Recent developments in XIOS

Yann Meurdesoif (IPSL-CEA/LSCE)
A. caubel(LSCE), J. Dérouillat(LSCE)
O. Abramkina (Idris/MdlS), Y. Wang (MdlS)

IS-ENES3 has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824084

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
- Non negligible impact on 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 on robustness, reliability, performance and flexibility for future evolutions

Major XIOS internal core rewriting phases, begun 2 years ago :

Dev branches : XIOS_ONE_SIDED -> XIOS_SERVICE -> XIOS_COUPLING

- ~ 210 commit
- ~ 40 000 code lines added, deleted or moved

New developments goals

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 performance
- 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

Main of major developments are finished (require MPI 3.1 standard). Now :

- Stabilization and checking phase
- Dead-lock hunting, memory leak hunting
- Going at scale : high resolution and high number of processes
- Performance and memory optimization

Once recovering full XIOS trunk functionalities => merging

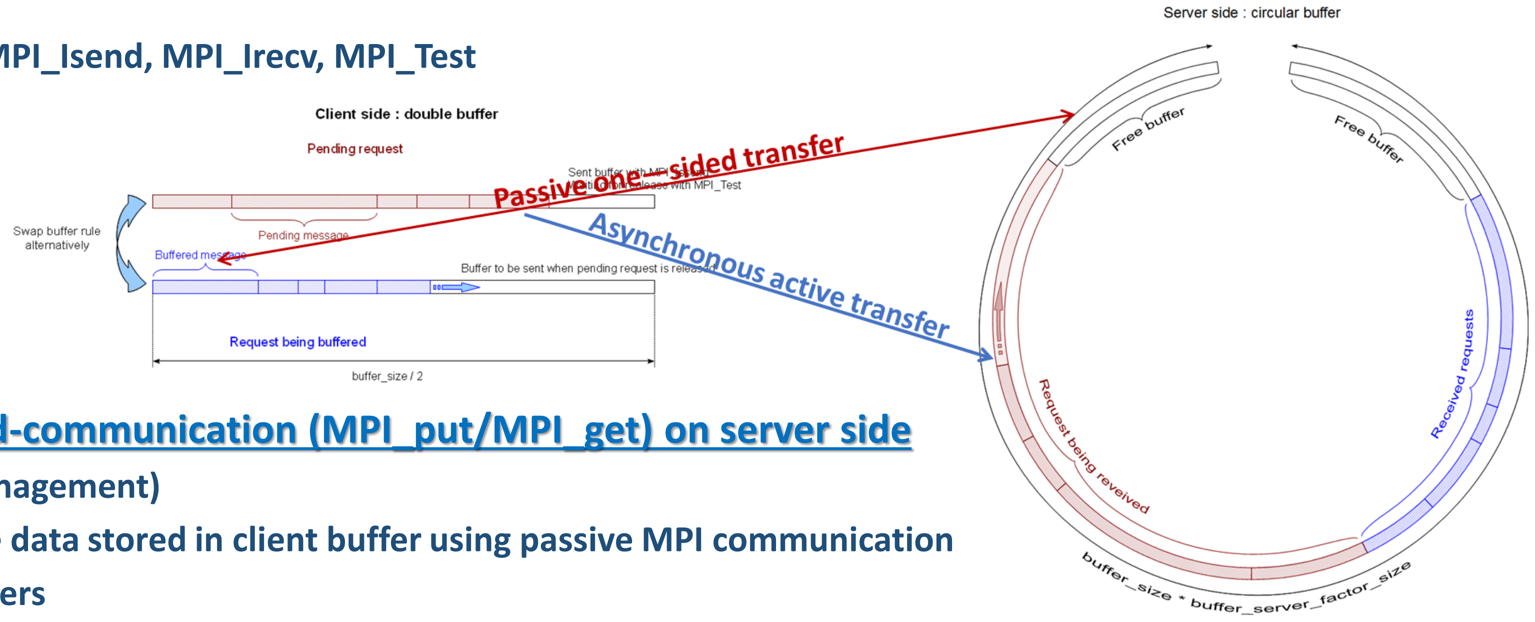
- Targeted Autumn 2021

Transfer protocol

Improvement of the transfer protocol clients <-> servers

Rise of Dead-lock in active protocol

- Previous protocol use only active transfer protocol : MPI_Isend, MPI_Irecv, MPI_Test
  - May lead to (rare) Dead-lock due to complex interactions
- Workaround may be a performance killer
  - Limiting the number of events stored in buffers


Server side : circular buffer
Client side : double buffer
Pending request
Sent buffer with MPI_Issend
Waiting for release with MPI_Test
Passive one-sided transfer
Asynchronous active transfer
Swap buffer rule alternatively
Pending message
Buffered message
Buffer to be sent when pending request is released
Request being buffered
buffer_size / 2
Free buffer
Free buffer
Received requests
Request being reveived
buffer_size * buffer_server_factor_size

We have now introduce part of passive one sided-communication (MPI_put/MPI_get) on server side

- Require MPI 3.1 MPI standard (dynamic windows management)
- In case of dead-lock, servers can directly access to the data stored in client buffer using passive MPI communication
- Remove the maximum number of events store in buffers

Fluidification of the transfer protocol at high numbers of processes

- Using new MPI 3.1 matching probe functionalities (MPI_ImProbe / MPI_Imrecv)

Dynamic resizing of transfer buffers

- Can grow dynamically
- Better reliability, memory saving

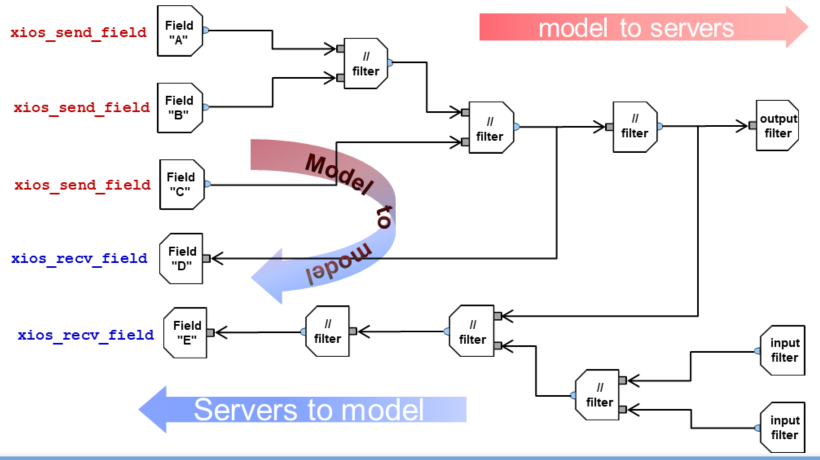
Internal rewrite of workflow engine

Introducing internally new concepts to rationalize developments

- “Views” : Object describing a local data over a global mesh
  - “Model view” : how data is stored in model memory
  - “Full View” : description of data without masking or indexing, area of output
  - “workflow view” : description of the “compressed data” running the workflow

- “Connectors” : objects which transform one view to another
  - Used to transfer data from model to workflow, make workflow transformation, transfer data between client and servers
  - Strongly inlined and optimized, intensive computation is concentrate into connectors
  - More easy for future OpenMP implementation or GPU porting

- “Filters” : objects that are chained together to achieve XIOS workflow
  - A filter embed one or more connectors
  - All computation part after the initialization phase has been totally expressed in term of chained filters
  - Port to OpenMP and/or GPU can be done incrementally, filters after filters



Rethinking and full rewrite of the transformation engine

- Improve robustness, removing pathological cases
- Performance improvement using transformation connectors
- More easy way to implement complex filters (i.e.: zonal mean)

Reducing the memory footprint

Reducing the memory footprint

- Large amount of memory is used for arrays of index
- Indexation is used for all data transformation/transfer which are now imbedded into views and connectors
  - From model to workflow
  - For computing workflow transformation
  - From client to server
  - For file writing or reading
- In past XIOS versions, array of index was commensurable to the size of the grid
- Reason was grid masking (3D masking for example), inducing relationship between domains and axes composing the grid

- Now we use the tensor product properties for connectors computation
  - No use anymore index arrays of grid size
  - 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

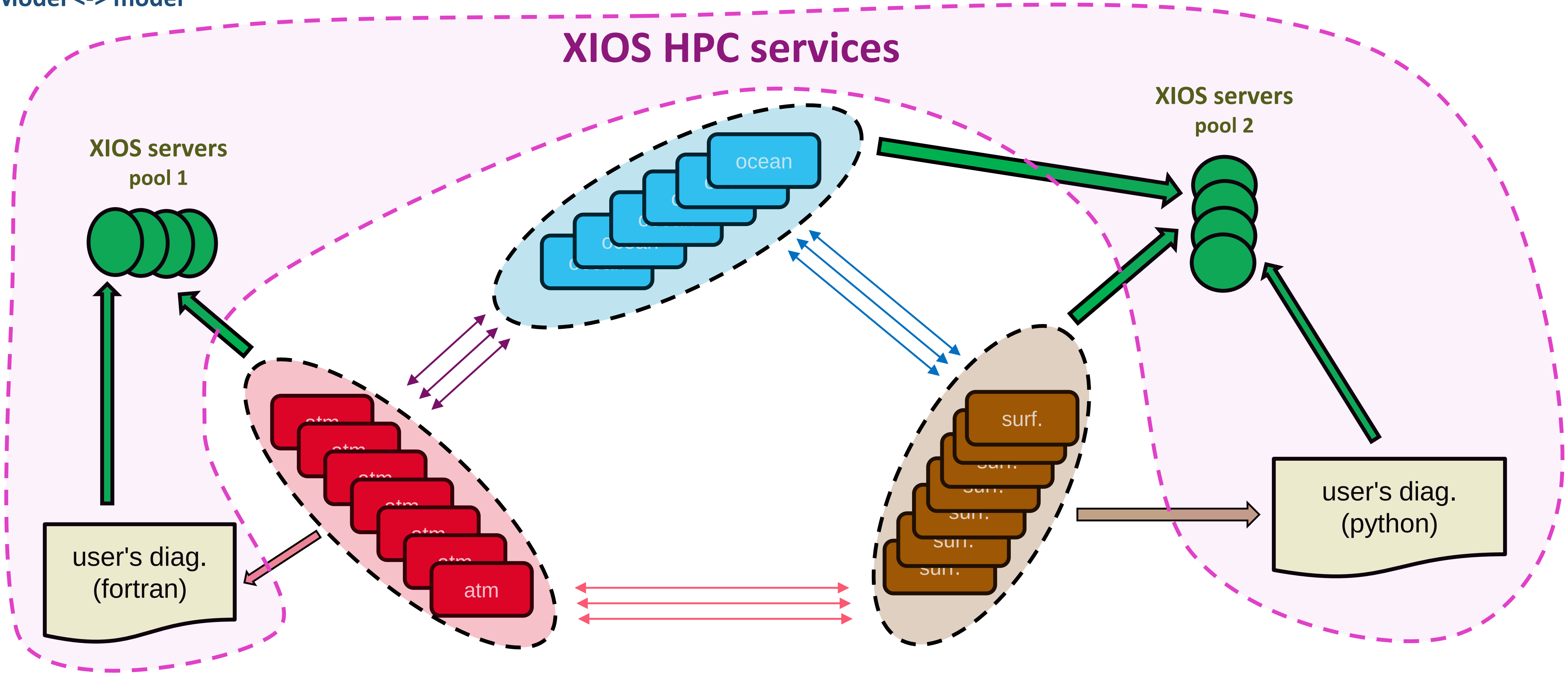
- Less memory access => higher computational performance

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

Unified way to exchange data flux between :

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



TOWARD A NEW XIOS INFRASTRUCTURE OF HPC SERVICES

What is an XIOS service

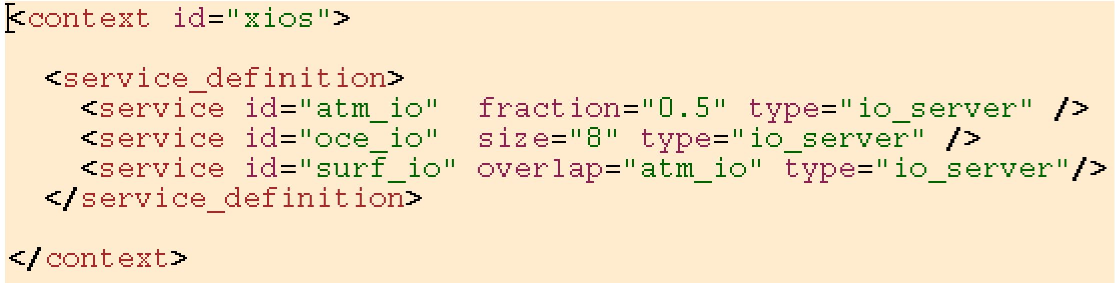
- A parallel and asynchronous program that runs over a fraction of dedicated pool of processes allocated in a simulation
  - XIOS schedules dynamically the launch of required services in free resources
  - Interconnection between models and services are achieved through XIOS middleware
  - One service can overlap an other one service if their size (processes) are the same
  - Contexts are launched and scheduled dynamically into services
  - One service can scheduled one more contexts
  - A model is saw by XIOS like a kind of service which manage itself it own resource

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
- In the future : services written by users
  - In fortran using standard XIOS interface
  - 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.

Managing services

- **Launching services**
  - Defined in XIOS the context
  - Launched at initialization onto
    - a fix set of resource
    - a fraction of the pool resource
    - overlapping an other service

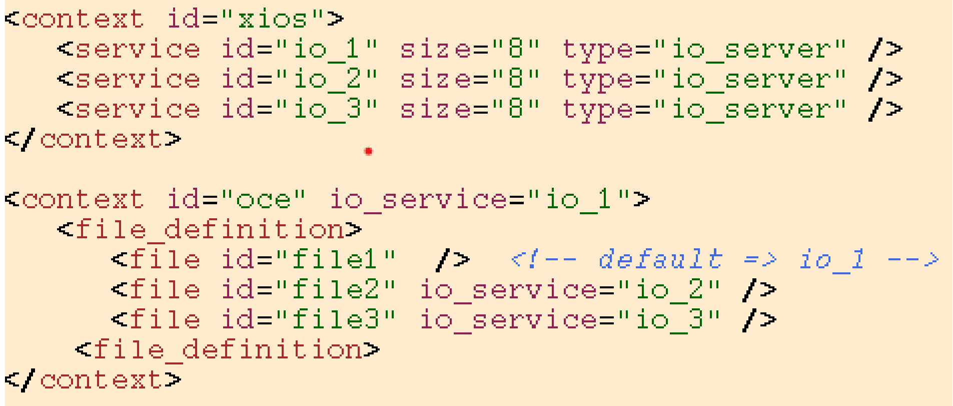


```
<context id="xios">

  <service_definition>
    <service id="atm_io"  fraction="0.5" type="io_server" />
    <service id="oce_io"  size="8" type="io_server" />
    <service id="surf_io" overlap="atm_io" type="io_server"/>
  </service_definition>

</context>
```

- **All current XIOS functionalities have been rewrote in terms of services**
  - By default, we retrieve the trunk behavior (server level 1 and 2)
  - Internally, “hidden” services are launched

- **Current XIOS services available :**
  - Server level 1 : “gathering” services
  - Server level 2 : “out_writer” services
  - I/O server stand alone : “io_server” services
  - Offloading : “offload” services

- **Each model can manage it own IO services**
  - Assigned for a whole context
  - Or even specifically by files



```
<context id="xios">
   <service id="io_1" size="8" type="io_server" />
   <service id="io_2" size="8" type="io_server" />
   <service id="io_3" size="8" type="io_server" />
</context>

<context id="oce" io_service="io_1">
   <file_definition>
      <file id="file1"  />  <!-- default => io_1 -->
      <file id="file2" io_service="io_2" />
      <file id="file3" io_service="io_3" />
    <file_definition>
</context>
```

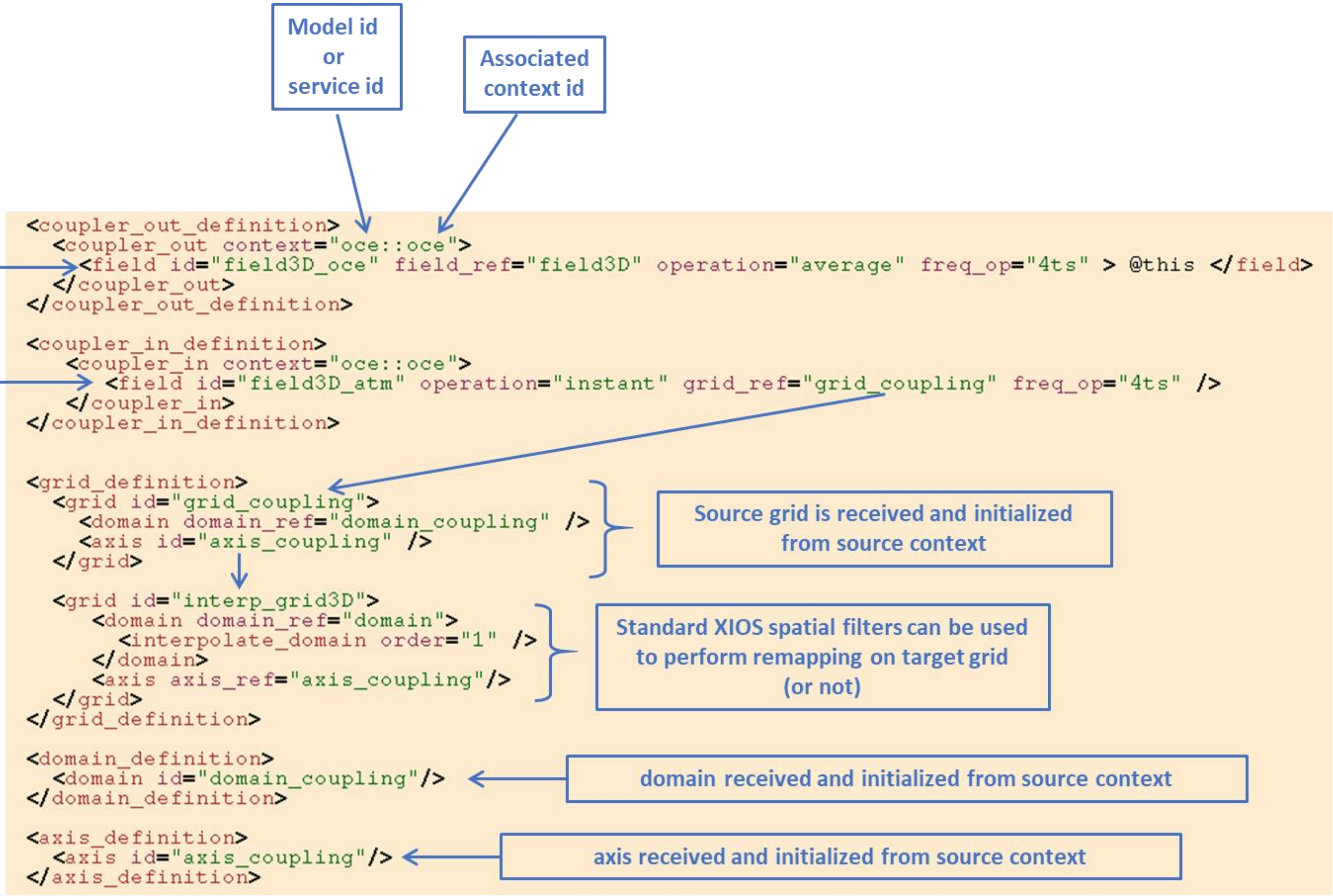
Flux exchanges : model coupling

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)
- **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

Sent field

Received field



Model id or service id

Associated context id

```
<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>
```

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

Ex : 2 way coupling between models

- **Grid exchange between models is performed by XIOS middleware**
- **Field interpolation from source to destination grid is performed by XIOS workflow**

Offloading

Costly diagnostics written in XML could be offloaded

- Similar way than for model coupling : ex : diagnostic offload that received field from model and resent to IO server

Launch offload service

```
<context id="xios">
  <service_definition>
    <service id="offload" size="8" type="offload"/>
  </service_definition>
</context>
```

Model context

```
<context id="oce" >

  <coupler_out_definition>
    <coupler_out context="offload::oce_diag" />
      <field id="field_diag" grid_ref="grid" />
    </coupler_out>
  <coupler_out_definition>

</context>
```

Grid sent to the offload context diag at initialisation

Data flux sent to diag at each time step

Offloaded diagnostic context

```
<context id="oce_diag" offload_service="offload" />

  <coupler_in_definition>
    <coupler_in context="oce::oce">
      <field id="field_diag" grid_ref="grid">
    <coupler_in>
  <coupler_in_definition>

  <file_definition>
    <file id="file_out" output_freq="1mo" />
      <field field_ref="field_diag" operation="average"/>
    </file>
  </file_definition>

</context>
```

Grid received from model at initialisation

Data flux received at each time step

Data flux is averaged and resent to IO servers

Future services proposal : ensemble

Ensemble management

- **Recent initiative :**
  - From NCAS/UREAD : high resolution ensemble (up to 100 members) for atmosphere with data reduction (min, max, average)
  - From IPSL : low resolution ensemble (80 members), but for a full coupled ESM
- **Works, but suffer of a lot of constraints**
  - All members must run simultaneously in same global MPI communicator
  - Impact XIOS and models performance efficiencies due to implicit synchronizations between members
  - If one member falls => everybody fall, difficult to get an efficient fault tolerance management
    - ➧ Initialy, the IO servers have not been designed for this…

- **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 a fallen member
  - Fallen members can be rerun independently later

Future services proposal : AI

Is new AI service could be useful ?

- **XIOS is a "windows" on the models**
  - ○ Full description of exported/imported data flux and 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-enes
INFRASTRUCTURE FOR THE EUROPEAN NETWORK
FOR EARTH SYSTEM MODELLING

The Consortium

Coordinated by **CNRS-IPSL**, the IS-ENES3 project gathers **22 partners** in **11 countries**

cnrs with cea SORBONNE UNIVERSITÉ
Met Office
National Centre for Atmospheric Science
NATURAL ENVIRONMENT RESEARCH COUNCIL
DKRZ DEUTSCHES KLIMARECHENZENTRUM
BSC
CERFACS
CENTRE EUROPÉEN DE RECHERCHE ET DE FORMATION AVANCÉE EN CALCUL SCIENTIFIQUE
cmcc Centro Euro-Mediterraneo sui Cambiamenti Climatici
Koninklijk Nederlands Meteorologisch Instituut
Ministerie van Infrastructuur en Waterstaat
SMHI
UK Research and Innovation
UC UNIVERSIDAD DE CANTABRIA
DLR
netherlands eScience
Norwegian Meteorological Institute
METEO FRANCE Toujours un temps d'avance
MANCHESTER 1824 The University of Manchester
ΔΗΜΟΚΡΙΤΟΣ
WAGENINGEN UNIVERSITY & RESEARCH
CHARLES UNIVERSITY
NORCE
LiU LINKÖPING UNIVERSITY

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N°824084

Our website
https://is.enes.org/

Follow us on Twitter !
@ISENES_RI

Contact us at
is-enes@ipsl.fr

YouTube
Follow our channel
IS-ENES3 H2020