XIOS roadmap
Recent and future developements

XIOS team

IPSL / CEA-LSCE/CERFACS

THE XIOS TEAM

Yushan Wang (IPSL-LSCE)

- Full time XIOS developer
- IS-ENES3 project => end of contract April 2021

Arnaud Caubel (CEA-LSCE)

- Permanent staff
- Integration of XIOS into IPSL model, support, DR2XML management for IPSL-ESM configuration

Yann Meurdesoif (CEA-LSCE)

- Permanent staff
- XIOS developer and manager (30-40% time), support

Marie-Pierre Moine (CERFACS)

- Permanent staff
- XIOS support into ES-ENES service, Integration of XIOS into CNRM model, DR2XML management

Planned team reinforcement

Olga Abramkina (IDRIS computing center / MdLS – Maison de la simulation)

- Starting October 2020
- 30% time on XIOS development

New permanent staff member (CEA-LSCE)

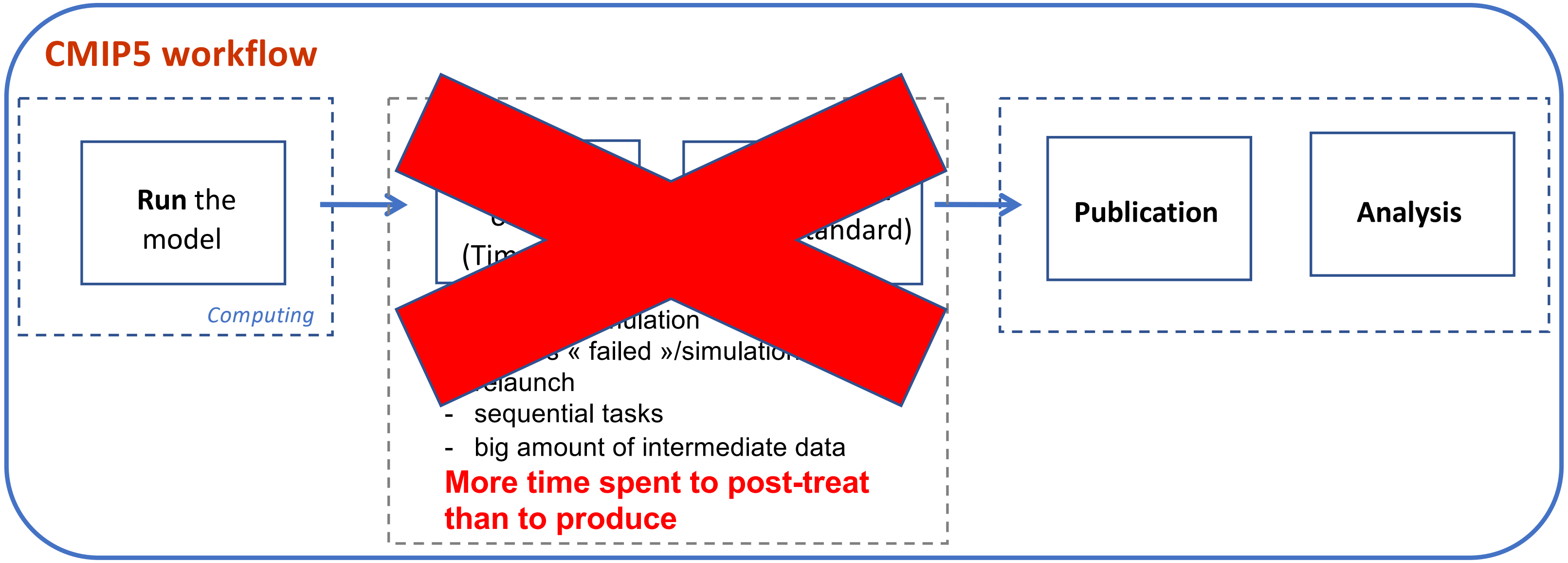
- Before end 2020
- Full time XIOS development and support

New fixed-term contract recruitment (~24mo, ESIWACE2)

- Starting beginning 2021
- Full time XIOS development and support

Post-CMIP6 developments

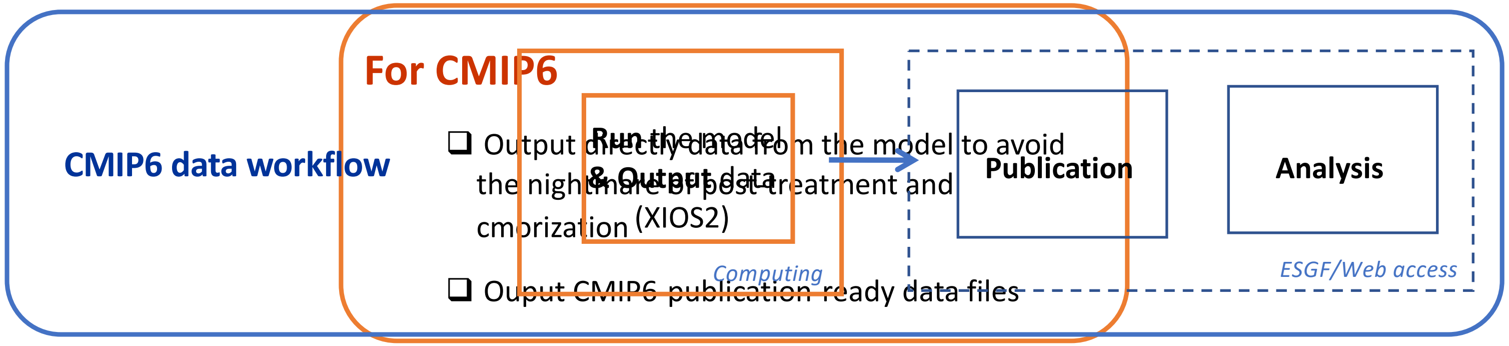
CMIP6 workflow : whole post-processing done by XIOS before write data



CMIP5 workflow

ulation
s « failed »/simulatio
elaunch
- sequential tasks
- big amount of intermediate data

More time spent to post-treat than to produce



For CMIP6

- ❑ Output directly data from the model to avoid the nightmare of post-treatment and cmorization
- ❑ Ouput CMIP6 publication-ready data files

Post-CMIP6 developments

Great functionalities, great success but…

Some painful lessons learned from many years of intense development:

- A lot codes lines (~120 000), more and more difficult to control
- Loosing experience and code knowledge when non-permanent staff leave
- Code infrastructure is in a poor condition
- When fixing bugs, strong uncontrolled side effects => slow down development
- Difficult for users to debug XIOS workflow when error is rising
- Non negligible impact onto model performance
- Difficulty reach high scalability for high resolution runs
- Huge memory consumption that doesn't go at scale
- Lack of flexibility of the client-server infrastructure that inhibits new developments

So we decided to freeze planed developments to focus first on robustness and reliability

Robustness and reliability improvement

Improve XIOS error diagnostics

- In case of error, full stack is now output by the exception manager
- Full information (attribute) of the concerned object (field, file, etc…) is output all along the stack

✓ *Done in 2019*

```
In file "field.cpp", function "void xios::CField::solveGridReference()",  line 1605 -> Field 'field2D' has both a
grid and a domain/axis/scalar.
Please define either 'grid_ref' or 'domain_ref'/'axis_ref'/'scalar_ref'.

(1) **************** void cxios_context_close_definition()

(2) **************** void xios::CContext::closeDefinition()
Object id="atm" object type="context"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[]
*** Additional information:
[enabled files="atm_ensemble "]

(3) **************** void xios::CContext::postProcessingGlobalAttributes()
Object id="atm" object type="context"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[]
*** Additional information:
[enabled files="atm_ensemble "]

(4) **************** void xios::CContext::postProcessing()
Object id="atm" object type="context"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[]
*** Additional information:
[enabled files="atm_ensemble "]

(5) **************** void xios::CContext::solveOnlyRefOfEnabledFields(bool)
Object id="atm" object type="context"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[]
*** Additional information:
[enabled files="atm_ensemble "]
```

```
(6) **************** void xios::CFile::solveOnlyRefOfEnabledFields(bool)
Object id="atm_ensemble" object type="file"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[append="true" enabled="true" output_freq="1ts" type="one_file" ]
*** Additional information:
[context="atm" enabled fields="__field_undef_id_0 "]

(7) **************** void xios::CField::solveOnlyReferenceEnabledField(bool)
Object id="__field_undef_id_0" object type="field"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[axis_ref="axis_ensemble" default_value="1e+20" detect_missing_value="true" domain_ref="domain" enabled="true"
field_ref="field2D" freq_op="1ts" grid_ref="grid3d" level="1" name="field2D" operation="instant" prec="8" ]
*** Additional information:
[]

(8) **************** void xios::CField::solveGridReference()
Object id="__field_undef_id_0" object type="field"
*** XIOS attributes as defined in XML file(s) or via Fortran interface:
[axis_ref="axis_ensemble" default_value="1e+20" detect_missing_value="true" domain_ref="domain" enabled="true"
field_ref="field2D" freq_op="1ts" grid_ref="grid3d" level="1" name="field2D" operation="instant" prec="8" ]
*** Additional information:
[]

    File                 Function                                              Line
(8) field.cpp              void xios::CField::solveGridReference()                       1594
(7) field.cpp              void xios::CField::solveOnlyReferenceEnabledField(bool)                978
(6) file.cpp             void xios::CFile::solveOnlyRefOfEnabledFields(bool)                  834
(5) context.cpp              void xios::CContext::solveOnlyRefOfEnabledFields(bool)                808
(4) context.cpp              void xios::CContext::postProcessing()                          1547
(3) context.cpp              void xios::CContext::postProcessingGlobalAttributes()                  579
(2) context.cpp              void xios::CContext::closeDefinition()                         701
(1) icdata.cpp             void cxios_context_close_definition()                          117
```

Robustness and reliability improvement

Performance profiling logs

- Implementation of "easy to use" timer class in XIOS
- Detailed performance and memory information to well understand bottleneck
- Logs generated at the end of the job

✓ *Done in 2019*

```
-> info :  CContextServer: Receive context <atm> finalize.
-> report :  Memory report : Context <atm> : server side : memory used for buffer of each connection to client
  +) With client of rank 0 : 10000000 bytes
-> report :  Memory report : Context <atm> : server side : total memory used for buffer 10000000 bytes
-> report :  Memory report : Context <atm_server> : client side : memory used for buffer of each connection to server
  +) To server with rank 0 : 10000000 bytes
-> report :  Memory report : Context <atm_server> : client side : total memory used for buffer 10000000 bytes
-> info : Closing File : atm_ensemble
-> info : CContext: Context <atm_server> is finalized.
-> report :  Memory report : Context <atm_server> : server side : memory used for buffer of each connection to client
  +) With client of rank 0 : 10000000 bytes
-> report :  Memory report : Context <atm_server> : server side : total memory used for buffer 10000000 bytes
-> report :  Memory report : Context <atm> : client side : memory used for buffer of each connection to server
  +) To server with rank 0 : 10000000 bytes
-> report :  Memory report : Context <atm> : client side : total memory used for buffer 10000000 bytes
-> info : CContext: Context <atm> is finalized.
-> info : Client side context is finalized
-> report :  Performance report : Whole time from XIOS init and finalize: 0.359765 s
-> report :  Performance report : total time spent for XIOS : 0.327634 s
-> report :  Performance report : time spent for waiting free buffer : 9.28231e-05 s
-> report :  Performance report : Ratio : 0.025801 %
-> report :  Performance report : This ratio must be close to zero. Otherwise it may be usefull to increase buffer size or
numbers of server
-> report :  Memory report : Minimum buffer size required : 5267 bytes
-> 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
```

```
-> report : Timer : Blocking time    -->   cumulated time : 9.28231e-05
Timer : Context : close definition    -->   cumulated time : 0.138143
Timer : Field : recv data    -->   cumulated time : 0.026445
Timer : Field : send data    -->   cumulated time : 0.03075
Timer : Files : close    -->   cumulated time : 0.00188847
Timer : Files : create headers    -->   cumulated time : 0.00926207
Timer : Files : get data infos    -->   cumulated time : 0.000444779
Timer : Files : open    -->   cumulated time : 0.00804241
Timer : Files : writing data    -->   cumulated time : 0.00190237
Timer : Files : writing time axis    -->   cumulated time : 0.0018695
Timer : Process events    -->   cumulated time : 0.03559
Timer : Process request    -->   cumulated time : 0.000316099
Timer : XIOS    -->   cumulated time : 0.327634
Timer : XIOS close definition    -->   cumulated time : 0.139521
Timer : XIOS context finalize    -->   cumulated time : 0.00226334
Timer : XIOS finalize    -->   cumulated time : 0
Timer : XIOS get variable data    -->   cumulated time : 0.000234546
Timer : XIOS init    -->   cumulated time : 0
Timer : XIOS init context    -->   cumulated time : 0.00179636
Timer : XIOS init/finalize    -->   cumulated time : 0.359765
Timer : XIOS send field    -->   cumulated time : 0.176479
```

Robustness and reliability improvement

Output and visualize XIOS workflow graph

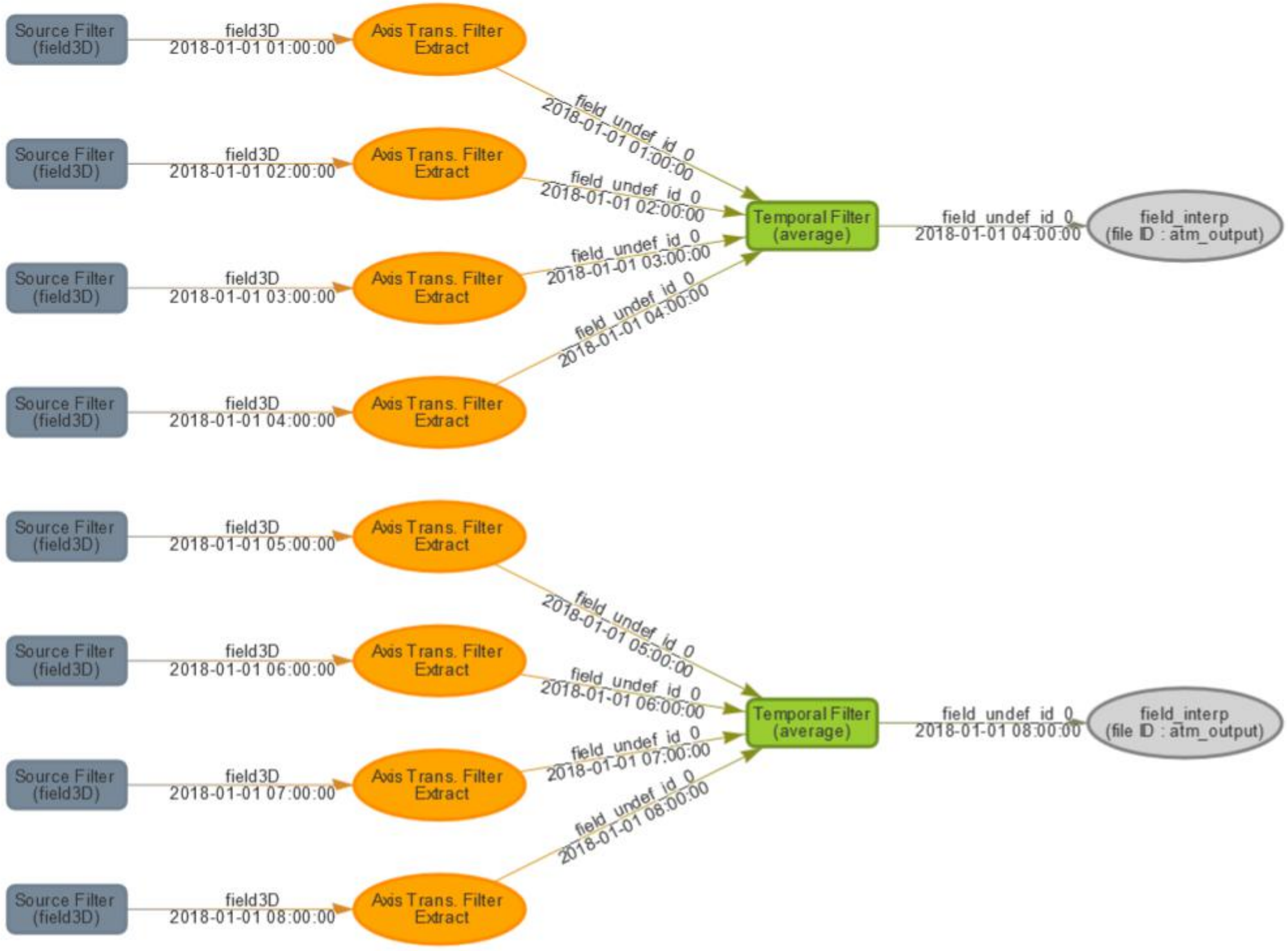
✓ *Done end 2019*

- Graphical view of spatial and temporal chained graph composing XIOS workflow
- Visualization within a standard web navigator
- Very useful to understand or debug workflow written in XML
- Time line is also manage
  - Can see if some are not well connected following the timestamp
- Very easy to use : one attribute to add on one or more field
  - All prerequisite or dependency of the field will be output

```
<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>
```

- Possibility of reducing graphs amounts by filtering over time periods"
  - "build_start_graph" and "build_end_graph" field attributes.
- Graphs generated at the end of execution trough a Jason file
- Can be loaded and visualize using online tool on standard navigator
  - http://forge.ipsl.jussieu.fr/ioserver/chrome/site/XIOS_TEST_SUITE/graph.html

Robustness and reliability improvement



Source Filter (field3D)
field3D 2018-01-01 01:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 01:00:00
Source Filter (field3D)
field3D 2018-01-01 02:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 02:00:00
Source Filter (field3D)
field3D 2018-01-01 03:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 03:00:00
Source Filter (field3D)
field3D 2018-01-01 04:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 04:00:00
Temporal Filter (average)
field_undef_id_0 2018-01-01 04:00:00
field_interp (file ID : atm_output)
Source Filter (field3D)
field3D 2018-01-01 05:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 05:00:00
Source Filter (field3D)
field3D 2018-01-01 06:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 06:00:00
Source Filter (field3D)
field3D 2018-01-01 07:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 07:00:00
Source Filter (field3D)
field3D 2018-01-01 08:00:00
Axis Trans. Filter Extract
field_undef_id_0 2018-01-01 08:00:00
Temporal Filter (average)
field_undef_id_0 2018-01-01 08:00:00
field_interp (file ID : atm_output)

Robustness and reliability improvement

Development of a test case suite for contiguous integration

✓ *finalized mid-2020*

- Build a generic test case (binary) that can handle all XIOS functionalities:
  - Test all kind of mesh, including mesh indexation and mask
  - Test for fields on scalar, 1-D, 2-D, 3D or 4-D grid
- Run is defined by a set of parameters list
  - Nb models, nb proc for client, nb proc for servers, selected mesh
- Tested functionalities are defined by a set of XML files
- All test case suite will be declined in unitary test and automated after each commit on different supercomputers
  - Compilation is also tested
- Results and regressions are exposed through a navigator

Param.def

```
&params_run
duration='1d'
nb_proc_atm=10
nb_proc_oce=5
nb_proc_surf=1
/
```

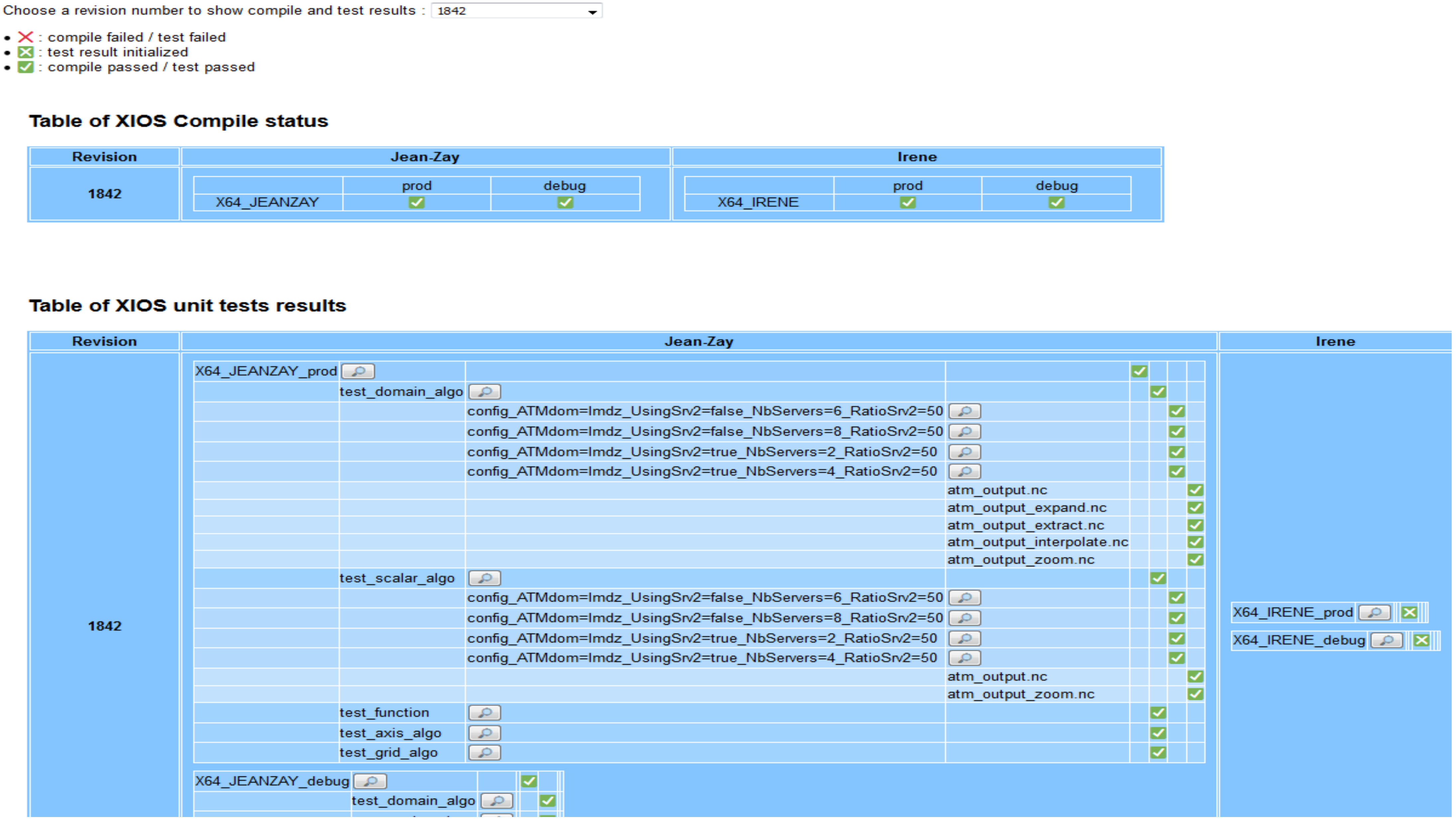
iodef.xml

```
<context id="atm">
   <variable_definition>

     <variable id="timestep"> 1h </variable>
     <variable id="domain"> lmdz </variable>
     <variable id="domain_mask"> true </variable>
     <variable id="axis_mask"> false </variable>
     <variable id="init_field2D"> academic </variable>
     <variable id="ni"> 36 </variable>
     <variable id="nj"> 18 </variable>
     <variable id="nlev"> 10 </variable>
     <variable id="pressure_factor"> 0.10 </variable>
     <variable id="mask3d"> false </variable>
     <variable id="domain_proc_frac">3</variable>
     <variable id="axis_proc_frac">2</variable>
     <variable id="axis_proc_n">2</variable>
     <variable id="ensemble_proc_n">2</variable>

<variable id="other_domain"> arpege </variable>
      <variable id="other_domain_mask"> false </variable>
      <variable id="other_axis_mask"> false </variable>
      <variable id="other_init_field2D"> rank </variable>
      <variable id="other_ni"> 36 </variable>
      <variable id="other_nj"> 18 </variable>
      <variable id="other_nlev"> 10 </variable>
      <variable id="other_pressure_factor"> 0.10 </variable>
      <variable id="other_mask3d"> false </variable>
      <variable id="other_domain_proc_frac">3</variable>
      <variable id="other_axis_proc_frac">2</variable>
      <variable id="other_axis_proc_n">2</variable>
      <variable id="other_ensemble_proc_n">2</variable>

    </variable_definition>
  </context>
```



Choose a revision number to show compile and test results : 1842

- ✗ : compile failed / test failed
- ☒ : test result initialized
- ✓ : compile passed / test passed

Table of XIOS Compile status

Revision	Jean-Zay			Irene		
		prod	debug		prod	debug
1842	X64_JEANZAY	✓	✓	X64_IRENE	✓	✓

Table of XIOS unit tests results

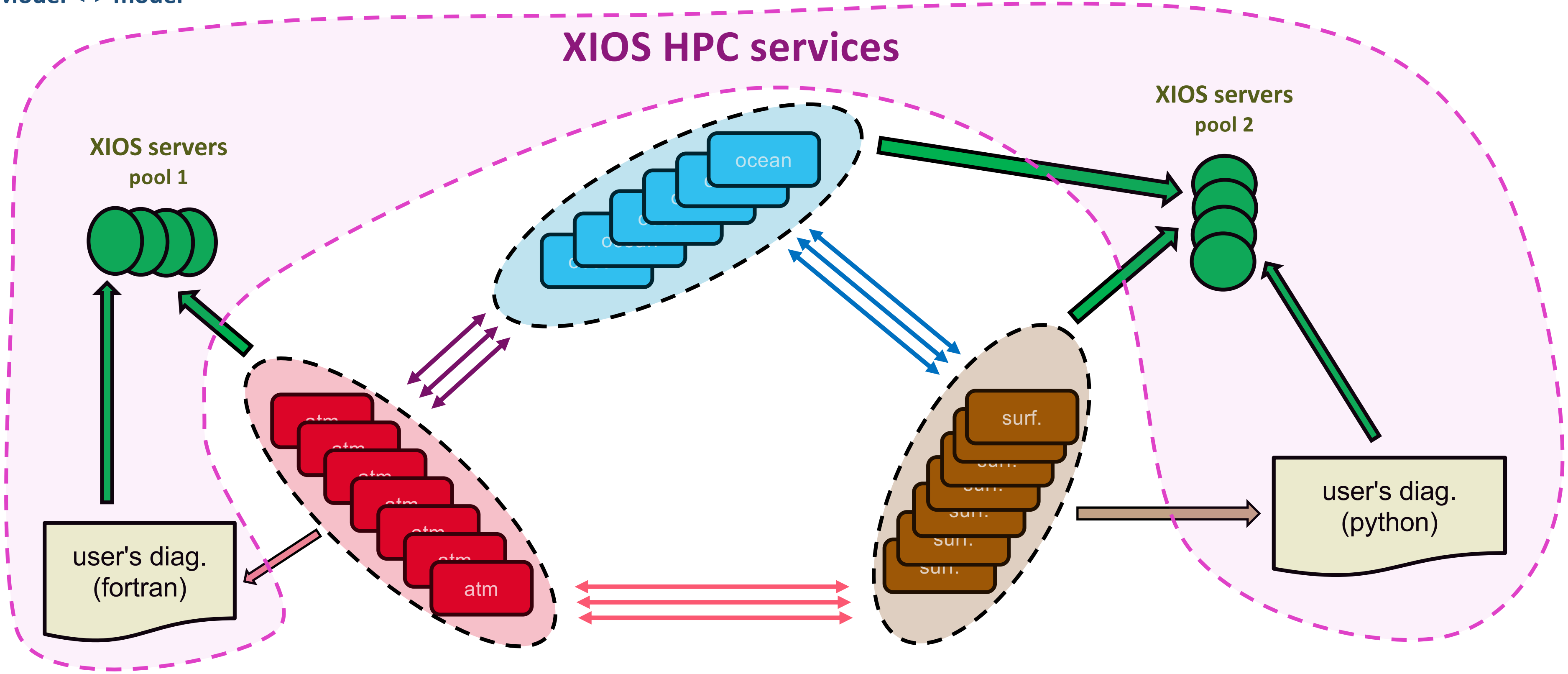
Revision	Jean-Zay								Irene
1842	X64_JEANZAY_prod 🔍				✓				X64_IRENE_prod 🔍 ☒ X64_IRENE_debug 🔍 ☒
		test_domain_algo 🔍				✓			
			config_ATMdom=lmdz_UsingSrv2=false_NbServers=6_RatioSrv2=50 🔍				✓		
			config_ATMdom=lmdz_UsingSrv2=false_NbServers=8_RatioSrv2=50 🔍				✓		
			config_ATMdom=lmdz_UsingSrv2=true_NbServers=2_RatioSrv2=50 🔍				✓		
			config_ATMdom=lmdz_UsingSrv2=true_NbServers=4_RatioSrv2=50 🔍				✓		
				atm_output.nc				✓	
				atm_output_expand.nc				✓	
				atm_output_extract.nc				✓	
				atm_output_interpolate.nc				✓	
				atm_output_zoom.nc				✓	
		test_scalar_algo 🔍				✓			
			config_ATMdom=lmdz_UsingSrv2=false_NbServers=6_RatioSrv2=50 🔍				✓		
			config_ATMdom=lmdz_UsingSrv2=false_NbServers=8_RatioSrv2=50 🔍				✓		
			config_ATMdom=lmdz_UsingSrv2=true_NbServers=2_RatioSrv2=50 🔍				✓		
			config_ATMdom=lmdz_UsingSrv2=true_NbServers=4_RatioSrv2=50 🔍				✓		
				atm_output.nc				✓	
				atm_output_zoom.nc				✓	
		test_function 🔍				✓			
		test_axis_algo 🔍				✓			
		test_grid_algo 🔍				✓			
	X64_JEANZAY_debug 🔍				✓				
		test_domain_algo 🔍				✓			

TOWARD A NEW XIOS INFRASTRUCTURE OF HPC SERVICES

HPC services can be launch into a pool of dedicated resources (free CPU processes) at any time

Universal way to exchange data flux between :

- Model <-> services
- Services <-> services
- Model <-> model



TOWARD A NEW XIOS INFRASTRUCTURE OF HPC SERVICES

What could be an XIOS service ?

- A specific services provided by XIOS
  - Current I/O servers level 1 or 2 (reader, writer, gatherer)
  - Future specific services (ensemble management, IA management, in situ visualization…)
- A piece of XML workflow
  - Automatic offload of costly diagnostics computed asynchronously onto dedicated resources
- A service written by users
  - In fortran using standard XIOS interface
  - In future, in python
    - Need to develop an XIOS python interface in a similar way than in Fortran
  - These kind of services can be see as a "light way coupling", the service is comparable to a small model.

How will be manage the data flux exchange (model<->model or model<->user services<->xios service) ?

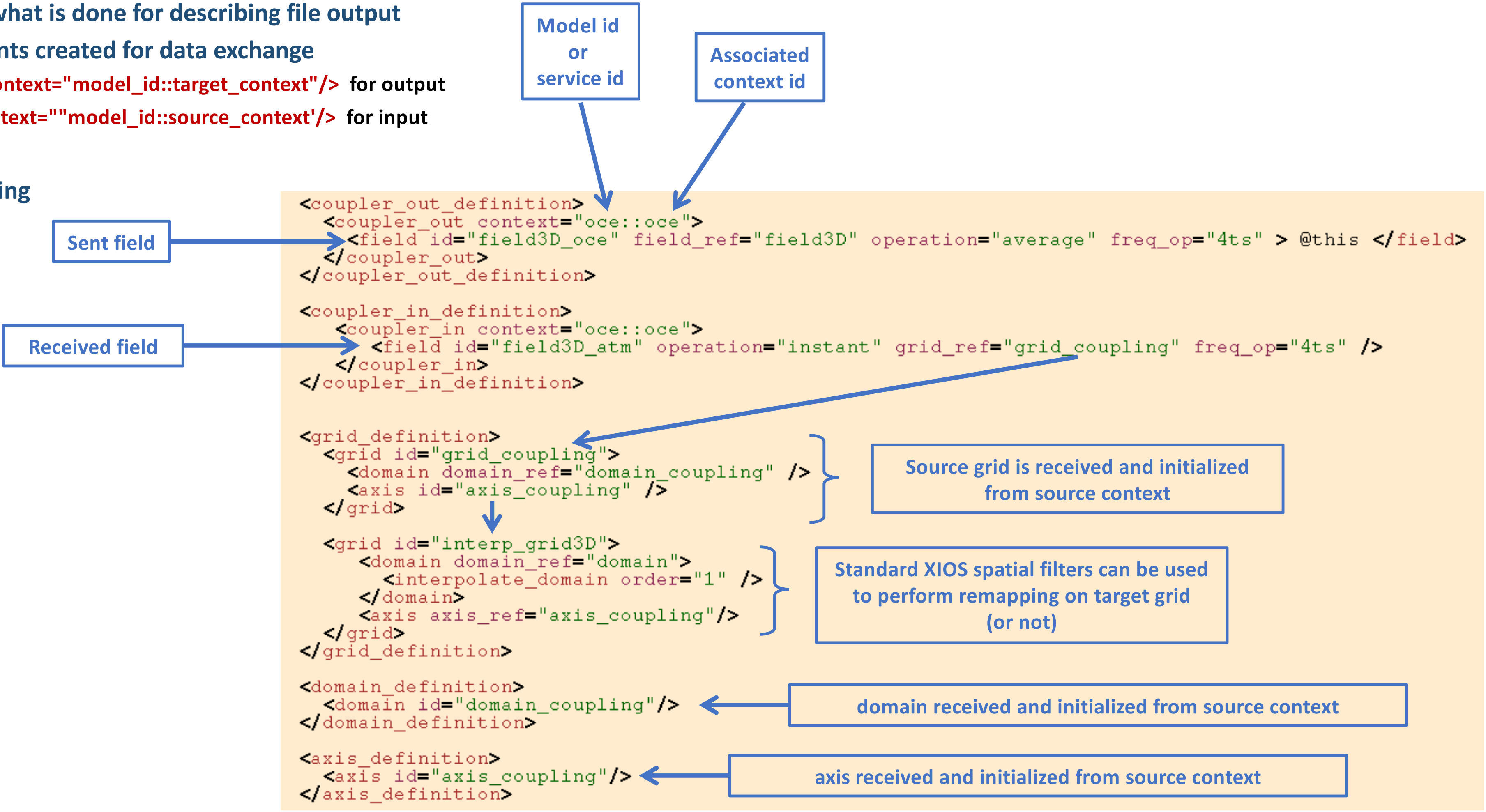
- Interface (for model or user written service interface)
  - We decide to keep the most simple interface which is the current standard one
  - To send data flux
    - CALL xios_send_field("field_id", field)
  - To receive data flux
    - CALL xios_recv_field("field_id", field)

TOWARD A NEW XIOS INFRASTRUCTURE OF HPC SERVICES

From XML

- Very similar of what is done for describing file output
- Two new elements created for data exchange
  - <coupler_out context="model_id::target_context"/> for output
  - <coupler_in context=""model_id::source_context'/> for input

- Ex : 2 way coupling



```
<coupler_out_definition>
  <coupler_out context="oce::oce">
    <field id="field3D_oce" field_ref="field3D" operation="average" freq_op="4ts" > @this </field>
  </coupler_out>
</coupler_out_definition>

<coupler_in_definition>
   <coupler_in context="oce::oce">
     <field id="field3D_atm" operation="instant" grid_ref="grid_coupling" freq_op="4ts" />
   </coupler_in>
</coupler_in_definition>


<grid_definition>
  <grid id="grid_coupling">
    <domain domain_ref="domain_coupling" />
    <axis id="axis_coupling" />
  </grid>

  <grid id="interp_grid3D">
     <domain domain_ref="domain">
       <interpolate_domain order="1" />
     </domain>
     <axis axis_ref="axis_coupling"/>
  </grid>
</grid_definition>

<domain_definition>
  <domain id="domain_coupling"/>
</domain_definition>

<axis_definition>
  <axis id="axis_coupling"/>
</axis_definition>
```

Model id or service id

Associated context id

Sent field

Received field

Source grid is received and initialized from source context

Standard XIOS spatial filters can be used to perform remapping on target grid (or not)

domain received and initialized from source context

axis received and initialized from source context

Now, the spring cleaning period

✓ finalized mid-2020

Major XIOS core rewriting, begun more than one years ago

Dev branches : XIOS_ONE_SIDED -> XIOS_SERVICE -> XIOS_COUPLING

- ~ 70 commit
- ~ 40 000 code lines added, deleted or moved
- Merging with trunk targeted beginning 2021

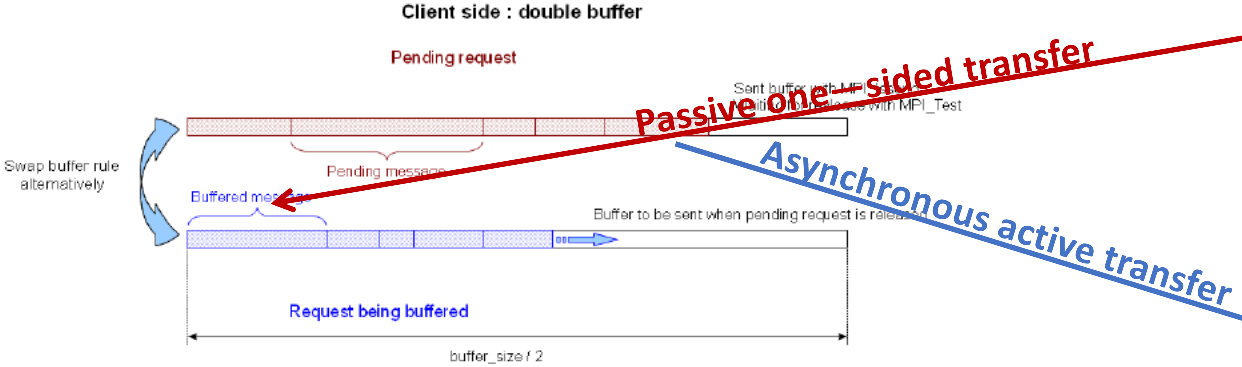
GOALS

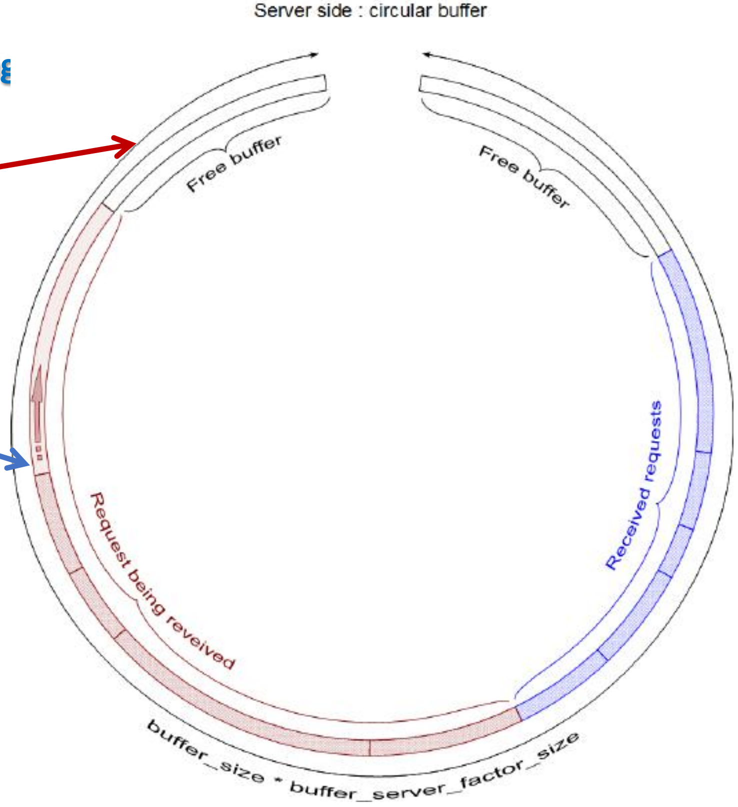
- Regaining control over 10 years of eclectic development
- Cleaning code and rationalizing internal concept
- Improving performance in order to be prepared at exascale area and high resolution modeling : global 10 km - 1 km
  - Improve transfer protocol
  - Improve workflow computing performance
  - Improve I/O performances
- Reducing memory footprint
  - Huge memory consumption at scale
- Introducing new infrastructure of services
- Implementing code coupling and unify data exchange protocol between models and services

Now, the spring cleaning period

Improving transfer protocol

- **Current protocol transfer asynchronously data from client to server using buffering**
  - **Using active transfer protocol : MPI_Isend, MPI_Irecv, MPI_Test**


Client side : double buffer
Pending request
Sent buffer with MPI_Isend
Waiting for release with MPI_Test
Swap buffer rule alternatively
Pending message
Buffered message
Buffer to be sent when pending request is released
Request being buffered
buffer_size / 2
Passive one–sided transfer
Asynchronous active transfer


Server side : circular buffer
Free buffer
Free buffer
Received requests
Request being reveived
buffer_size * buffer_server_factor_size

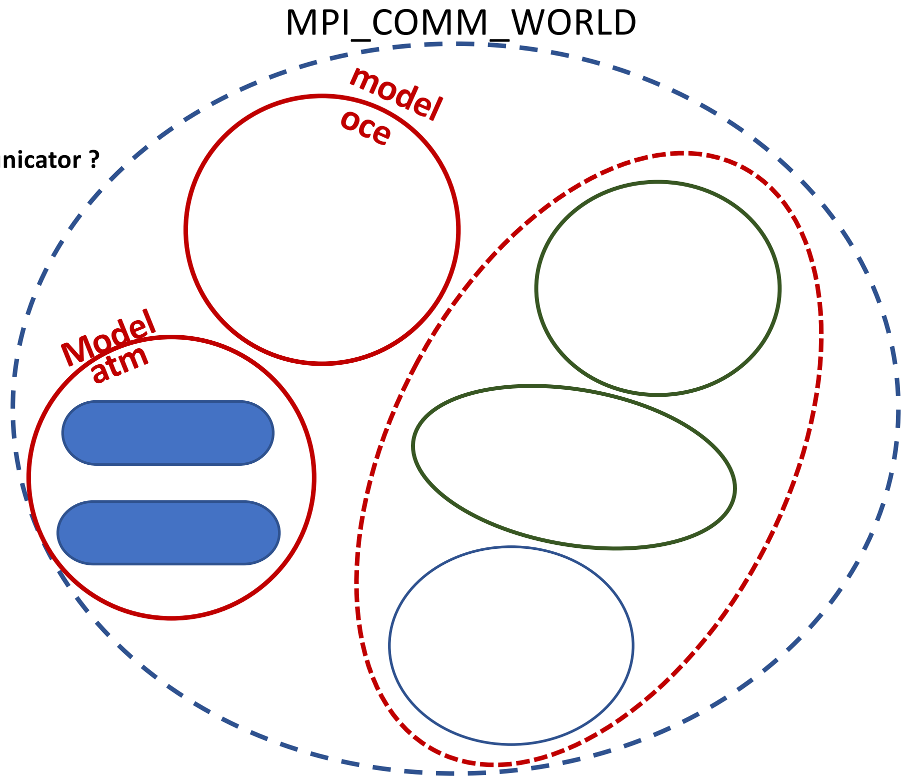
- **But in some (rare) situation this protocol may lead to dead-lock**
  - o Complex interaction due to limitation of buffer size, between client that can wait other where in the code and servers that are waiting for an event.
  - o These dead-lock can be overcome by limiting the number of event stored in client side, even if not full.
  - o Large impact on performance in some case, because this number can be small.

✓ *Done end-2019*

- **We have now introduce part of passive one sided-communication (MPI_put/MPI_get) on server side**
  - o In case of dead-lock, servers can access to the data stored in client buffer using passive MPI communication
  - o The limitation on the maximum number of stored event can be removed

Development of new infrastructure for XIOS services

- Developing a resource manager
  - Where are free resources, and allocate them to a service
- Developing a service manager
  - Launch services into allocated resource,
  - Manage event loop and wait for context registration
- Developing a context manager
  - Create a context inside service, manage the associated event loop
- Developing a name service
  - Where are services and associated context, where are models in the MPI_COMM_WORLD communicator ?
  - Retrieve inter-communicator between 2 contexts, living in given services



Current XIOS functionalities have been rewrote in such infrastructure

- Server level 1 : gathering service
- Server level 2 : I/O writer service
- Each services are interconnected and can exchange data

What can be done more easily now

- Dedicated I/O servers for each model
- Offloading of XML workflow
- Code coupling
- Future XIOS services

✓ *Done first Quarter-2020*

Development of coupling functionalities

- MPI inter-communicator between models created on the fly thanks to Name Service
- Need to transfer grid from source context to targeted context
- Need to manage the graph dependency of the new coupling grid to build the XIOS workflow
- In case of 2 way coupling (or more), need to schedule and synchronize grid sending to avoid dead-lock
- Flux transfer reuse the data file transfer protocol between clients and servers

✓ *Done mid-2020*

First 2-way coupling test case achieved mid-2020 !

Reducing the memory footprint

- Large amount of memory is used for array of index
- Indexation is used to transfer field data :
  - From model to workflow
  - For computing workflow transformation
  - From client to server
  - For file writing or reading
- In past XIOS versions, array of index are commensurable to the size of the grid
- Reason is grid masking (3D masking for example) induce relationship between domains and axes composing the grid

So we removed the grid masking

- We keep the functionality, but grid masked value are replace by NaN value, and computations are done on them into XIOS workflow
- Domains and axes masking remain unchanged

We can use the tensor product properties to compute the transfer

- Only keep indexes for domains and axes
- Ex : grid4D = domain2D ⊗ axis1D ⊗ axis1D
- Grid4D = 200 x 200 x 100 x 50 = 200 000 000 indexes
- Now : domain2D (200 x 200) + axis1D (100) + axis1D (50) = 40 150 indexes => reduction of a factor ~ 5000

Large impact on memory footprint and computational performance is expected

- Less memory access => higher computational performance

Huge rewrite of whole transfer filters

- New objects created : the "connectors"
- Replace the current grid indexation by recursive inlined transfer methods using tensor product properties
- All intensive computation is now concentrated into connectors and filters
  - Small part of the whole code
  - More easy in future to work on performance optimization
  - Facilitate future implementation of OpenMP parallelism or GPU porting

Source filters and terminal filters are now up to date

- Model -> workflow, workflow -> model
- Client workflow -> server workflow, server workflow -> client workflow
- Worklow -> file writing, file reading ->workflow

✓ *Done third quarter-2020*

Remain to rewrite the spatial transformation filters

- Work targeted before end 2020

Targeted end 2020

Merging with trunk targeted beginning 2021

Stable version expected at mid-2021

Targeted mid-2021

Planed development for new functionalities

Urgent NEMO consortium request for XIOS supporting tiling

- Improve NEMO performance using cache blocking mechanism
- Will be implemented on the trunk in a light way for fast reply
- Will be generalized in the current dev. version
  - More easy to manage tiling within connectors
- First demonstrator expected November 2020

Targeted november 2020

Implementing XIOS restartability

- Currently XIOS is not restartable
  - Model can be stop only at a multiple of the highest frequency of the time filters (averaging)
- Will enable models and XIOS workflow to be shut down at any time and then restarted
  - Longer averaging frequency (yearly means)
  - Decadal seasonal means
- Restartability is also a requirement for model coupling

Targeted first quarter 2021

Planed development for new functionalities

Improvement of the internal time line management

- **Implementing time interpolations**
  - Remove current limitation : temporal filters are applied at a multiple frequency of model time step
  - Time interpolation filter will uncoupled the XIOS workflow from the models time step.
  - A lot of practical examples…
    - Enable models with variable time step
    - For reading, a monthly file can be interpolated daily before to be injected into model

Targeted end 2021

Improving spatial filters

- **Implement more complex spatial filters by chaining internally already developed primary filters**
  - Zonal means, grad , div and curl filters…
  - Exemple : zonal mean : 3 chained elementary filters
    - Interpolation toward a regular mesh
    - Local reduction over the longitude
    - Global reduction over the longitude

Targeted end 2020

- **Efficient station output management**
  - Currently done using interpolation, performance killer…
- **Implement still missing remapping operator**
  - 2nd order with slope limiters => conserving extrema
  - Nearest neighbors ?

Targeted mid 2022

On HPC side…

More and more cores available by nodes

- GPU offloading keep a lot of unused cores on hosts

Why not asynchronize the xios client part (the xios workflow) ?

- Past attempts unsuccessful because MPI transfer was a performance killer…
- But new MPI-3 functionalities make more easy the MPI transfer in shared memory
  - One sided (MPI_Put/MPI_get) passive communication in shared memory

Proposal : dedicated XIOS process on models node to compute workflow

- Objectives : 0 cost for models.
- Data transferred in shared memory by xios client with passsive MPI_get
- Workflow will run asynchronously.
- Overlap model computation and workflow computation.
- But synchronization still needed if model required data from client (file reading, coupling)

Make a proof of concept to evaluate the potential performance gains

- Targeted end-2021

Targeted end-2021

On HPC Side…

About OpenMP ?

- XIOS not multithreaded is a huge potential bottleneck

- Previous attempt using MPI_Endpoint technology was not so successful
  - Elegant and no invasive approach which was working well
  - But with no conclusive gains in performance for small domain
  - MPI latency in MPI_THREAD_MULTIPLE mode compensate the gains due to multithreading
  - Will not probably be part of future MPI-4 standard

- Explore other way to exploit multithreadism ?
  - Parallelize explicitly filters with fork and join model ?
    - More easy in new infrastructure
  - Exploit OpenMP 3 tasking
    - The workflow graph can be browsed concurrently by several tasks
    - Independent workflow branches can be computed concurrently

Exploring work planed starting end - 2021

On HPC side…

What about GPU computing ?

- Not a lot of ESM models are running onto GPU for now
- Difficult problem, internal structure in C++ is not convenient for openACC or OpenMP-GPU porting
- But new infrastructure make it more easy
- Try to port individually each filters onto GPU
  - Similar to OpenMP fork and join approach
  - Progressive approach

Sustainable alternative can be overlapping GPU model computation by dedicated XIOS client processes

- Or a mixed of the two approaches

Exploring work can be targeted mid-2022 or beginning 2023

Targeted 2022-2023 ?

New services

What about future services in new infrastructure ?

- **Grenville will present works done on ensemble output, managed by XIOS in next talk…**
  - Similar project at IPSL now
  - Works, but suffer of a lot of constraints
  - All members must run simultaneously in same global MPI communicator
  - Reduce the XIOS and models efficiency due to implicit synchronizations between members
  - If one member falls, difficult to get an efficient fault tolerance management

- **The proposal is to develop a dedicated service for ensemble management**
  - Models members may run independently of each other in their own local communicator
    - No code change for ensemble management
  - They may connect dynamically to the ensemble service in a similar way than for XIOS file server
  - The ensemble service collect data from each member and can store internally data until all members have run a given timestep
    - Use local disk storage for buffering
  - Once data is collected from every member, make local reduction : ensemble averaging, standard deviation, etc. before sending to I/O writer service
  - Ensemble service must be restartable
  - More easy in future to ensure fault tolerance, since we just need to invalidate communicator of fallen member
  - Fallen member can be rerun independently later

Targeted 2021-2022

New services

Is new AI service can be useful ?

- XIOS is a "windows" onto the models
  - Knowledge of data exported and of the associated mesh
  - Easy to develop to specific service to
    - Export data from model to train a neural network
    - Export data from model for inference and reimport data from the trained neural network
  - Neural network will be trained or inferred "in Situ"

- Can be also built as an "user service"
  - Need to develop a python interface for XIOS to make more easy the connection with the AI world

Is a "in situ" visualization service can be useful ?

- Data received by servers can be directly visualized "in situ" instead to be wrote in files
- C++ make easy to send field data for example to Catalyst (in situ visualization on tools from Paraview)
- Connection with ESIWACE WP5

QUESTIONS ?

- Previous subjects ?
- Other subjects ?

SUGGESTIONS ?