

The PRISM Coupling and I/O System: OASIS4

Sophie Valcke¹, Damien Declat¹, René Redler², Hubert Ritzdorf², and Reiner Vogelsang³

¹ CERFACS, Toulouse, France

² NEC-CCRLE Sankt Augustin, Germany

³ SGI Munich, Germany

Abstract. The aim of the PRISM coupling and I/O system, developed in the framework of the EU funded PRISM project, is to provide a portable, efficient and easy-to-use open source software package, which includes a concise application programmer interface (API) to manage the coupling of arbitrary climate component models as well as the I/O of each individual component. In this article we will focus on the way the PRISM coupler drives the whole coupled model, ensuring the synchronization of the different component models, the exchange of the coupling fields directly between the components or via additional transformation processes, and I/O actions from/to files.

1 General view of the system

A new coupler for Earth System Models (ESMs) is currently under development in the framework of the Program for Integrated Earth System Modelling (PRISM). PRISM is an infrastructure project funded by the European Commission, with 22 European institutions actively participating. PRISM started in December 2001 for a total duration of 3 years. Its main objective is to provide a software infrastructure facilitating the assembly, execution and post-processing of Earth System model simulations, based on existing state-of-the-art European Earth System component models. To achieve this goal, PRISM aims at defining and promoting technical and scientific standards for Earth System modelling [1].

We present here the PRISM coupler as a central part in the Earth System modelling environment, which drives the whole coupled model, ensuring the synchronization of the different component models, the exchange of the coupling data directly between the components or via additional transformation processes, and the exchange of I/O data between the components and disk files. As I/O and coupling data share many characteristics, it was in fact decided to develop one common model library for both purposes.

The different constituents of the PRISM Coupling and I/O System are therefore the Driver, the PRISM System Model Interface Library (PSMILe) and the Transformer. The PSMILe includes the Data Exchange Library, which performs the exchanges of coupling data, and the I/O library which reads/writes the I/O data from/to files. The internal PSMILe communication follows the MPI [2] standard; the supported file format is NetCDF [3]. To interact with the rest of the PRISM System at run-time, the component models to be coupled have to include specific PSMILe instructions.

Each PRISM application, i.e. each code, is provided with an Application Description (AD) XML file which describes its general characteristics. For each component model included in each application, a Potential Model Input and Output Description (PMIOD) XML file describes the relations the component is able to establish with the rest of the coupled model through data inputs and outputs. The AD and PMIODs are written by the model developer.

The data described in the PMIOD are the *transient* variables, i.e. data evolving during the run, received or provided at run-time by the model at an a priori unknown frequency, from or to an external entity (another model or a disk file). The PMIOD also contains the *persistent* variables, i.e. parameters requested initially by the component to define its physical configuration. These parameters have default values but may be adapted by the user.

To configure one particular coupled simulation, the user writes through a Graphical User Interface (GUI) a Specific Coupling Configuration (SCC) XML file which contains the general characteristics and process management information of the coupled simulation, based on the information contained in the ADs of selected applications. The user also writes through the GUI for each component, based on the related PMIOD information, the Specific Model Input and Output Configuration (SMIOC) XML file which specifies the relations the component model will establish at run time with the rest of the coupled model through coupling exchanges or I/O for a specific run.

At run-time, the Driver initially reads in the SCC and SMIOCs information, and distributes the relevant information to the different component model PSMILes. During the run, the component PSMILes and the Transformer then automatically act accordingly to the user's specification contained in the SCC and the SMIOC XML files. The switch between coupling exchange and I/O action is therefore totally transparent for the component model itself.

The component models can also operate in a stand-alone mode without modifications, with or without the Driver; in that case, all external interactions correspond to I/O actions from/to disk files.

2 The Driver

The PRISM Driver extracts the information from the user configuration files and organizes the process management in the PRISM simulation.

The first task of the PRISM Driver is to get the information given by the user in the SCC XML configuration file. The information is first extracted using the libxml library, and then passed from C to Fortran to fill up the PRISM Driver structures.

Once the PRISM Driver has accessed the process management information contained in the SCC XML file, it is able to launch the different applications. The Driver then participates in the establishment of the different MPI communicators (see Section 3.1), and transfers the relevant SCC information to the different component model PSMILes (corresponding to their PRISM_Init call). When the PRISM simulation context is set, the Driver accesses the SMIOCs XML files (see Section 1) The Driver sorts this component specific information, and defines global identifiers for the components, their grids, their transient variables, etc. to ensure global consistency between the different

processes participating in the coupling. Finally, the Driver sends to each component PSMILe the relevant coupling and I/O information (e.g. source or target components or files and their global identifier) and information about the local and non-local transformations required for the different transient fields. With such information, the PRISM applications and components are able to run without any other interactions with the Driver. Thus the Driver process is then used to execute the Transformer routines (see Section 4).

When a component reaches the end of its execution, its processes send a signal to the Transformer instance via a simple MPI.Send by calling the PRISM.Terminate routine (see Section 3.5). Once the Transformer instance has received as many signals as processes active in the coupled run, the Transformer routines stop and the Driver finalizes the simulation.

3 Interface and communication

As already mentioned in the introduction the whole PSMILe communication is handled by MPI calls. To communicate with the rest of the coupled system, each component model needs to be linked to the PRISM System Model Interface Library (PSMILe). The PSMILe is the software layer that manages the coupling data flow between any two (possibly parallel) component models, directly or via additional coupling processes, and handles data I/O from/to files.

The PSMILe is layered, and while it is not designed to handle the component internal communication, it completely manages the communication to other model components and the details of the I/O file access. The detailed communication patterns among the possibly parallel participating component processes are established by the PSMILe, based on the source and target components identified for each coupling exchange by the user in the SMIOCs and on the local domain covered by each component process. This complexity is hidden from the component codes as well as the exchanges of coupling fields *per se* built on top of MPI. The interface was designed to keep modifications of the model codes at a minimum when implementing the API.

A major principle followed throughout the declaration phase and during the transmission of transient fields is that of using handles to data objects accessible in the user space once they have been declared.

The PSMILe API routines that are defined and implemented are not subject to modifications between the different versions of the PRISM coupler. However new routines may be added in the future to support new functionality. In addition to that the PSMILe is extendable to new types of coupling data and new types of grids. Some complexity of the API arises from the need to transfer not only the coupling data but also the meta-data as will be explained below.

The next sections describe the functioning of the PSMILe describing its different routines in the same logical order in which they should be called in a component model.

3.1 Initialisation phase

The following routines participate in the coupling initialisation phase:

- PRISM_Init
- PRISM_Init_Comp
- PRISM_Get_Local_Comm
- PRISM_Initialized

The initialisation of the PRISM interface and the coupling environment is performed with a call to `PRISM_Init`. `PRISM_Init` belongs to the class of so-called collective calls and therefore has to be called once initially by each process of the coupled application either directly or indirectly via `PRISM_Init_Comp` (see below). Since all communication is built on MPI routines, the initialisation of the MPI library is checked below `PRISM_Init`, and a call to `MPI_Init` will be performed if it has not been called already by the application or during a previous call to `PRISM_Init`. It is therefore not allowed to place a call to `MPI_Init` after the call to `PRISM_Init` in the application code, since this will lead to a runtime error with most MPI implementations.

Conversely, a call to `PRISM_Terminate` (see Section 3.5) will terminate the coupling. If `MPI_Init` has been called before `PRISM_Init` and `PRISM_Init_Comp`, `MPI_Finalize` must be called somewhere after `PRISM_Terminate` in order to shut down the parallel application in a well defined way. The four possibilities to place the initialisation and termination calls are summarized in Fig. 1.

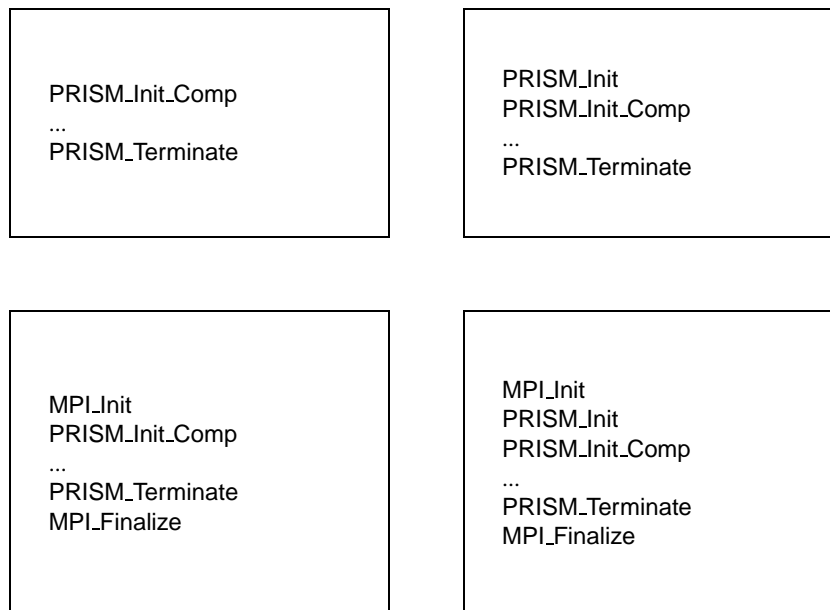


Fig. 1. The four allowed sequences of calls for initializing and terminating PRISM coupling and MPI.

Within PRISM_Init it is detected if the coupled system has been started in a static or pseudo-dynamic mode with the latter using MPI2 spawn functionality [4]. In the pseudo-dynamic case all child processes remain in PRISM_Init and participate in the launching of further child processes until the launching is completed.

PRISM_Init_Comp needs to be called once initially by each process for each component model executed, no matter if different component models are executed sequentially by the same process(es) or if each process is devoted to only one single component.

If PRISM_Init has not been called before by the same process, PRISM_Init_Comp calls PRISM_Init and returns with a warning. Although recommended it is therefore not necessary to implement a call to PRISM_Init.

In order to allow for a separation between the communication that is internal to the individual application or component after the calls to PRISM_Init_Comp and PRISM_Init, MPI communicators for the application and the component local communication are provided by the PSMILe and can be accessed via PRISM_Get_Local_Comm. This routine needs to be called only by MPI parallel code. PRISM_Get_Local_Comm is the only MPI specific call in the PSMILe API. subsectionDeclaration of the grids and related quantities

The PSMILe API has been designed in such a way that grids are defined in two steps. In the first step, the grid and its volume elements are defined. The second step covers the definition of grid points where transient variables are located within the volume elements.

Definition of the grid and its volume elements

The related API routines are:

- PRISM_Def_Grid
- PRISM_Set_Corners
- PRISM_Set_Scalefactors
- PRISM_Def_Partition

Once the component is declared, the first step in the grid definition phase is to announce a grid with its most general characteristics like the type of the grid and the shape of the local computing domain with a call to PRISM_Def_Grid.

In order to match the data structures of the various component codes (in particular for the geographical information) as closely as possible, Fortran90 overloading is used. All descriptive arrays provided by the component code through the PSMILe API (e.g. the grid point location through PRISM_Set_Points, see below) can have one, two or three dimensions and can be of type real or double precision. There is no need to copy the data arrays prior to the PSMILe API call in order to match some predefined internal PSMILe shape. To interpret the received array correctly, it is therefore required to define properly a PRISM_Def_Grid argument describing the grid type which implicitly specifies the shape of the data array that is passed to the PSMILe. Grids that are supported cover completely regular grids (longitude, latitude and the vertical), irregular grids in longitude and latitude but regular in the vertical, and sigma coordinates in the vertical for both regular and irregular horizontal grids. Completely unstructured grids

and horizontally unstructured grids having either sigma coordinates in the vertical or a regular vertical axis are also supported.

For gridded data, the volume elements which will discretize the computing domain covered by the local partition on the sphere are then defined. To achieve this, the corner points (vertices) of the volume elements of the local partition are provided for each grid with a call to `PRISM_Set_Corners`. The communication and possibly the repartitioning of gridded data between two coupled component models will be based on this geographical description of the local partition. The volume dimensions of the elements may also be provided by calling `PRISM_Set_Scalefactors`.

For horizontal grid definition, the only units supported in a first step are degrees east and degrees north (spherical coordinate system).

In a first stage, no vertical unit transformation is included in the coupler. If no vertical interpolation is required, all vertical units are allowed as long as they are the same on the source and target sides. If vertical interpolation is requested, only units in which interpolation make sense (e.g. depth or pressure) are supported and have to be the same on the source and target sides. In a later stage, the PSMILe interface should be able to transfer functions and associated variables; the coupler will then be able to perform unit transformation (e.g. transform hybrid coordinates into meters given the temperature, pressure and humidity fields).

For non-gridded data, it is not required to specify any of the geographical information. The transfer and possibly the repartitioning of non-gridded data between two component models will be based on the description of the local partition in terms of indices in the global index space defined through a call to `PRISM_Def_Partition`.

Placement of grid points in the volume elements

In these volume elements, different sets of points on which the variables will be evaluated can then be placed. The related API routines are:

- `PRISM_Set_Points`
- `PRISM_Set_Vector`
- `PRISM_Set_Subgrid`
- `PRISM_Set_Mask`
- `PRISM_Set_Vectormask`

The model developer decides where the points are located and what they represent in terms of the volume elements. This allows to describe a number of different contents on the same grid. Points can be used to represent means, extrema or other properties of the variable within its volume element. The localization of a set of points is defined through a call to `PRISM_Set_Points`.

For vectors, three sets of points can be placed with `PRISM_Set_Points`, and then grouped together with a call to `PRISM_Set_Vector`, so that different vector components can be placed at different locations (staggered grids). Following the same logics, mask arrays can be specified for points or vector components (`PRISM_Set_Mask` or `PRISM_Set_Vectormask`).

Subgrid variables can also be represented. This introduces a 4th dimension within the volume which represents the different subgrid classes. The fraction of each subgrid class for the volume elements can be defined through a call to `PRISM_Set_Subgrid`.

```

PRISM_Init_Comp ( comp_id, ... )

PRISM_Def_Grid ( grid_id, ..., comp_id, ... )

PRISM_Set_Corners ( grid_id, ... )

PRISM_Set_Points ( point_id, ..., grid_id, ... )

PRISM_Set_Subgrid ( subgrid_id, ..., grid_id, ... )

PRISM_Set_Mask ( mask_id, ... )

PRISM_Set_Scalefactors ( grid_id, ... )

PRISM_Set_Vector ( vector_id, ..., point_ids, ... )

PRISM_Set_Vectormask ( vectormask_id, mask_ids, ... )

PRISM_Def_Partition ( grid_id, ... )

PRISM_Def_Var ( var_id, ..., point_id, mask_id, ... )

```

Fig. 2. The dependency of handle Ids.

3.2 Declaration of Transient Variables

Each transient input or output variable is then declared and associated with the previously defined grids and masks, through a call to `PRISM_Def_Var`.

For the case where a set of variables located on the same set of points needs to be treated together, the bundle notion has been introduced. This means that a set of related variables can be ordered along a 4th dimension, and declared and transferred as one coupling variable.

As already mentioned, one design principle for the PSMILe API routines is the usage of handle identifiers (ids); the dependency of handle ids between the various call is highlighted in Fig. 2.

3.3 Neighborhood search and determination of communication patterns

Following `PRISM_Init`, `PRISM_Enddef` is the second collective call and has to be called once by each application process when all components within the respective application have completed the definition phase.

Let us note here that all grid characteristics (mask, areas, subgrid fraction, etc.) will be allowed to evolve during the run. Every time the grid has changed the developer is allowed to re-invoke the appropriate grid description calls (see Section 3.1) completed by a new `PRISM_Enddef` call to invoke the search and recalculation of neighbourhood relations. A comparison of data arrays is performed by the `PSMILe` to detect which part of the data information has changed, and appropriate actions will then be performed concerning the modified part only.

In order to reduce the overhead caused by the coupling, i.e. the actual exchange of coupling fields, it is required to establish communication only between those pairs of processes that actually have to exchange data based on the given coupling configuration. To achieve this the neighbourhood search is performed in this phase and the communication pattern is generated. For large grid sizes the neighbourhood search can become very costly, especially when considering adaptive grids. Therefore the neighbourhood search is performed in a parallel way within `PRISM_Enddef` in order to save communication time, to decrease the memory consumptions and to reduce time spent in the search. In order to achieve this goal, `PRISM_Enddef` works on the local application grids as much as possible.

In an initial step, bounding boxes are locally computed for each volume grid which was defined by `PRISM_Set_Corners` and which is used for coupling. These bounding boxes of all grids on all application processes are collected and distributed to all processes. After this initial step, the application processes have a global view on grids and processes which may have common coupling interfaces. Using this collected data, the processes can locally create a sequence of simplified grids which are currently coarsened by a factor of 2 (similar to a Multigrid Algorithm) and which are used for the neighbouring search. These sequence of grids covers the parts of the volume grid which may have a coupling interface. Using the collected data, the processes can also decide which grid points or corner points (this depends on the interpolation type) have to be transferred to a “source” process for the neighbourhood search. After receiving these coordinates, the “source” process performs the neighbourhood search and informs the sending process, which points were found in the volume grids, for which points the transient variables can be sent directly to the sending process since a matching grid point in the “source” process was found and for which points the Transformer has to perform the interpolation. If points have to be interpolated, the “source” process also sends the data requested by the interpolation, and which can already be computed in `PRISM_Enddef`, to the Transformer processes.

For non-gridded data, the search is already finished after the initial step since the application processes can locally decide after this step which values of transient variables have to be directly transferred within the coupling exchange.

Coupling data produced by one source component model may be only partially consumed by the target component model. The neighbourhood calculation performed

in the source component PSMILe allows the automatic extraction of useful subdomains only and therefore minimizes the amount of data transferred.

3.4 Coupling exchange and file I/O of transient variables

The PRISM_Put and PRISM_Get routines treat the exchange of transient variables.

The PSMILe exchanges are based on the principle of "end-point" data exchange. When producing data, no assumptions are made in the source component code concerning which other component will consume these data or whether they have to be written to a file, and at which frequency. Likewise, when asking for data a target component does not know which other component model produces it or whether they are read in from a file. Furthermore source data can be directed to more than one target (other component models and/or disk files).

One argument of the sending and receiving routines, respectively PRISM_Put and PRISM_Get, is the data array to be sent or received. Each process of a parallel component model sends or receives only its local partition of the data, corresponding to its local grid defined previously (see Section 3.1).

The PRISM_Put and PRISM_Get routines can be placed anywhere in the source and target code and possibly at different locations for the different coupling fields. The destination or the source (another component model or a file) for each field was defined by the user in the SMIOC (see Section 1) and the coupling exchanges and/or the I/O actions take place according to the user external specifications. The switch between the coupled mode and the forced mode is therefore totally transparent for the component model.

Other arguments of the PRISM_Put and PRISM_Get routines are the actual date at which the call is performed and the date bounds for which it is valid. These routines can be called by the model at each timestep. The sending/receiving is actually performed only if the date and date bounds corresponds to a time at which it should be activated, given the field coupling or I/O period indicated by the user in the SMIOC; a change in the period is therefore also totally transparent for the component model itself. The PSMILe can also take into account a timelag defined by the user in the SMIOC.

Local time transformations can be performed in the source component PSMILe below the PRISM_Put like time accumulation or time averaging, and/or in the target component PSMILe below the PRISM_Get like time interpolation. Other local operations specified by the user in the SMIOC may also be performed automatically in the source or target PSMILe, such as general algebraic operations, combination with other coupling or external data, reduction in any dimension -minimum, maximum, average-, statistics, scattering, gathering, etc.

Coupling exchanges

The coupling exchange occurs directly between two component models or via additional Transformer processes if needed (see Section 4). When the component models are parallel and have different data partitioning, repartitioning is automatically performed either directly or via the Transformer. For gridded data, the repartitioning is based on the geographical description of the local partitions; for non-gridded data, it is based on the description of the local partition in terms of indices in the global index space (see Section 3.3).

I/O exchanges

Instead of exchanging data between component models, the user can specify in the SMIOC that the source or target is a file. Simply speaking, the MPI send/receive operations underneath of PRISM.Put and PRISM.Get are “replaced” by write and read operations respectively. On a read action, a matching of the PRISM.Get date and date bound arguments with the time stamps of the time axis contained in a file is performed.

The PSMILe file I/O can operate in two general I/O modes:

- *Distributed I/O*

Each process of component model works on a individual file containing the transient field information local to the domain on which that process works. Domain partitioning information are written into the resulting files such they can be merged into one file during a post processing step.

- *Parallel I/O*

The data of a transient field are read from or written to one file. The domain partitioning information is exploited such that the data are collected - stitched together - during write operations or distributed to the parallel processes of a component model. Domain stitching and distribution of data happen transparently for the parallel component model.

The supported file format is NetCDF according to the CF convention which ensures:

- Portability of data between different computer hardware platforms.
- A self-explaining description of data according to the needs of climatologists.
- The seamless usage of post-processing and visualization tools developed by other workpackages of the PRISM project.

For the NetCDF file operations we exploit the MPP package [5] developed by the FMS project (Flexible Model System) ¹ at GFDL Princeton. The MPP package is proven technology and is used by the ocean model MOM4, for instance. Moreover, the MPP package supports the two general I/O modes - distributed and parallel - as described above.

The MPP package is driven by a PSMILe internal layer which interfaces with various sources of information. For instance, the definition of grids and masks as well as the form of the data (bundle or vector) of a transient field is provided through the PSMILe

¹ <http://www.gfdl.gov/~fms>

API. On the other hand the information with regard to the CF standard name and unit are provided by the SMIOC through the Driver which reads that source of information.

The internal interface layer copes also with the fact that the NetCDF file format has certain restrictions with respect to size of a file. Therefore, on output chunking of a series of time stamps across multiple files will be provided depending on a threshold value of the file size.

Last but not least, the internal interface to the read and write primitive routines of the MPP package handles the proper mapping of rank and type of arrays passed.

As a PSMILe further enhancement, we are now going to evaluate a very recent development of a parallel NetCDF library called Par-Netcdf [6].

3.5 Termination

The following routines participate in the termination of the coupled run:

- PRISM_Terminate
- PRISM_Terminated
- PRISM_Abort

In analogy to the initialisation phase, a call to PRISM_Terminate, which again is a collective call, will make the calling process to wait for other processes participating in the coupling to reach the PRISM_Terminate as well. At this point, the following actions are performed:

- All open units under control of the PSMILe are closed.
- The output to standard out is flushed.
- The Driver is notified about the termination of the respective process.
- All memory under control of PSMILe is deallocated.

After calling PRISM_Terminate no coupling exchanges are possible anymore for this process and no further I/O actions under control of the PSMILe can be performed; however, it is still possible for the application to perform local operations and to write additional output which shall not be under control of the PSMILe. Furthermore, if MPI_Init has been called in the code before the call to PRISM_Init, component internal MPI communication is still possible after the call to PRISM_Terminate, until the MPI_Finalize is called by the component (see also Section 3.1).

PRISM_Terminated can be used to check whether PRISM_Terminate has already been called by this process. This may help to detect ambiguous implementations of multi-component applications.

It is common practice in non parallel Fortran codes to terminate the program by calling a Fortran STOP in case a runtime error is detected. In MPI-parallelized codes it is strongly recommended to call MPI_Abort instead to ensure that all parallel processes are stopped and thus to avoid non-defined termination of the parallel program. For coupled application, the PSMILe provides a PRISM_Abort call which guarantees a clean and well-defined shut down of the coupled model.

4 The Transformer

The PRISM Transformer manages non-local transformations of the coupling field. In the PRISM context, a transformation is considered as non-local when the grid information needed to perform this transformation comes from different components. In that case, the data are transferred to the Transformer where the transformation is performed. The main non-local transformation is the regridding (often called the interpolation), i.e. the expression on the grid of a target component of a source field given on its grid. The Transformer performs only the weights calculation and the regridding *per se*. As explained in section 3.3, the neighborhood search, i.e. the determination for each target point of the source points that will contribute to the calculation of its regridded value, is performed in parallel in the source PSMILe. Other transformations, such as global conservation or combination of different fields coming from different components with different grids, can also be performed by the Transformer for efficiency reasons.

The PRISM Transformer is part of the PRISM Driver process. Analysing the SMIOC information (data exchanges, grid descriptions, required transformations), the PRISM Driver is able to determine if the Transformer process is required or not. If it is the case, the PRISM Transformer is launched via simple routine calls on the Driver process.

The PRISM Transformer can be assimilated to an automate that reacts following predefined sequences of actions considering what is demanded. The implementation of the Transformer is based on a loop over the receptions of predefined arrays of ten Integers sent by the component PSMILe. These ten integers give a clear description of what has to be done by the Transformer. Following pre-defined sequences, the PRISM Transformer is thus able to react to the corresponding sequence activated on the sender side.

During the initialisation phase, all information given through the SMIOC files is set up in the Driver/Transformer structures. This concerns e.g. the description of the user chosen transformations but not the data like the latitude, longitude, mask, or areas arrays of the different grids involved in the regridding processes, which are in fact received from the different components through the PSMILe API in the declaration phase (see section 3.1).

Then the Transformer receives from the different component PSMILes the grid information resulting of the different neighbouring searches. As described in Section 3.3, the information transferred at this stage (and the corresponding data arrays sent later during the progress of the simulation) cover only to the data involved in the regridding process. The Transformer then calculates the weight corresponding to each source neighbour depending on the regridding method chosen by the user. The end of this phase corresponds in the components to the PSMILe routine `PRISM.Enddef`.

During the simulation timestepping, the Transformer receives orders from the components processes to receive data for transformation (source component process) or to send transformed data (target component process). After a reception, the PRISM Transformer applies the appropriate transformations or regridding following the information collected during the initialisation phase (here, the regridding corresponds to applying the pre-calculated weights to the source field). In case of request of fields, the PRISM Transformer is able to control if the requested field has already been received and trans-

formed. If yes, the data field is sent; if not the data field will be sent as soon as it will have been received and treated.

The PRISM Transformer is informed by the participating processes once they are ready to finish the coupled simulation; the PRISM Transformer gives the hand back to the PRISM Driver. sectionConclusion

In this paper, the general design of a new parallel coupling and I/O system for Earth System Models currently under development in the framework of PRISM project was presented. Its different parts, i.e. the Driver, the PSMILe and the Transformer were described in more details.

With this new coupling and I/O system, PRISM ensures efficient and parallel 3D coupling for full climate models built on complex component models representing the different parts of the Earth System (atmosphere and ocean general circulation models, but also sea ice, land surface, atmospheric chemistry and ocean biogeochemistry models) running on the present and next generation of advanced high performance computing facilities. Instead of being coupled to other parts of the system each component can use the same interface in a stand-alone mode to receive data from files.

A first prototype of this coupler is delivered at end of December 2003. The final coupler gathering all the functionality described here above will be available at the end of PRISM project, i.e. in December 2004.

References

1. Guilyardi, E., Budich, R., Komen, G., Brasseur, G.: PRISM System Specification Handbook, Version 1. PRISM report series No.1, 230 pp. ISBN: 90-369-2217-8. (2003) <http://prism.enes.org/Results/Documents/Handbook/Handbookv1.0.1.pdf>.
2. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI – The Complete Reference. Second edn. Volume 1, The MPI Core. MIT Press (1998)
3. Rew, R.K., Davis, G.P.: Unidata's netCDF Interface for Data Access: Status and Plans. In: Thirteenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, Anaheim, California, American Meteorology Society (1997)
4. Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., Snir, M.: MPI – The Complete Reference. Volume 2, The MPI Extensions. MIT Press (1998)
5. Balaji, V.: Parallel Numerical Kernels for Climate Models. In: ECMWF TeraComputing Workshop 2001, ECMWF, World Scientific Press (2001)
6. Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., Zingale, M.: Parallel netCDF: A Scientific High-Performance I/O Interface. In: SC2003 Proceedings, Supercomputing (2003)