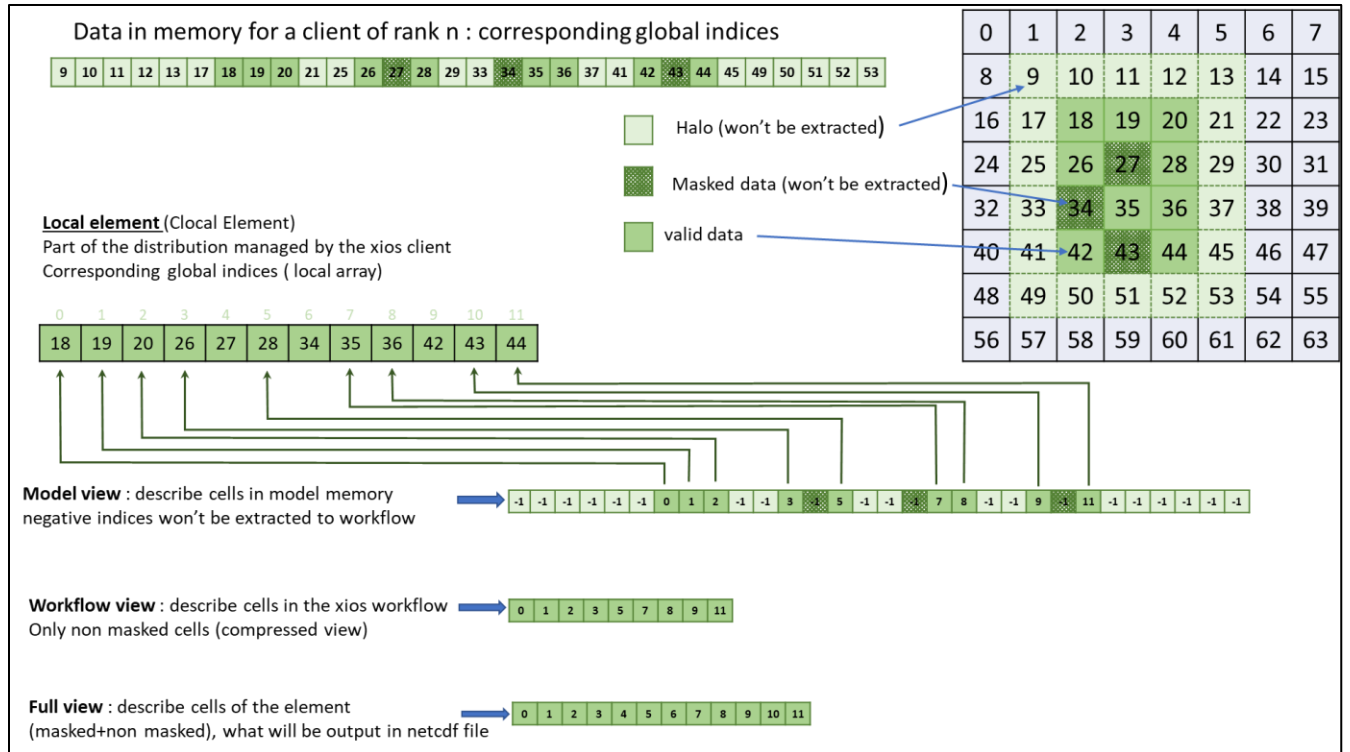


FIGURE 19 : EXAMPLE OF THE REPRESENTATION OF THE 3 TYPES OF VIEWS: MODEL VIEW, WORKFLOW VIEW AND FULL VIEW.

### (iii) The CGridLocalView and CGridDistributedView classes

Encapsulates the notions of views for multidimensional fields.

#### c) The connectors

##### (i) View connectors

Connectors are the objects that allow data flows to pass from one view to another. They are initialized based on a source view and a destination view. This initialisation phase can be more or less complex depending on the type of connector. It calculates the coefficients of the transfer matrix between the source view and the destination view. During the workflow, the "transfer" class method will be applied to the source flows, which uses the previously precalculated coefficients to obtain the destination flow. A large part of the cost of the workflow will be found in these calls, which is why we have carefully optimised them and completely inlined them in the form of computation kernels. There is a whole family of connectors, each with a specific function. We can mention the main ones:

- Local connectors (*CLocalConnector*): allows to switch from the local view of an element to another local view of the same element. Used to extract data flows from the model to the workflow (and vice versa): model view -> workflow view (or workflow view -> model view). Or to write (or read) workflow data to a file: workflow view -> full view (or full view -> workflow view).

- Scatterer connectors (class *CScattererConnector*): allow to switch from a local view to a distributed view of the same element. Used for example for transfers from clients to servers, where each client distributes its data stream to different servers.
- Gatherer connectors (class *CGathererConnector*): allows to switch from a distributed view to a local view of the same element. Used for example when receiving data on the server side, to gather data streams received by several clients
- Transformation connectors (class *CTransformConnector*): used to pass the flows of a source element necessary for a spatial transformation. It is a connector allowing to switch from the local view of an element to the local view of another type of element. Internally, it chains a scatterer connector transfer, followed by MPI exchanges between clients, and then a gatherer connector transfer.

#### (ii) *Grid connectors*

To move from one flow on a multidimensional view (*CGridLocalView* or *CGridDistributedView*) to another, grid connectors are used, which recursively call view connectors for each of the elements composing the grid, using the properties of the tensor product. This involves calculating the transfer matrix independently for each element. Compared to version 2, where the transfer matrix was calculated at the full grid level, the storage of coefficients in memory is greatly reduced. Let's take for example a 4-D grid composed of a 2-D domain (size 100x100) and 2 1-D axes (of size 100). The number of indices in the grid is therefore  $100^4 = 100\,000\,000$ .

- Version 2: Transfer matrix  $\sim \alpha 100\,000\,000$
- Version 3: domain  $\otimes$  axis  $\otimes$  axis  $\Rightarrow \alpha 100*100 + \alpha 100 + \alpha 100 \sim \alpha 10\,000$

This results in a reduction in memory storage by a factor of 10,000. Furthermore, as memory accesses are reduced, a gain in performance is also expected.

#### (iii) *Chain of connectors*

The connectors are encapsulated in filters (family class *CFilter*), which can themselves combine several types of connectors. These filters are then chained together to form the complete workflow (see Figure 20). This new infrastructure now concentrates all computational costs in filters and connectors that are highly reusable. This approach will greatly simplify future code optimisations and porting to new technologies. In particular, in the near future we wish to implement OpenMP (multithreaded) or port to GPU architectures, using an incremental approach on the computational kernels.

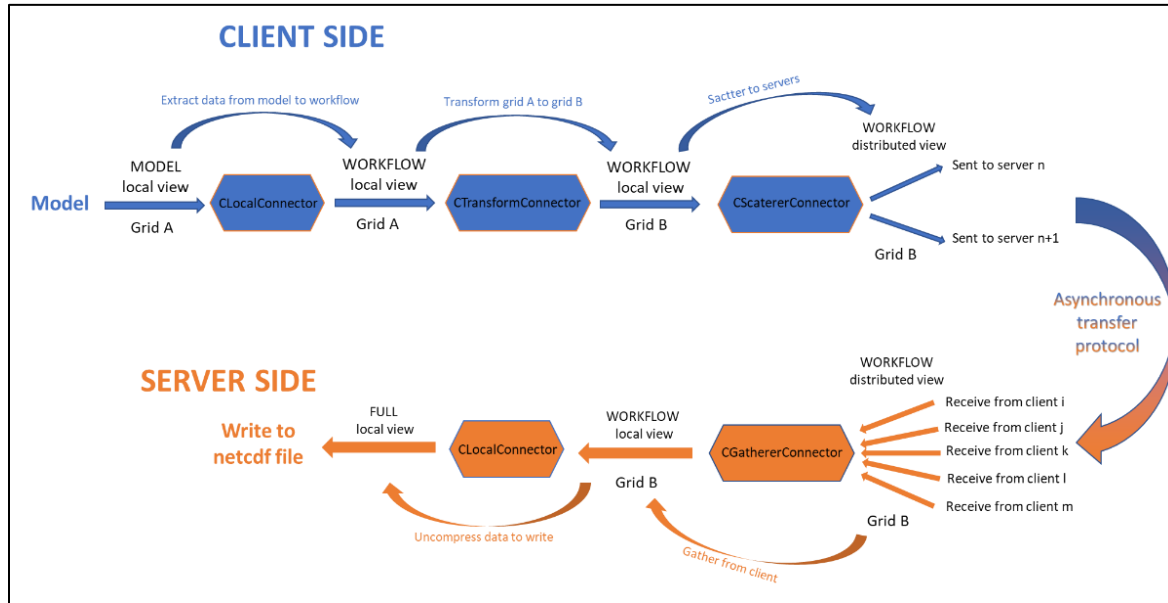


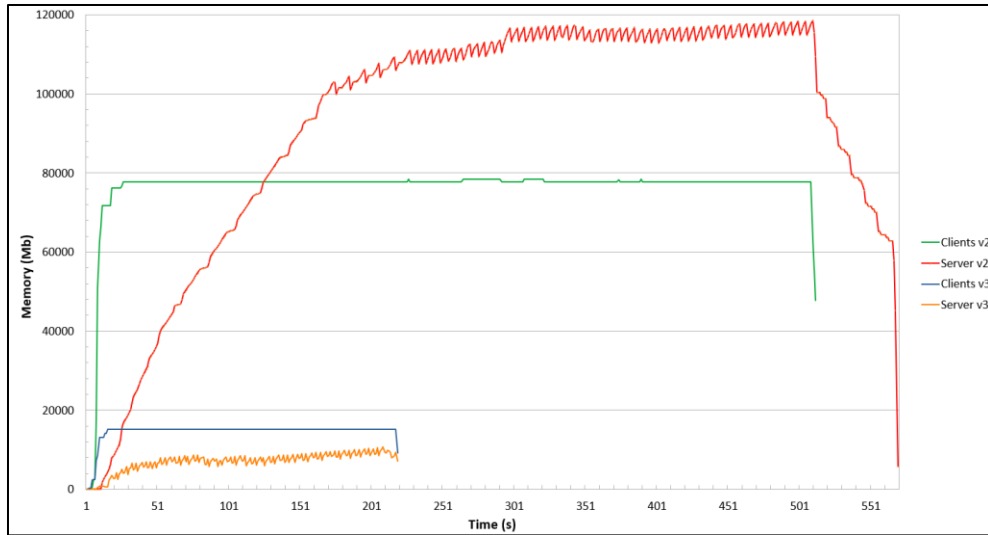
FIGURE 20 : AN EXAMPLE OF A SIMPLE WORKFLOW SHOWING THE CHAINING OF CLIENT-SIDE AND SERVER-SIDE CONNECTORS.

#### d) Preliminary performance and memory consumption tests

So far, we have validated XIOS-3 mainly on the functionalities and little on the performance. Nevertheless, to illustrate the interest of the new approach using connectors, we have executed an identical test case on V2 and V3, consisting of an extremely simple workflow: 6 fields to be written, each on a different 4D grid of size 100x100x100x100. One output per day over 10 days is performed (see Figure 21). We use 48 clients and 1 server. The curve shows the total memory used by the 48 clients and the server for the XIOS-2 and XIOS-3 versions. Over the entire test, we observe a reduction in the maximum memory used of almost a factor of 10 on the client side and almost a factor of 15 on the server side, in favour of V3. In addition, a factor of 2 was observed in the restitution time with a reduced initialization phase. These tests are very encouraging and show the potential of this new version, without having still specifically looked at optimisation.

### 3.4 Development of a new internal HPC service-oriented infrastructure

In version 2.0, XIOS had a very rigid infrastructure allowing several components of an ESM to exchange data flows with a single server pool, with either write or read functionality. Version 2.5 introduced some flexibility by adding the possibility of a second level of servers. The first level receives the streams from the models and then redistributes them to the different 2<sup>nd</sup> level pools. Each 2<sup>nd</sup> level pool is responsible for writing one or more files, which increases the level of I/O



**FIGURE 21 : COMPARISON BETWEEN XIOS-V2 AND XIOS-V3 ON THE MEMORY CONSUMED AS A FUNCTION OF TIME, FOR A SIMPLE TEST CASE CONSISTING IN SENDING 6 4D FIELDS ON 4 DIFFERENT GRIDS OF SIZE 100x100x100x100. 1 OUTPUT PER DAY OVER 10 DAYS. 48 CLIENTS AND 1 SERVER.**

parallelism, as several files can be written simultaneously. In version 3, we wanted to completely revise this overly rigid infrastructure by introducing the notion of "HPC service". The main idea is still to dedicate a set of free resources for XIOS servers within the MPI partition. During execution, depending on requirements, one or more services can be dynamically launched on the part of the resources still free. A service groups a set of MPI processes dedicated to the same function, that work together in parallel. All these services can be interconnected through the new XIOS middleware, which uses the asynchronous transfer protocol. This new middleware therefore unifies the way in which flows are exchanged between:

- Models <-> services
- Services <-> services (chaining of services)
- Models <-> models (code coupling)

The old features of version 2.5 level-1 and level-2 servers were therefore redesigned and recoded in terms of service in the new infrastructure:

- Stand-alone I/O server: IO\_SERVER services
- Level-1 server: GATHERER services
- Level-2 server: WRITER services

It is now possible to run several services of the same type that work independently. For example, each model has the possibility to have its own dedicated IO\_SERVER service. It is even possible to assign a file to a specific IO\_SERVER service.

This infrastructure has been designed in such a way that it is easy to create new services and interconnect them through an external XML description. Moreover, the following are currently under development:

- A **READER** service to perform reads: currently this functionality is carried by the **IO\_SERVER** or **GATHERER** services. This will allow the decoupling of reads and writes into separate services.
- An **OFFLOAD** service: deporting part of the workflow, described in XML, to dedicated resources. This will allow expensive diagnostics to be run asynchronously before sending the data produced to a **WRITER** service.

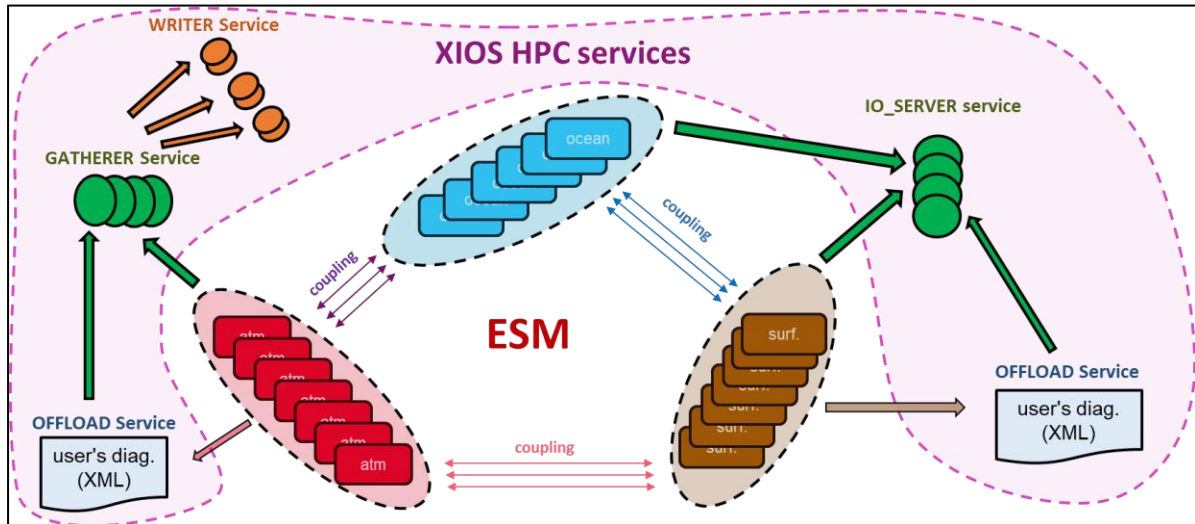


FIGURE 22: ILLUSTRATION OF THE NEW XIOS-V3 INFRASTRUCTURE, ENABLING THE DYNAMIC LAUNCH OF INTERCONNECTED SERVICES AND THE COUPLING OF MODELS.

#### a) Internal infrastructure

Service management is based on a very hierarchical internal infrastructure: resources --> services -> contexts.

##### (i) *Resource management*

When the code is launched, the set of allocated cores represents the entire available resource. When the global communicator `MPI_COMM_WORLD` is split, each code receives a share of the resources via its associated communicator. The classes representing these allocated resources are of type "resource pool" (`CPoolRessource` class). In each "resource pool" a single "CLIENT" type service is launched, corresponding to the execution of one of the models. This makes it possible to consider the models as services in their own right which produce or receive data flows.

The servers also receive a communicator with which free resources are associated. In these free resources, it is possible to create several independent resource pools which will be encapsulated in objects of type "server resource" (`CServerRessource` class). The object allowing to manage its allocations is the "resource manager" (`CResourcesmanager` class) (see Figure 23). Each server pool is associated with a unique identifier. Subsequently, one or more services can be launched in each of the newly created pools.

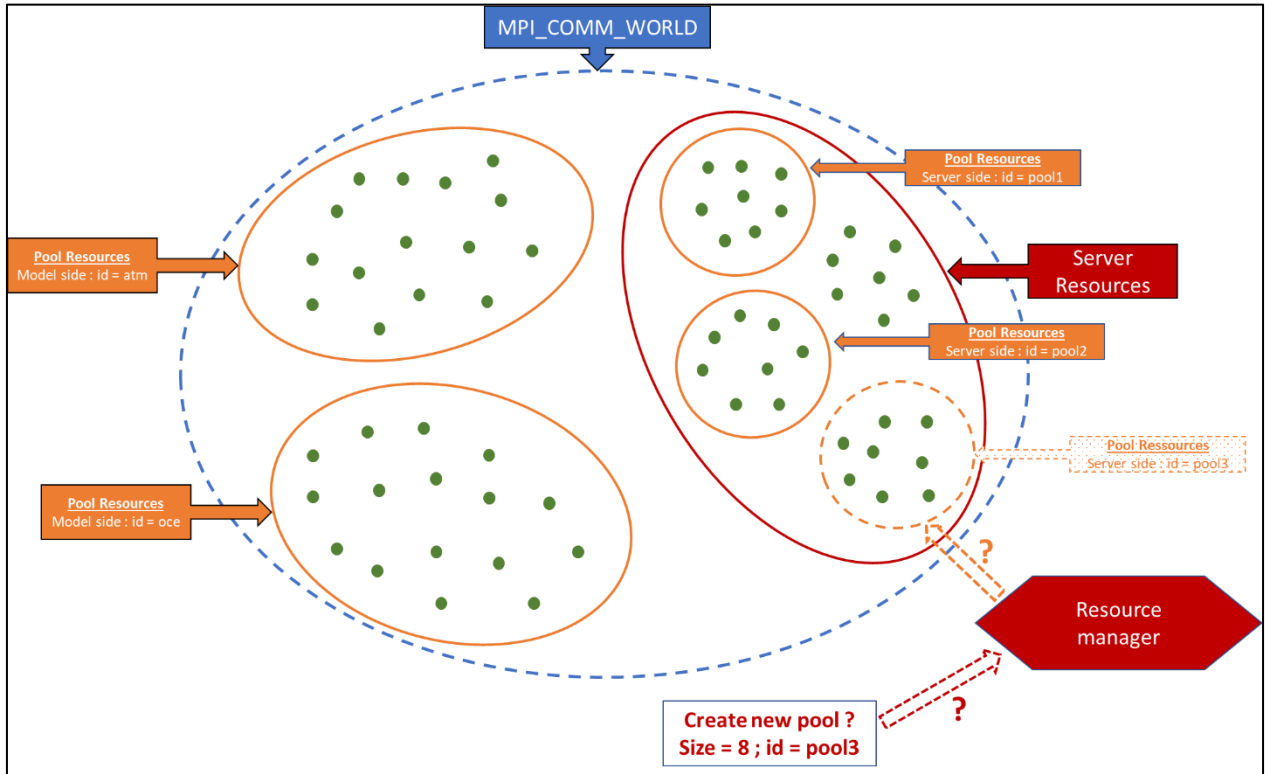


FIGURE 23 : ILLUSTRATION OF RESOURCE MANAGEMENT IN THE NEW INFRASTRUCTURE.

### (ii) Service management

In a "resource pool", one or more services can be launched depending on the free resources. This request is made through the "service manager" (*CServiceManager* class), by specifying the identifier of the new service, its type (WRITER, GATHERER, IO\_SERVER) and its size (number of processes). If there is enough space in the pool, the new service is created. It is possible to overlap several services on the same processes, provided that there is a total overlap (same size) otherwise they must be totally disjoint. Services are encapsulated in the *CService* class. The service identifier must be unique within the same resource pool. At the global level, it is uniquely identified by the string "pool\_id::service\_id".

### (iii) Managing contexts

In each newly created service, one or more contexts can be launched. The contexts all overlap completely and their size is that of the service with which they are associated. The creation of new contexts is done through a request to the "context manager" (*CContextManager*) by providing the identifier of the new context as well as the identifier of the service on which it will be created.

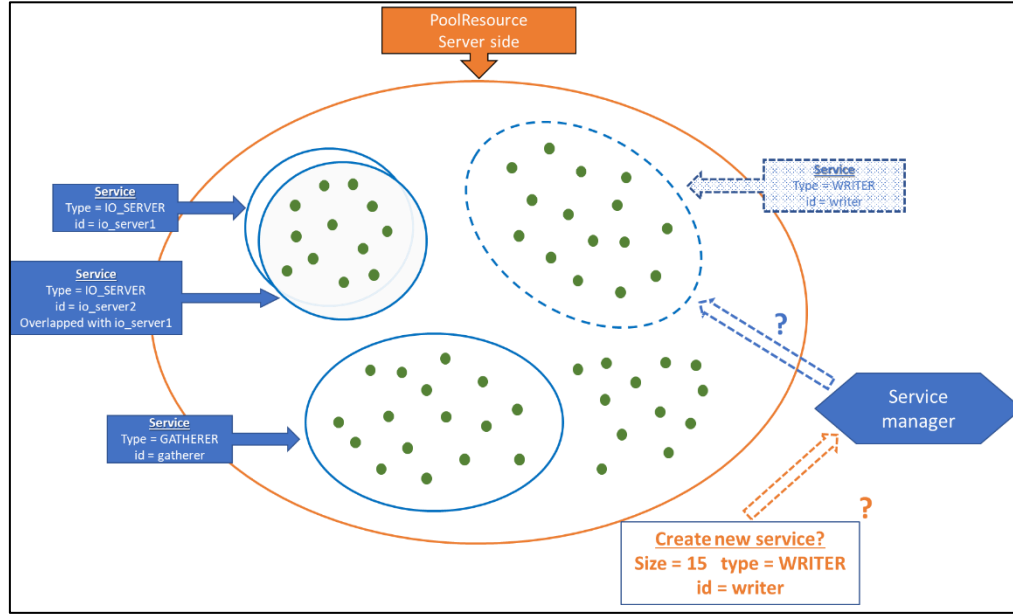


FIGURE 24: ILLUSTRATION OF SERVICE MANAGEMENT IN THE NEW INFRASTRUCTURE

Contexts are encapsulated in the *CServerContext* class. They are uniquely identified by the string: "poolId::serviceId::contextId". Another extremely important role of the context manager is to interconnect two contexts from their unique source and destination identifiers. An intercommunicator is then returned, which then allows flows to be exchanged between the two contexts via the transfer protocol.

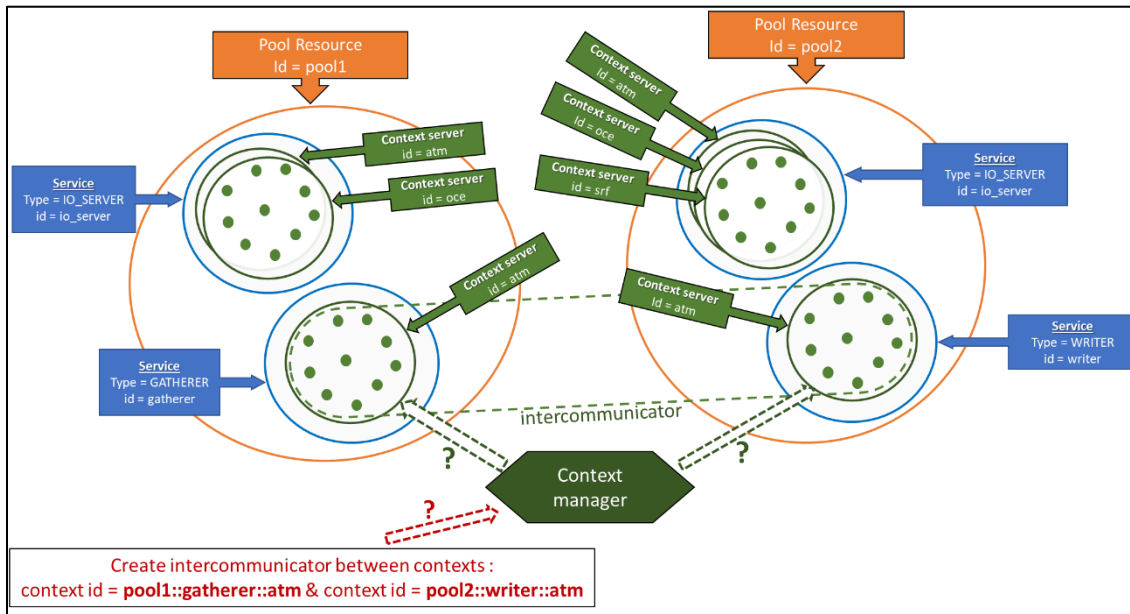


FIGURE 25: ILLUSTRATION OF SERVER CONTEXT MANAGEMENT IN THE NEW INFRASTRUCTURE.



## b) Use of services

### (i) *Default behaviour*

For the sake of backward compatibility, the default service behaviour of the XIOS-3 version emulates the XIOS-2.5 version. Namely:

- No resources for servers: activation of the "attached" mode
- A single server level: a default pool is created in which a single IO\_SERVER service is started. Contexts are started in this service
- 2 levels of servers: a default pool is created in which a GATHERER service and a WRITER service are launched in proportion to the resources defined by the "ratioServer2" parameter

### (ii) *Definition of specific services*

These features are still experimental and only IO\_SERVER type services can be managed manually. In the XIOS context, through XML, it is possible to define server resource pools in which services will be launched. The volume of resources can be expressed either as a number of processes or as a fraction of the total resources available. For services, this refers to the fraction of the resource pool in which it is running. It is also possible to specify that the service uses the same resources as another (overlap attribute) (see Figure 26).

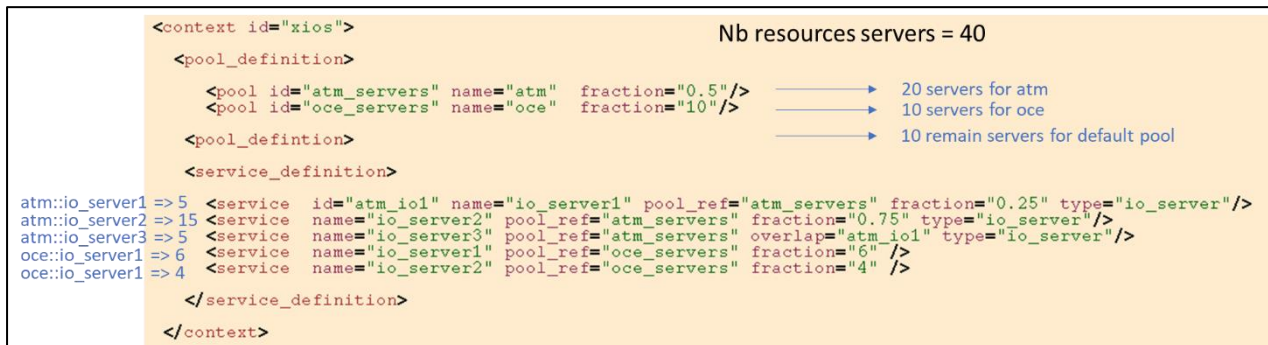


FIGURE 26 : DECLARATION OF RESOURCE POOLS AND SERVICES AT THE XML LEVEL.

When defining the model context, an IO\_SERVER service can be assigned which will be used by default for all files, but it is also possible to assign a specific service for a given file (Figure 27).

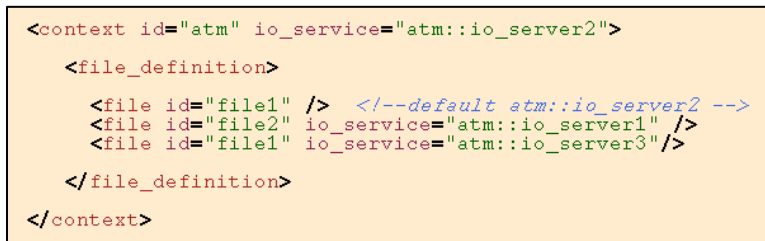


FIGURE 27 : ASSIGNMENT OF SERVICES FOR OUTPUTS.



### c) Model coupling

In the new infrastructure, models have the ability to exchange data through their XIOS context. A new type of container has therefore been developed to identify outgoing and incoming flows to be exchanged, very similar to the `<file/>` type container.

- `<coupler_in context="model_id::source_context_id"/>` : for streams received from another context
- `<coupler_out context="model_id::target_context_id"/>` : for streams sent to another context

As the streams are received on the source grid, it was also necessary to exchange the grid definition information in advance during the initialisation phase. Then, if we wish to remap to the destination grid, it is sufficient to apply a piece of workflow to carry out the interpolations, whether they are vertical or horizontal (Figure 28). The sending and receiving of model-side flows remain unchanged and still use the standard interface: `CALL xios_send_field("field_id", field) / CALL xios_rcv_field("field_id", field).`

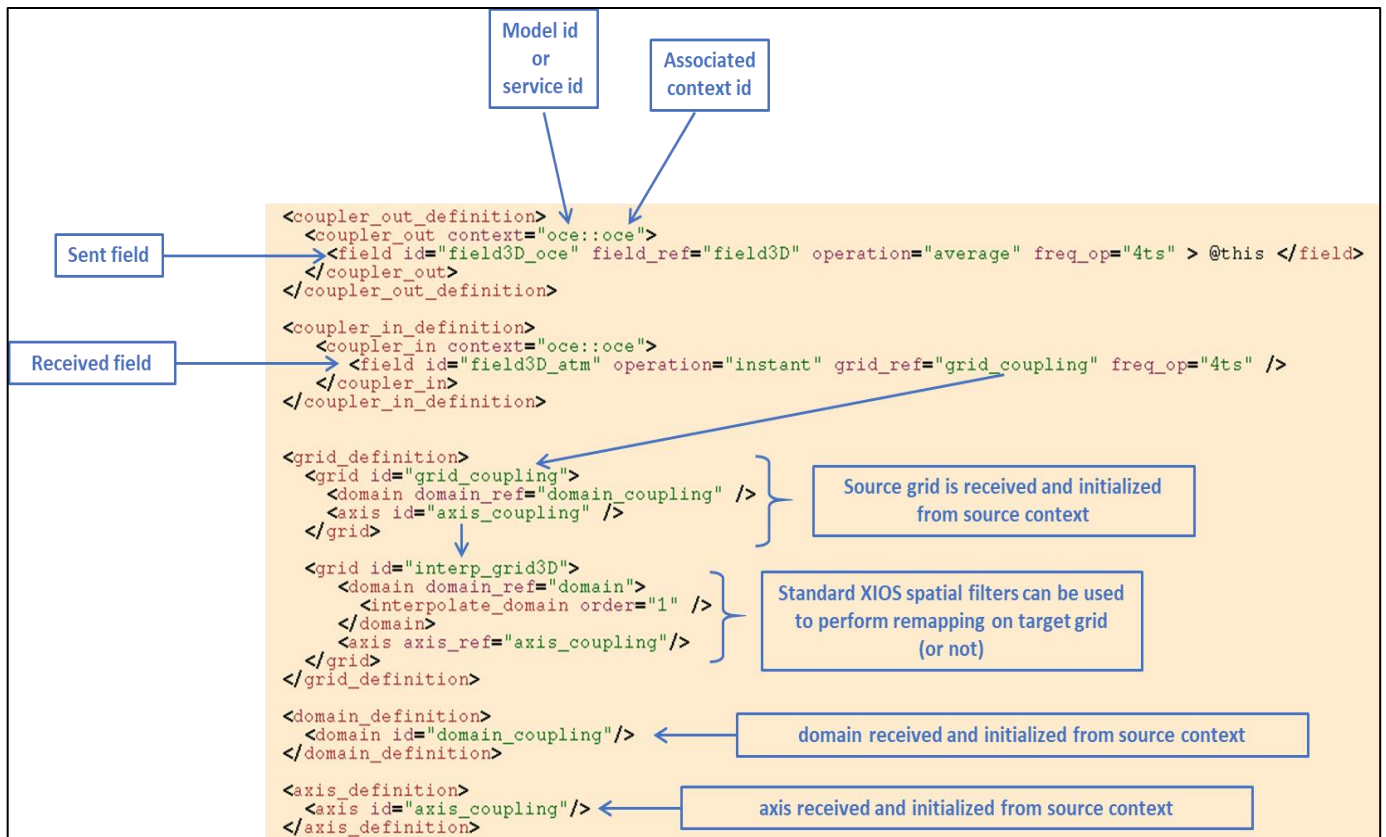


FIGURE 28 : EXAMPLE OF AN XML WORKFLOW FOR CONTEXT COUPLING. A FLOW IS SENT FROM THE CURRENT CONTEXT (ATM) TO THE "OCE" CONTEXT (NOT SHOWN) VIA THE "COUPLER\_OUT" CONTAINER. ANOTHER STREAM IS RECEIVED FROM THE "OCE" CONTEXT VIA A "COUPLER\_IN" CONTAINER. A HORIZONTAL INTERPOLATION FOLLOWED BY A VERTICAL INTERPOLATION IS THEN APPLIED TO THE INCOMING STREAM.

## 4. Validation

The new version XIOS-3 has been independently validated on each principal component of the IPSL coupled system-earth model. This includes the DYNAMICO, LMDZ, ORCHIDEE and NEMO models. In a first step, we focused on the functionalities, to ensure the non-regression of the results. This verification was carried out on:

- Readings and forcing of the different models
- The standard outputs of the models
- The CMIP6 workflow of models, including outputs associated with diagnostics of various kinds, making extensive use of XIOS workflow functionality.

In all, nearly 1000 variables were compared bit to bit to the same outputs using version 2.5. In some cases, some differences appeared due to bugs in version 3 which have since been corrected. In other cases, the algorithm philosophy of the transformation engine in V3 has changed compared to version 2.5, so some changes had to be made to the XML files that describe the workflow. In a few rare cases, it is version 2.5 that has been caught out by not giving the correct results. Thus, to date, version XIOS-3 gives the expected outputs for all the components tested.

We also checked that the complete ESM, coupled with OASIS3-MCT, was working well. We have not yet had time to look in detail at the performance, but for the resolutions tested (100km~200km), the restitution times and memory consumption seems to lead to good results and has not shown any significant problems. The first explorations made on test cases show significant gains in output times and large decreases in memory consumption (of the order of a factor of 10). However, as a model is much more complex than a test case, a detailed analysis as well as possible corrections/optimisations will still take time, which we expect to perform before the end of the IS-ENES3 project.

## 5. Discussion and conclusion

We now have clean, robust code over which we have full control. All the tools to improve robustness and reliability will allow us to accelerate future developments by providing adequate diagnostics for debugging, performance and non-regression of features. The work done on the redesign of the whole code, including the development of new protocols, connectors, the new transformation engine and many other details, will allow us to improve the performance for scaling up and to make efficient global simulations at a few-kilometre resolution. This is a strong demand from users and institutes. In particular, memory consumption at these scales was a crucial issue that needed to be addressed. Finally, the new HPC service-oriented infrastructure is an open door to the future of climate modelling. As models become more complex, more and more diagnostics will have to be deported asynchronously. There will also be a need to run large ensembles of the Earth's climate system simultaneously, which will produce a lot of data. The development of a new service to manage and reduce the volume of data to be written “in situ” will be of great interest.

The arrival of neural networks and artificial intelligence methods will probably revolutionise climate modelling in the coming years. But interfacing these solutions developed in Python with our models in Fortran is not an easy task. We can therefore see the interest in developing new XIOS services that allow us to train neural networks “in situ” with data from our models, or conversely, during inference, to inject data flows from already trained neural networks. This new infrastructure has been specifically designed to allow the simple creation of new types of services that can be interconnected with the existing ecosystem.

The developments described in the initial deliverable, which have been postponed, will be taken up and finalised in the time between now and the end of the IS-ENES3 project, because although they are considered to be of lower priority than the code refactoring, they remain of great interest. Moreover, given the work already completed, their development in this new version will be much faster than the initial estimates.

The short-term priority is stabilization of the current functionalities. In addition to the XIOS3/trunk revision, which remain dedicated to evolve toward new developments, we have released a pre-stable beta version:

- <http://forge.ipsl.jussieu.fr/ioserver/browser/XIOS3/stable/xios3.0-beta>.

This version is now considered as the reference version for models testing. No new developments will be added, only bugs fixing. After being tested on production for several ESM models, it will be tagged as a stable “xios-3.0” version, targeted for the end of the year 2022.

## IS-ENES3 Deliverable D8.3

### WP8/Task4: XIOS development – PART B: extension of the Dr2xml package

*Reporting period: 01/01/2022 – 31/03/2022*

Authors: Marie-Pierre Moine (CERFACS), Gaëlle Rigoudy (Meteo-France/CNRM)

Reviewer(s): Sophie Valcke (CERFACS)

Release date: 30/09/2022

### ABSTRACT

Dr2xml is a python tool that was intensively used during the CMIP6 exercise, integrated to the IPSL and CNRM-CERFACS modelling workflows to generate the XML configuration files specifying the output variables that the climate model must produce via XIOS. It allowed our two modelling groups to exactly satisfy the CMIP6 Data Request and to respect the standard climate data formats (CF-NetCDF, CMOR), making the model output data directly publishable on the ESGF network. The interest for Dr2xml now goes beyond the purely CMIP context and other communities are starting to express their need to satisfy their own metadata standards, their own nomenclature but also new diagnostic calculation needs. The work conducted for this deliverable has precisely aimed to make the necessary code evolutions to this extension of the scope of use of Dr2xml and to include new physical diagnostics. This went through several steps, from the collection of user needs, the design of a software solution and its coding (that required a reorganization of the code and the development of new interfaces), and finally the implementation in use cases as a demonstrator. This work led to the delivery of a new version dr2xml V3.0 that is available on Github.

| Dissemination Level |   |   |
|---------------------|---|---|
| PU                  | Public  | X |
| CO                  | Confidential, only for the partners of the IS-ENES3 project |   |

| Revision table |            |                                       |   |
|----------------|------------|---------------------------------------|---|
| Version        | Date       | Name                                  | Comments  |
| 1.0            | 30/08/2022 | Marie-Pierre Moine,<br>Gaëlle Rigoudy | Initial version submitted to the reviewer.  |
| 2.0            | 30/09/2022 | Marie-Pierre Moine,<br>Gaëlle Rigoudy | Minor corrections according to reviewer's comments and recommendations. Add the Abstract and the Executive Summary. |

## Table of contents

|  |    |
|--|----|
| 1. Introduction .....                                    | 39 |
| 1.1 Brief presentation of Dr2xml .....                   | 39 |
| 1.2 Extended usages and new needs .....                  | 39 |
| 2 Methodology & Results .....                            | 39 |
| 2.1 Sounding community needs, collecting use cases ..... | 40 |
| 2.2 The proposed software solution .....                 | 41 |
| 3 Conclusions and Recommendations .....                  | 51 |
| References .....   | 51 |

## Executive Summary

The work presented here is the second part of the IS-ENES3 deliverable D8.3: "XIOS development" dedicated to the extension of the scope of Dr2xml to climate modelling projects other than CMIP6. It is linked to the XIOS deliverable because Dr2xml is a tool that heavily relies on XIOS, made to automatically write the so-called "file\_def" XML configuration files. The two parts of this deliverable were carried out separately because they are independent each other and are therefore presented in two separate documents.

The work carried out follows what was described in the DoW and in the successive periodic reports: (i) collect the needs of Dr2xml users from other climate modelling communities (regional climate, seasonal forecast, impacts); (ii) examine the possibilities of code evolutions to take into account these requests; (iii) decide on the restructuring of the code to make it more generic and enlarge the scope of application of the tool; (iv) propose a software solution and (v) illustrate it on a few use cases, for demonstration purposes.

The main novelties brought by the proposed software solution are:

- A generic "project" interface allowing to describe in an exhaustive way the metadata and formats expected in the NetCDF output files;
- A new physical diagnostic requested by the impact and regional modeling community;
- A solution to make the Dr2xml code fully independent from the CMIP6 Data Request.

All these new features are embedded in the V 3.0 of Dr2xml released in June 2022 and available on Github.

## 1. Introduction

### 1.1 Brief presentation of Dr2xml

Dr2xml is a python tool whose role is to automatically writes XIOS XMLs files which configure the model output (the so called “*file\_def*”). The need of such an automatic *file\_def* writer arose with the CMIP6 exercise and its complex data request (Juckes et al. 2019<sup>1</sup>). Thanks to XIOS, it was possible to have exhaustive *file\_def*, including both directives for online computations (e.g. temporal mean or extraction, spatial reduction, regridding, weighting, etc.) and for metadata gluing. The resulting NetCDF files produced by the XIOS-interfaced model then satisfy both the CMOR3 and CF-NetCDF conventions imposed by CMIP6. The main advantage is that model output files are then ready to be published on the ESGF data grid without any additional post-processing step. Dr2xml proved to be a robust key tool that facilitated the contribution to CMIP6 of the two French modelling groups, CNRM-CERFACS and IPSL. Thanks to the XIOS-dr2xml tandem, these two modelling groups were the first one to share their model results on ESGF.

### 1.2 Extended usages and new needs

Although initially designed for the sake of the CMIP6 exercise, Dr2xml is progressively adopted for all current modelling activities in our laboratories: FAIR principles are ensured by the conformance to the CMIP6 standard and Dr2xml avoid tedious hand-writing of the *file\_def* XMLs files. Some request of usability outside the sole CMIP6 framework also started to emerged, in particular in the field area of regional climate modelling and seasonal prediction. Part of the work planned within WP8/Task 4 (“XIOS development - extend the dr2xml package”) was devoted to this objective of enlarging the scope of application of Dr2xml and facilitating some extended usages, so as to benefit widely to other climate related communities, for example impact and climate services.

## 2 Methodology & Results

The work done for Dr2ml extension followed a three-step approach: (i) survey the community needs and identify some use-cases, (ii) propose a code solution, including necessary refactoring and finally (iii) implement the dr2xml solution in the use-cases as a demonstrator.

## 2.1 Sounding community needs, collecting use cases

In 2021, two sessions (March and April respectively) of a 5-day training on XIOS (4.5 days) and Dr2xml (0.5 day) were ensured by IPSL (XIOS part) and CNRM/CERFACS (Dr2xml part), which gathered about 40 participants. In the line of these trainings where people from IS-ENES community had opportunity to discover Dr2xml, the initial plan was to organize a wide consultation to identify the needs, the necessary evolution to enlarge the circle of Dr2xml users by diversifying the application scopes. Due to the COVID-19 pandemic context (the training sessions were exclusively remote which limited human interaction), we decided to start clearing the ground with a limited sub-group, starting with people from CNRM-CERFACS and IPSL laboratories working with non-CMIP6 models (e.g. regional), for different projects (e.g. CORDEX, Copernicus) and focused on different scientific goals (e.g. seasonal prediction). In January 2022, we set up a working group of 12 peoples and a video meeting took place on January 31<sup>st</sup>. After an overview of the different applications (global modelling at IPSL and CNRM, Meteo-France C3S seasonal prediction system, regional climate modelling at CNRM and IPSL) and their solution for I/O management, diagnostic computation and data formatting solutions, emerged the need of a dedicated Dr2xml training. In order to help Dr2xml developers to identify precisely the user's needs and envisage code evolutions, people from the working group were encouraged to describe their use-cases for which they would like to try Dr2xml. Four use-cases were identified, representative of the already mentioned application types: Regional modelling with the AROME atmospheric model, with the ALADIN one, a land-only configuration with SURFEX for C3S\_INIT and with the Copernicus operational system for seasonal prediction.

Users' expectations expressed during the work and training sessions can be grouped in four categories:

- **Metadata:** be able to conform to other metadata standard requested by other non-CMIP6 projects like CORDEX
- **Computing functions:** introduce new diagnostic computation
- **Usability:** simplify installation and usage by removing the dependency to the CMIP6 Data Request package
- **File format:** add the possibility of derogating to the "one variable / one file" CMIP6 rule, writing multiple variables into a single netCDF file

After code adaptation performed to satisfy users' requirements, a new training session was organized on 20 May 2022 where the Dr2xml new features were presented.

These new features are included in Dr2xml V3.0 released in June 2022 and available on GitHub:  
<https://github.com/rigoudyg/dr2xml/releases/tag/V3.0>



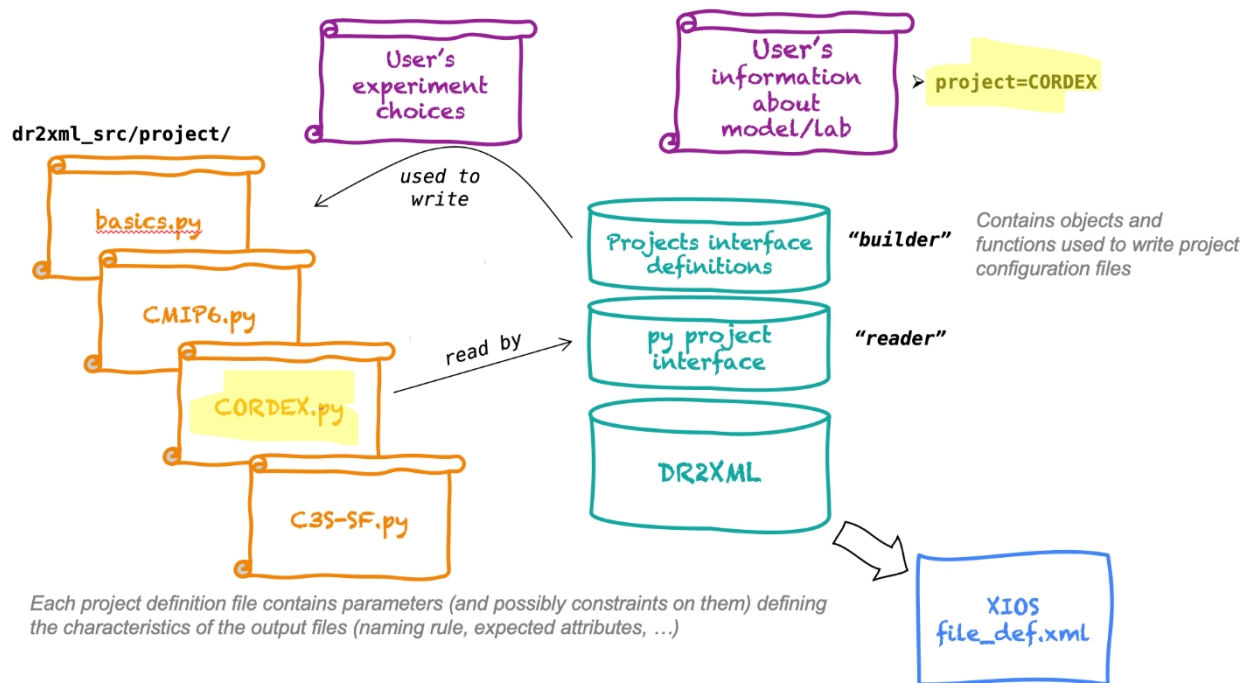
## 2.2 The proposed software solution

In order to meet the four objectives listed above, an important code cleaning and refactoring process has been done.

### (i) Adaptation to other projects' metadata standards

In its previous version, Dr2xml was only designed for the CMIP6 project needs. The CMIP6 standard imposes a set of metadata to be present in the model output files (NetCDF), the possible relationships between these metadata, their status (mandatory or optional) and their dependencies on user's modelling choices (experiment, simulation member, etc.). Consequently, these elements were up to now mainly hard coded in Dr2xml.

The main challenge was to make the metadata customizable, according to the project specifications.



**Figure 1:** Schematic view of the new Dr2xml code structure for its extension to any projects. At top (purple) the classical Dr2xml configuration files where the possibility to select a given project was added; On the right (green), two new interface modules on top of Dr2xml: a "Project definition" module containing the python classes and functions useful for writing the project definition files and a "Project interface" for enabling Dr2xml to read and interpret the project definition files; On the left (orange): the project definition files. A new parameter is introduced in the classical Dr2xml configuration file so as to now be able to select a project framework (the CORDEX example is highlighted in yellow).

To that end, several code evolutions were made:

- A code cleaning was performed in order to remove all CMIP6-supposing pieces of code from the Dr2xml core;
- All metadata specifications are now externalized in the so-called “*project definitions files*” (orange boxes in Fig. 1)
- Two new interface modules were developed:
  - o one ([projects\\_interface\\_definitions.py](#)) containing the basic objects used for building the “project definitions files”. For the sake of simplicity, we will refer to this module as “*the project builder*” in the following;
  - o another one ([py\\_project\\_interface.py](#)) containing the elementary objects necessary for the “project definition file” to be read and interpreted by Dr2xml. This module will be called “*the project reader*”.

The “*project definition files*” contain all the metadata specifications, both for the naming conventions applied to the model output NetCDF files and for their internal attributes. For a given project (e.g. CORDEX.py), the user writes the “*project definition file*” once for all, using the basic objects provided by the “project builder”. A given building object can be used by the other ones or even by itself to define the attributes of these/this latter. There are 6 basic building objects (also called “constructors”):

- **FunctionSettings**: an object to define a function, with the name of the function to call and the values of arguments to pass;
- **ValueSettings**: an object to set the provenance and value of a parameter. Multiple sources are planned (e.g.: Dr2xml configuration/settings file, Data Request, user-defined dictionary, json external file, ...). It is also possible to generate the desired value from several sources, aggregated according to a given format to give the final value;
- **ConditionSettings**: an object to build a checker, defining the value to be checked, the comparison operation to make, with possibly a list for reference values;
- **CaseSetting**: an object for expressing a case test, i.e. implementing a condition test and telling what value to take if the test result is positive;
- **ParameterSettings**: an object for defining the properties of a parameter, for instance, its authorized/forbidden values, default values, condition to satisfy to be an expected NetCDF attribute, etc.;
- **TagSettings**: an object to build XML tag for XIOS configuration files.

To avoid duplication work when writing a “project definition file”, there’s the possibility to inherit from an already existing one. In the project examples written for the sake of the identified use-cases, this option is activated: a low level “project definition file” has been created (basics.py) and the other projects definition files (CMIP6.py, CORDEX.py, C3S-SF.py) inherit from this latter thanks to the key `parent_project_setting="basics"` (see the [projects directory](#) in the Dr2xml source code for these three examples).

To illustrate, let's take a simple example. The CORDEX project's standard imposes to provide information about the global model used to force the regional one. This attribute must be present both in the model output file name, for example here for CNRM\_ALADIN63:

```
tasmin_EUR-11_ECMWF-ERAINT_evaluation_r1i1p1_CNRM-ALADIN63_v1_day_20160101-20181231.nc
```

and in the file internal metadata `model_id`:

```
// global attributes:
      :Conventions = "CF-1.6" ;
      :creation_date = "2018-11-16T13:31:12Z" ;
      :name = "EUR-11_ECMWF-ERAINT_evaluation_r1i1p1_CNRM-ALADIN63_v1_day" ;
      :institute_id = "CNRM" ;
      :model_id = "CNRM-ALADIN63" ;
      :experiment_id = "evaluation" ;
      :experiment = "Evaluation run with reanalysis forcing" ;
      :contact = "contact.aladin-cordex@meteo.fr" ;
      :product = "output" ;
      :Convention = "CF-1.6" ;
      :driving_model_id = "ECMWF-ERAINT" ;
      :driving_model_ensemble_member = "r1i1p1" ;
      :driving_experiment_name = "evaluation" ;
      :driving_experiment = "ERA-INTERIM, evaluation, r1i1p1" ;
      :rcm_version_id = "v1" ;
      :project_id = "CORDEX" ;
      :CORDEX_domain = "EUR-11" ;
      :frequency = "day" ;
      :[...] ;
}
```

The user decides to set this attribute name (`driving_model_id`) and value (ECMWF-ERAINT) in the Dr2xml configuration file `simulation_settings.py`:

```
simulation_settings = {
    [...],
    "driving_model_id": "ECMWF-ERAINT",
```

He now needs to indicate to Dr2xml where to find this attribute and what to do with it, i.e. use it as a file name facet and add it as a NetCDF attribute in the output file. These rules are to be written in the project definition file `CORDEX.py` using the building objects.

First, `driving_model_id` is defined, using the **ParameterSettings** constructor:

```
common_values = dict(
    [...],
    driving_model_id=ParameterSettings(
        key="driving_model_id",
        default_values=[
            ValueSettings(
                key_type="simulation",
                keys="driving_model_id"),
            [...])
```

Its name is set through the key `key` and its default value set using the **ValueSettings** constructor. The provenance of this `driving_model_id` parameter is indicated with `key_type="simulation"`, which means that it is to be read in the *simulation\_settings.py* configuration file. It is stored in a dictionary named `common_values` (N.B: there is 3 possible dictionaries for parameter definitions: `internal_values`, `common_values`, `project_settings`, not discussed further but to be known).

Next step is to write the XIOS *file\_def.xml* configuration file, according to the XIOS syntax. This is done using the **TagSettings** constructor. Instruction must be given to add the `driving_model_id` parameter as a facet of the NetCDF output file name. This is done in “Part 1” as shown in the insert below, using a user-defined file name building function `build_filename` (not shown); `driving_model_id` is passed as an argument of this function using the options keyword.

And finally, the user has to tell that `driving_model_id` is a NetCDF attribute to write in the output file (see Part 2 in the insert below). Firstly, `driving_model_id` has to be added to the list of XIOS variable list, `vars_list` (“variables” in XIOS language correspond to NetCDF attributes). Secondly, the user has to specify in which dictionary `driving_model_id` is defined (here `key_type="common"` means that it is defined in the `common_values` dictionary)

```
file_output=TagSettings(
# Part 1: Defining "driving_model" as a facet of the NetCDF output file
  attrs_constraints=dict(
    name=ParameterSettings(
      key="name",
      default_values=[
        ValueSettings(func=FunctionSettings(
          func=build_filename,
          options=dict( [...],
            driving_model_id=ValueSettings(key_type="common", keys="driving_model_id"),
          )) ],
      fatal=True
    ),
# Part 2: Defining "driving_model" as an attribute of the NetCDF output file
    vars_list=[ [...], "driving_model_id", [... ]
  ],
  [...],
  driving_model_id=ParameterSettings(
    key="driving_model_id",
    default_values=[
      ValueSettings(key_type="common", keys="driving_model_id")
    ],
    fatal=True
  )
)
```

## *(ii) Implementation of a new diagnostic*

In the context of climate change impact studies, the climate services community needs the wind components at certain altitudes to study the evolution of the wind potential in particular. Same demand came from the CORDEX regional modelling program. The version of Dr2xml used for CMIP6 was not able to specify this diagnostic (only vertical interpolation on pressure levels was coded). The XIOS spatial operators are fully capable of performing this vertical interpolation. It was therefore necessary to code this functionality into Dr2xml so that it could automatically write the XIOS instructions to compute this kind of diagnostic.

The problem is the following: The model calculates the wind components at model levels and it is necessary to interpolate them at height levels above the ground. The vertical levels of a climate model do not necessarily follow the topography (this is the case for the CNRM-CM6 model whose vertical coordinate is the hybrid coordinate) and we cannot associate an altitude value to each model level. However, we have access to the geopotential field on each model level and we know the surface altitude (orography). Moreover, XIOS has an interpolation function which allows to choose

the reference field to be used as the target coordinate. This is what we use to specify this new diagnostic “*physical field interpolated on altitude levels*”.

**The first step** is to form this “*reference coordinate*” field (in the XIOS sense) which is the difference between the geopotential (known at each model level) and the orography (surface field). The small subtlety is that an XIOS subtracting operation can only involve fields with the same shapes. It was therefore necessary to use an XIOS trick to transform the 2D orography into a 3D field.

In the model *field\_def* xml files where output model variables are described (in the XIOS sense), the orography is defined on the 2D horizontal grid of the model (FULL\_grid, composed of the horizontal domain FULL, not shown):

```
<field id="orog" grid_ref="FULL_grid" long_name="surface altitude" unit="m" />
```

The trick consists in creating an XIOS pseudo-3D grid (FULL\_grid\_scalar), i.e., 2D with a singleton vertical coordinate:

```
<grid id="FULL_grid_scalar">
  <domain domain_ref="FULL"/>
  <scalar scalar_ref="orog_level" name="orog_level"/>
</grid>
```

Where the scalar *orog\_lev* is defined as follows:

```
<scalar id="orog_level" value="0" axis_type="Z" positive="up" unit="m"/>
```

Then a target 3D grid is defined thanks to the *duplicate\_scalar* XIOS keyword:

```
<grid id="FULL_klev_hlev">
  <domain domain_ref="FULL"/>
  <axis axis_ref="klev">
    <duplicate_scalar/>
  </axis>
</grid>
```

Any pseudo-3D field interpolated on this FULL\_klev\_hlev grid will see its values on the singleton vertical level duplicated on every vertical level of the klev axis which is the vertical model axis. Applied to orography, we obtain an orography field mapped onto the model 3D grid (CMIP6\_rog\_scalar\_duplicate). At each level, the field values are the same (the surface altitude):

```
<field id="CMIP6_rog_scalar_duplicate" field_ref="CMIP6_rog_scalar" grid_ref="FULL_klev_hlev"/>
```

It's now possible to compute the difference between the geopotential (zg) and the orography, that gives the height above ground (height\_over\_rog); i.e. the altitude of each model grid point:

```
<field id="height_over_rog" grid_ref="FULL_klev"> zg - CMIP6_rog_scalar_duplicate </field>
```

height\_over\_rog will serve as reference coordinate for the interpolation of winds on height levels.

**Now**, let's suppose we want to interpolate the zonal wind component at 100 meters. We have to define the target vertical axis, here split in two XIOS axis elements, the first one referring to the second one:

```
<axis id="height_hglev100" axis_ref="height_over_rog_axis" n_glo="1" value="(0,0)[ 100. ]"/>
<axis id="height_over_rog_axis" positive="up" name="z" standard_name="height_over_rog" unit="m">
  <interpolate_axis type="polynomial" order="2" coordinate="height_over_rog"/>
</axis>
```

Then, the 3D grid, made of the 2D horizontal grid and of the 100m vertical axis we have just defined:

```
<grid id="FULL_klev_height_hglev100">
  <domain domain_ref="FULL"/>
  <axis axis_ref="height_hglev100"/>
</grid>
```

The zonal wind at 100m can now be obtained by interpolating the zonal wind on the 3D model grid (CMIP6\_ua) on the grid FULL\_klev\_height\_hglev100:

```
<field id="CMIP6_ua_height100m" field_ref="CMIP6_ua" grid_ref="FULL_klev_height_hglev100"/>
```



The previous detailed steps are the backstage of the coding. Obviously, it is not to the user to code this by hand, Dr2xml offers an interface and automatically generates the appropriate XML tags for XIOS.

A new spatial shape (in the CMIP6 understanding) “XY-HG” is now reserved for 3D fields on height levels and a new syntax allows to ask for such a diagnostic in the VARNAME column of the “home data request”:

```
label_var!label_ref
```

with `label_var` the desired name of the output diagnostic (target variable);  
and `label_ref` the name of variable to interpolate (source variable).

To continue with our zonal wind at 100m example, the line to add in the home data request is:

```
#-----  
#TYPE; VARNAME; REALM; FREQUENCY; TABLE; TEMPORAL_SHP; SPATIAL_SHP; EXPNAME; MIP  
#-----  
extra; ua100m!ua; atmos; 6hr; CNRM_HOMZ6hr; time-point; XY-HG; ANY; ANY
```

The important elements are VARNAME=ua100m!ua and SPATIAL\_SHP=XY-HG. Here we decided to use type “extra” where ua100m properties and metadata are defined (but this is not mandatory, we could have used type “perso” - see the [online Dr2xml documentation](#) for more details).

If using “extra” type, ua100m is declared in the corresponding extra-table (here `CNRM_HOMZ6hr.json`):

```
"ua100m": {
  "frequency": "6hr",
  "modeling_realm": "atmos",
  "standard_name": "eastward_wind",
  "units": "m s-1",
  "cell_methods": "area: time: point",
  "cell_measures": "area: areacella",
  "long_name": "Eastward Wind at 100m",
  "comment": "Zonal wind at 100m height",
  "dimensions": "longitude latitude time height100m",
  "out_name": "ua100m",
  "type": "real",
  "positive": "",
  "valid_min": "",
  "valid_max": "",
  "ok_min_mean_abs": "",
  "ok_max_mean_abs": "",
  "cnrm_priority": "1" }
```

And height100m dimension has to be declared in the *CNRM\_coordinates.json* file:

```
"height100m": {
  "standard_name": "height",
  "units": "m",
  "axis": "Z",
  "long_name": "height",
  "climatology": "",
  "formula": "",
  "must_have_bounds": "no",
  "out_name": "height",
  "positive": "up",
  "requested": "",
  "requested_bounds": "",
  "stored_direction": "increasing",
  "tolerance": "",
  "type": "double",
  "valid_max": "15000.0",
  "valid_min": "0.0",
  "value": "100.",
  "z_bounds_factors": "",
  "z_factors": "",
  "bounds_values": "",
  "generic_level_name": "" }
```

In order to validate the implementation of this new diagnostic in Dr2xml, a comparison between the zonal and meridional wind components interpolated at 25 m and their counterparts on the nearest model level (level 2) was done (cf. Fig.2). Both wind components spatial patterns and order of magnitude are very similar (note that we do not seek for perfect equality), which validates the new diagnostic.

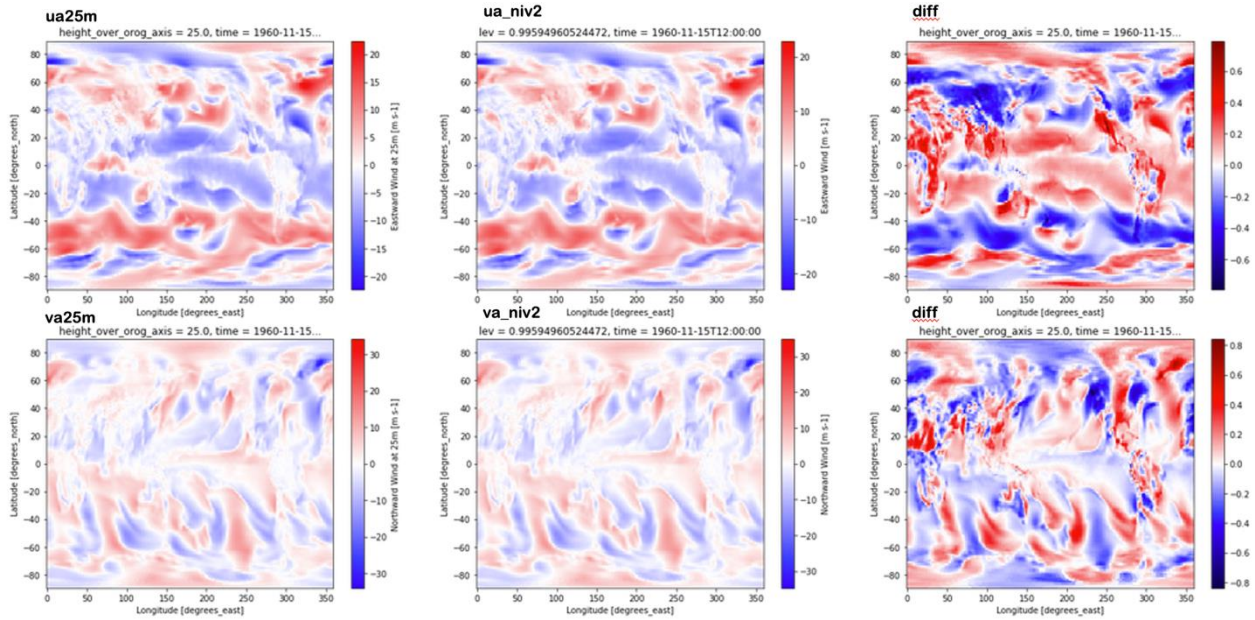


Figure 2: Zonal (upper line) and meridional (bottom line) wind components from the CNRM-CM6-1 model. Results of the implemented diagnostic: wind components interpolated at 25 m (left column); their values on the nearest model level (middle column) and difference between the two (right column).

### (iii) Removing the dependency to the CMIP6 Data Request

Even if Dr2xml configuration options allows to filter out all output variables requested by the CMIP6 Data Request, it is cumbersome and inconvenient to have the Data Request necessarily installed and to have to select a CMIP6 experiment. The objective of this new set of modifications is to provide an interface that completely free of the CMIP6 Data Request (no Data Request or an alternative Data Request) while keeping the “Home Data Request” mechanism.

In the *lab\_and\_model\_settings.py* Dr2xml configuration file, the new key “**data\_request\_used**” is introduced to set the Data Request to use (default value is “CMIP6”, “no” if none). While selecting a Data Request (“DRx”) it activates its associated interface *DRx.py*. So far two DR interface have been coded, *CMIP6.py* and *no.py*. A *definitions.py* routine, where DRx’s main functions are defined (so far only *Scope.py* of the CMIP6 DR) is there to ensure compatibility. All together *DRx.py* and *definitions.py* form the “dr\_interface” new module; the referral to the available version of the DRx requested in the is ensured by *\_\_init\_\_.py*.

### 3 Conclusions and Recommendations

Beyond the CMIP exercises, more and more users are becoming interested in Dr2xml in climate modelling field area, and not only for global modelling. The last evolutions of the tool, described in this WP8/Task4 deliverable, are intended to widen of the scope of application of Dr2xml (regional modelling, seasonal forecasting, etc.). In the initial description of work and in the successive periodic reports, we committed ourselves to (i) collect the needs of Dr2xml users; (ii) examine the possibilities of code evolutions to take into account these requests; (iii) decide on the restructuring of the code to make it more generic and enlarge the scope of application of the tool; (iv) propose a software solution and (v) illustrate it on a few use cases, for demonstration purposes. In the last 6-month progress report (RP3), we identified the short-circuiting of the CMIP6 Data Request as a potential issue, that would be not overcome within the deliverable period. In the end, all these steps were carried out, even including this latter. Only the user consultation was less extensive than expected, not being extended to all potentially interested European laboratories. However, the working method adopted was undoubtedly effective: start with a small circle of users with varied applications and define a software solution on this basis. It is relevant because the needs identified are representative of the wider communities (e.g. the CORDEX project, climate impacts, seasonal forecast) and the proposed software solution is sufficiently generic and modular to be applied to any field of application in climate modelling, whatever the forecasting scales and types of models, global or regional, forced or coupled. Dr2xml new version is fully operational: all the python builders necessary for the application to a project context other than CMIP6 (by "project context", understand "metadata specifications, naming rules, new computing functionalities"). This reshaped version of Dr2xml has now to be widely tested by users who want to take the plunge using Dr2xml to automatically configure XIOS for their model output handling. Their feedbacks will be essential to see if the proposed solution is suitable or not: expected feedbacks are both qualitative (does the proposed solution meet the user's expectations?) and on usability (is the user interface for this Dr2xml extended usage sufficiently understandable?). A good way to answer these questions would be enlarge the circle to other beta-testers, in particular in laboratories who already manifested their interest for Dr2xml (for e.g. at BSC, NCAS/Reading).

### References

1. M. Juckes et al. "The CMIP6 Data Request (version 01.00.31)". In: Geoscientific Model Development Discussions 2019 (2019), pp. 1–35. DOI: 10.5194/gmd- 2019-219. URL: [https://www.geosci-model-dev-discuss.net/gmd- 2019- 219](https://www.geosci-model-dev-discuss.net/gmd-2019-219)