A Graphical User Interface for configuring YAC

Version 2.0.0

Rene Redler[1], Moritz Hanke[2], and Maxim Yastremsky[1]

[1] *Max-Planck-Institut für Meteorologie, Hamburg*
[2] *Deutsches Klimarechenzentrum, Hamburg*

Aug 2020

We introduce the usage of the graphical user interface to generate a coupling configuration Extensible Markup Language (XML) file for YAC. We explain the necessary steps to define the coupling between a pair of model components and describe the options we offer to configure a particular interpolation method. In the appendix we provide an example of the XML files which are required for input and explain some elements of the resulting XML file generated by the GUI.

1. Starting the GUI

The java source code for the graphical user interface (GUI) is shipped together with the source code of the coupler library. A precompiled Java archive (jar) file is available. To get up and running with the precompiled archive file, follow these steps:

1. Make sure you have a OpenJDK[1] installed. The following open-source builds of the Java Development Kit have been tested:
   - openjdk 11.0.1[2]
   - openjdk 12[3]

[1] https://openjdk.java.net/
[2] http://jdk.java.net/11/
[3] http://jdk.java.net/12/

2. launch the YAC GUI via command line

```
java -jar CouplingGui.jar
```

The jar file can be compiled by using ant:

1. Make sure you have Apache Ant [4] installed.
2. Once, ant is available the whole java project is built within the GUI directory using

```
ant build
```

and the java library is generated by typing

```
ant create_run_jar
```

Note that the Runtime Environment is sufficient for executing the (shipped) jar file. To lauch the GUI invoke

```
java -jar CouplingGui.jar
```

In its current version 1.5 the GUI requires a Extensible Markup Language (XML) description of model components as input to allow for the configuration of the coupling. An example of a component XML file is provided in Appendix A. Compared to the coupling XML file (an example is provided in Appendix B) a component XML file has a very simple structure and can easily (probably much faster) be generated with a simple text editor.

The start window appears as shown in Fig. 1. To create a new coupling from scratch the user needs to load two components represented by XML files by clicking on `Component 1` and `Component 2` buttons below the file status line. After clicking on any of these buttons the standard Open File Dialogue will appear. When the user has selected the component XML file and clicked on the `Open` button, the content of the component file will be loaded and displayed in the component panel. Typically, this will consist of a list of transients (see Fig. 2). Likewise, the second component has to be loaded. For each transient the GUI displays the name of the field, the name of the numerical grid on which the field is defined and its collection size. As YAC currently supports a 2-dimensional interpolation in the horizontal (on the sphere) only, the collection size can either be the number of vertical levels of this field or the number of horizontal fields hidden behind this particular field name, sometimes also referred to as bundles.

[4] http://ant.apache.org/manual/index.html

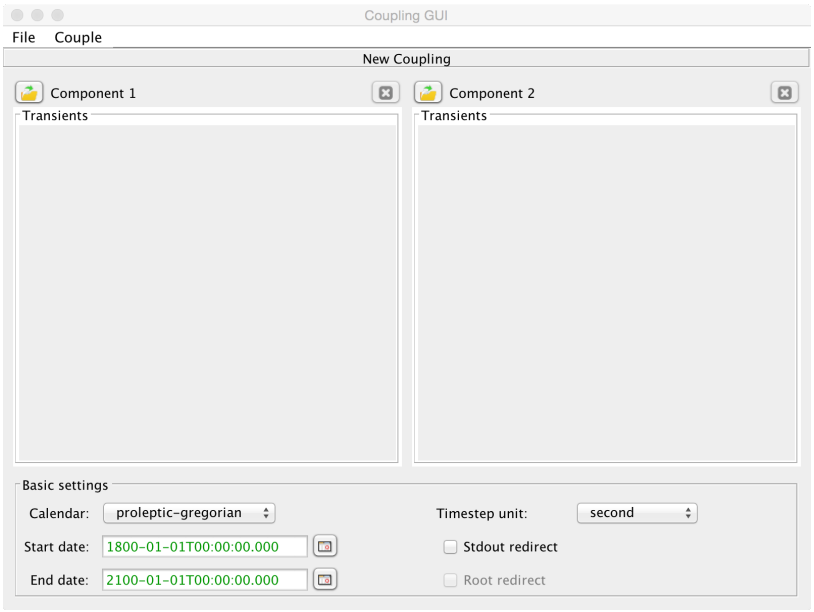

Figure 1: YAC XML configuration GUI start window.

2. New coupling

To start creating a coupling between any two physical fields (transients) of two different components the user needs to click the check box representing the source field. In this case all transients which cannot be coupled with this source transient will remain inactive. The transient is considered as valid for coupling only if it has the same name and collection size. After finding the second possible transient for coupling and the checking of its corresponding check box, a red arrow connecting these two transients will be drawn. The direction of this arrow is pointing into the direction of the coupling, from the source to the target. Both directions of coupling are possible: from the left component to the right one and vice versa. Once an interpolation instruction is defined the colour of the respective arrows are turned into green, like shown in Fig. 2 for the heat flux.

Note that “collection size” has the same meaning as in the Fortran/c user interface where it is used to describe the number of vertical levels or the number of horizontal fields that are stored with this one transient. The same horizontal interpolation stack is applied to all members in the collection.

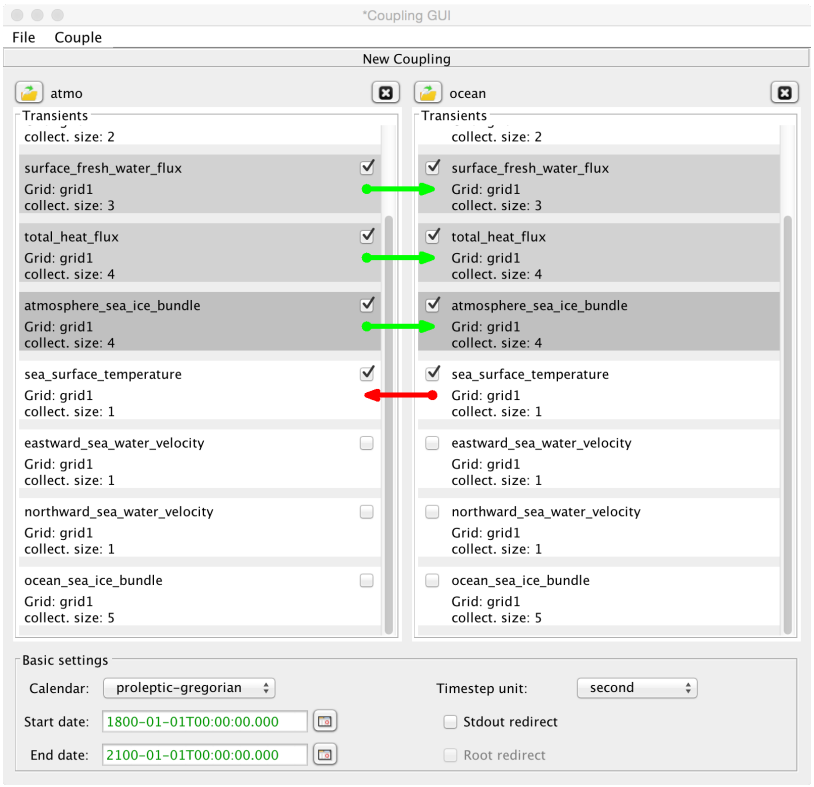

Figure 2: Example of a coupling configuration.

3. Basic settings

Time settings of coupling can be specified in the Basic settings panel of the main window. The parameter `Calender` has 3 options: `Proleptic-Gregorian`, `360d` and `365d` which activates either the Proleptic-Gregorian calendar[5], a calendar with an equal length of all months (30 days), or a calendar without any leap years. Specific dates of coupling can be set manually in the fields `Start date` and `End date`. After any user input or other interactions with values of these fields the user will be notified by the colour of the strings whether the provided date is correct or not according to the specified calendar. A green colour of values means that the date is correct for the selected calendar while red means it is not. For the Proleptic-Gregorian calendar also a quick date picker is available as an alternative. It helps to select any date with a few clicks. This widget will disappear after selecting the day of the chosen year and month or with the starting of any user interactions in the date edit field. Note that YAC offers the option to internally overwrite the calendar settings at runtime in the coupled application by using the Fortran or C interface routines. However, this will not change the contents of the XML file.

[5] http://en.wikipedia.org/wiki/Proleptic_Gregorian_calendar

The `Timestep` unit parameter indicates which time units are to be used in the timestep parameter tab described in Sec. 4.1.2. The options are `second`, `minute`, `hour`, `day`, `month`, `year`, `millisecond`, `ISO_format`, `string`. If the timestep parameter shall be provided e.g. as a shell variable, for example `${dt_atm}`, the user has to select `string` as unit. In this case the final coupling XML file needs to be edited manually, and `string` needs to be replaced by the time unit (or `ISO_format`) in which the value is provided that is going to replace the shell variable.

Furthermore, the user can request the redirection of stdout and stderr by checking `Stdout redirect`. When this is activated, output will be redirected after the application has called yac_finit or yac_cinit. The output file name gets the name of the respective component followed by the local MPI process ID. When `Stdout redirect` is activated the user has the additional choice to let only the component root processes redirect their output rather than letting each process write into its own file.

4. Subpanels

4.1. Setting parameters of specific transients coupling

In case the user has coupled two transients, additional parameters have to be provided for the coupling. To open the dialogue for setting up coupling parameters (Fig. 3) the user has to click on any of the coupled transients in the component panel area. The coupling parameters dialogue will appear after that. This dialogue has 2 tabs: Interpolation and Timestep.

4.1.1. Interpolation parameters tab

The user has the option to store the weights of the interpolation stack for coupling a transient into a file. Writing out is enabled by checking `Enforce write weight file` and specifying the filename with the extension .nc (Network Common Data Form[6]).

During the exchange the interpolation stencils are filled on the source and interpolated values are sent to the target processes. When deselecting `Mapping on source` the raw source data are sent to the target where the interpolated values are calculated. With the second option the halo exchange on the source grid is avoided with the caveat that possibly more messages are sent to the target. The put will be faster but the get will perform the implicit synchronisation instead on the target side.

[6] https://en.wikipedia.org/wiki/NetCDF

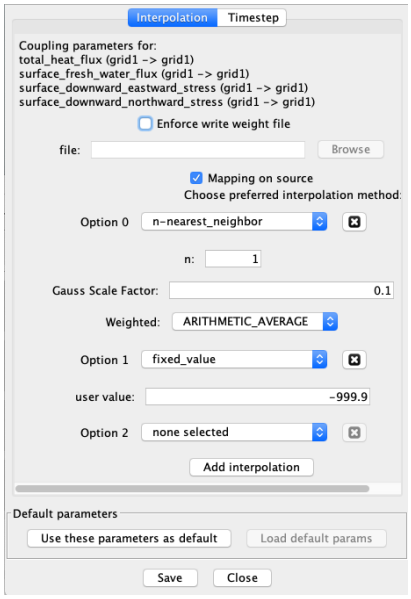

Figure 3: Coupling interpolation parameters window.

Interpolation parameters are used to specify the sequence of interpolation methods which will be used for the interpolation. The specification of interpolation methods is organised in a list. The default number of interpolation methods which can be specified is three, but additional method options will appear after clicking on the `Add more interpolations` button. The user has to define at least one interpolation method in order to activate the coupling. Otherwise, no interpolation weights will be calculated and the particular exchange of transients will remain inactive.

Most interpolation methods require more specific parameters. These will be dynamically added next to the particular interpolation method and described in some more detail in Sec. 4.2.

The complete selection can be saved as default setting for defining further transient couples with `Use these parameters as default`. This will then activate `Forget default params`. For a next transient couple definition `Load default params` will apply the currently stored default. New defaults can be set by first pressing `Forget default params`, setting new parameters and again `Use these parameters as default`.

4.1.2. Timestep parameter tab

On the timestep tab (Fig.4) the user can indicate time parameters of the coupling. In particular we require the user to specify the model time step or in

other words, the period with which the respective exchange routines are called by the application. Next we require the user to specify the coupling period for the transient. The units of these parameters are controlled in the main window of the application in the Basic settings panel described earlier in Sec. 3. When the unit is set to time unit like e.g. `minute`, the user has to enter non-negative integers. If the unit is `ISO_format`, the time step is represented by the format defined by `ISO 8601` for time durations[7]. When the unit is `string`, the user is supposed to enter the name of the parameter (or shell variable) that is later used to store the value of the time step, for example `${dt_oce}`.

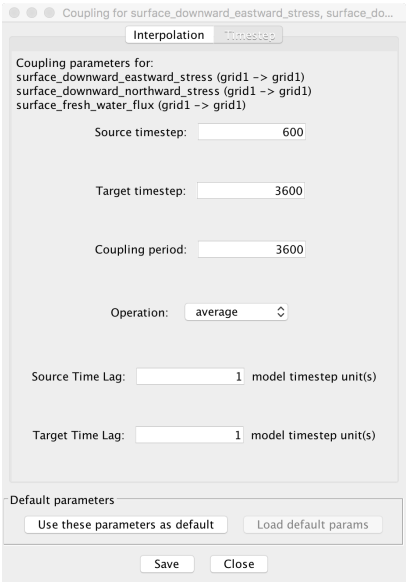

Figure 4: Coupling timestep parameters window.

In addition, the `Source Time Lag` and `Target Time Lag` can be specified. The lag is a positive integer number or zero. It is used in the library to adjust the internal event trigger clock for the source (put) and target (get) operations according to the timestepping algorithm used in the application. A source time lag of 2 will put forward the internal clock for the put event by 2 times the source time step. Like for the time step, the lag can be provided as a shell variable, for example `${atm_lag}`.

Note, that the shell variables need to be converted into valid input before it is read by the YAC library.

To ensure that no negative integers are provided, the symbol `-` is not accepted in the above mentioned two input fields.

Like with the interpolation parameters tab, default settings can be stored and applied.

[7] https://en.wikipedia.org/wiki/ISO_8601#Durations

4.2. Interpolation specific parameters

Masking is not handled and cannot be triggered via the coupling xml configuration file. Masks can only be set via the Fortran and c interface. If masks are set they are considered by all interpolation methods!

4.2.1. Average interpolation

The values at the vertices or other locations of an element can be combined to a simple arithmetic average or can contribute to the source points weighted by their distances. In the latter case distances between sources and target location are calculated on the sphere. These distances are then normalised to 1. Using the vertices of a quadrilateral element the distance-weighted average is equivalent to a bi-linear interpolation.

Partial coverage (Boolean)

By setting this to `true`, partially covered target cells will receive an interpolated value.

Weighted (Enumeration: Distance weighted or Arithmetic average, default Arithmetic average)

Choice for weighting

4.2.2. N-nearest Neighbour interpolation

The interpolation finds a number n of nearest neighbours to the target location. The n nearest neighbours can be combined to a simple arithmetic average or can contribute weighted by their distances to the source points. For the latter case the "DISTANCE_WEIGHTED" operation uses normalised inverse distances between sources and target location are on the sphere. "GAUSS_WEIGHTED" uses a Gauss kernel

$$\exp(-\frac{d^2}{sL^2})$$

where d is the between source and target points, L is a length scale derived from the source points of the individual stencil (average distance over all pairs of source points of the local stencil), and s is a scaling factor provided by the user. The default of $s = 0.1$ showed the lowest relative error for our test cases.

N (Integer value)

Number of source point values requested for the nearest-neighbour interpolation.

Weighted (Enumeration: "DISTANCE_WEIGHTED", "GAUSS_WEIGHTED", or default "ARITHMETIC_AVERAGE")

Choice for weighting

gauss_scale (Double value)

Scaling factor of the Gauss kernel, default 0.1

4.2.3. Radial Basis Functions

The interpolation finds a number n of nearest neighbours around the target location and used radial basis functions to obtain a value at the target location.

N (Integer value)

Number of source point values around a target location requested for the construction of the radial basis functions.

RBF Kernel (Enumeration: Gauss Kernel, default Gauss Kernel)

For further information see

- Joshua Reinheimer, 2018: Vector Field Interpolation using Radial Basis Functions, Master Thesis, Department of Mathematics University of Hamburg, Germany, October 26, 2018, 68 pages.

4.2.4. Conservative interpolation

The classical first-order conservative remapping identifies for each target cell all overlapping source cells. Interpolation weights are calculated based on the fractional overlap. Special care is taken about the nature of the grid and peculiarities close to the poles. The YAC algorithm works seamlessly across poles.

Order (Integer, default `1`)

YAC supports 1st order (1) and 2nd order (2) conservative remapping. For further information see

E. Kritsikis, M. Aechtner, Y. Meurdesoif and T. Dubos, 2017: Conservative interpolation between general spherical meshes, Geosci. Model Dev., 10, 425-431, `https://doi.org/10.5194/gmd-10-425-2017,2017`.

Enforced conservation (Boolean, default `false`)

By setting this to `true`, local conservation is achieved by correcting the local weights obtained from the partial area contributions such that they locally add up to 1 as far as numerical precision allows.

Partial coverage (Boolean, default `false`)

By setting this to `true`, partially covered source cells will be interpolated.

Normalization (Enumeration: FRACAREA or DESTAREA, default DESTAREA)

In case *Partially coverage* is set to `true`, a request for FRACAREA will use the fractional (partially covered) area to normalise the weights. When DESTAREA is selected weights will be normalised with the destination area, the area of the target cell.

4.2.5. Bernstein-Bézier interpolation

As one option for a higher-order interpolation YAC provides an interpolation based on hybrid cubic spherical Bernstein-Bézier (HCSBB) polynomials or patches. This interpolation can be considered as an alternative for a bicubic interpolation which is frequently used for interpolations on block-structured grids with quadrilateral elements. The HCSBB patches do not conserve properties, but in contrast to the first-order conservative remapping it does deliver results which are not piecewise constant especially when interpolating from coarse to fine grids.

For further information see

- Peter Alfeld, Marian Neamtu, Larry L. Schumaker, 1996: Bernstein-Bézier polynomials on spheres and sphere-like surfaces, Computer Aided Geometric Design, Volume 13, Issue 4, Pages 333-349, https://doi.org/10.1016/0167-8396(95)00030-5.
- Xiaoyu Liu and Larry L Schumaker, 1996: Hybrid Bézier patches on sphere-like surfaces, Journal of Computational and Applied Mathematics, Volume 73, Issues 1-2, Pages 157-172 https://doi.org/10.1016/0377-0427(96)00041-6
- https://en.wikipedia.org/wiki/Bernstein_polynomial
- https://en.wikipedia.org/wiki/Bernstein-Bezier_curve

4.2.6. User file interpolation

Files containing the interpolation weights can be generated off-line e.g. using the CDO. By selecting the user file interpolation and providing a name for the file containing the specific weights YAC will use these weights for the remapping. As any other interpolation the user file interpolation is part of the interpolation stack, and other interpolations can be selected that shall be applied to target cells which are not included in the weight file.

Furthermore, the user can force YAC to write out the interpolation weights into a file by checking `Enforce write weight files` (see Sec. 4.1.1) which can then also be used in subsequent settings for the user file interpolation.

YAC location (Enumeration: CELL, EDGE or CORNER)

For the user file interpolation it is still required to specify whether the data are located on cell edges, cell corners (vertices) or somewhere in the cell center.

4.2.7. Source to target map

The source to target mapping has been introduced mainly to support the remapping of drainage from the hydrology model onto the ocean grid. The drainage is provided on grid cells at the land-sea border representing the river mouths. All source cells containing some drainage need to be assigned to a target cell, while a target cell may accept input from more than just one source cell. The source to target mapping ensures that a non-masked source cell is assigned to the nearest available (non-masked) target cell. Each of those source cells contributes with a weight of 1, no matter how many source cells are assigned to a particular target cell.

5. Saving a new XML file

To save the coupling that has been created through the previously described actions in the application the `Save` and `Save as ...` options are available in the File menu. In case the user did not save the coupling before, for both of these menu options firstly the `Saving File` dialogue will appear. After a successful saving of the file, the file status string will show the file name of the newly created coupling XML file. Subsequent plain save operations will overwrite the existing file. The GUI does not provide any automated versioning of XML files. This is left to the user with the `Save as ...` option. If user exits the program without saving the newly created file, a warning window will pop up to ask if the user want to save it before exiting.

6. Modification of an existing coupling configuration

The above allows for saving intermediate steps or the final setting. In order to continue the work to create or to modify an existing setting a valid coupling XML file can be loaded using `open` in the File menu. As described in the previous sections all parameters can be modified, and connections between any two fields can be activated or deactivated.

7. Multi component coupling

Once a pair of components has been loaded following the description above (and the coupling has been defined) additional component XML descriptions can be loaded via the menu entry `Couple add component`.

Alternatively we allow to use the "add component" mechanism to first load all components before starting with any further configuring.

When selecting `select active components` from the `Couple` menu a window similar to what is depicted in Fig. 5 will pop up. In the upper part the GUI lists the available component descriptions while the lower part displays those component pairs for which some coupling between transients has already been defined.

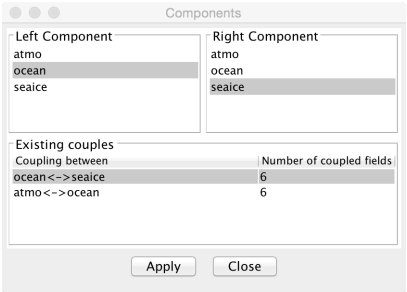

Figure 5: Selection of component pairs.

A new set of component pairs can now be selected from the list in the upper half. By clicking `Apply` the selected two components will now be displayed similar for Fig. 2 and the coupling can be defined. Already (partly) defined component pairs can also be selected from the lower part, by a click on `Apply` this particular pair is now on display for further editing.

A. Component XML description

In order to generate a coupling XML file the GUI requires a XML description of two model components which are going to be coupled as input. As can be seen in Fig. A1 the structure of such a component XML file is rather simple and can easily be generated with a simple text editor.

```
<?xml version="1.0" encoding="UTF-8"?>

<component
        xmlns="http://www.w3schools.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3schools.com component.xsd">

        <id>2</id>
        <name>ICON-ocean</name>
        <model>ICON</model>
        <simulated>ocean</simulated>

        <transient_grid_refs>
                <transient_grid_ref id="1" transient_ref="1" grid_ref="1" collection_size="2" />
                <transient_grid_ref id="2" transient_ref="2" grid_ref="1" collection_size="2" />
                <transient_grid_ref id="3" transient_ref="3" grid_ref="1" collection_size="1" />
                <transient_grid_ref id="4" transient_ref="4" grid_ref="1" collection_size="4" />
                <transient_grid_ref id="5" transient_ref="5" grid_ref="1" collection_size="4" />
                <transient_grid_ref id="6" transient_ref="6" grid_ref="1" collection_size="2" />
        </transient_grid_refs>

        <transients>
                <transient id="1" transient_standard_name="surface_downward_eastward_stress"/>
                <transient id="2" transient_standard_name="surface_downward_northward_stress"/>
                <transient id="3" transient_standard_name="sea_surface_temperature"/>
                <transient id="4" transient_standard_name="water_flux"/>
                <transient id="5" transient_standard_name="heat_flux"/>
                <transient id="6" transient_standard_name="albedo"/>
        </transients>

        <grids>
                <grid id="1" alias_name="R2B04_no_land"/>
        </grids>

</component>
```

Figure A1: Example of a component XML file

The `id` has to be selected such that it is unique among the component XML descriptions, likewise the `name` of the component shall be unique.

A list of `transients` links standard names with a unique local identifier. For each component these IDs can run from 1 to N.

The list of `transient_grid_refs` provides additional information for the transients. The IDs of the `transient_grid_refs` shall again run from 1 to N. These references are later used in the coupling XML description to access the transient information. The number of the `transient_ref` refers to the `transient` ID explained above. A `transient_ref` of 3 refers to `transient` ID 3, and thus to `heat_flux`. Likewise, the `grid_ref` of 1 refers to the grid ID 1 defined below. In this case we have only defined one grid, thus the `grid_ref` is set to 1 for all transients in the list of `transient_grid_refs`. Last but not least the collection size (size of bundle or number of vertical levels) has to be provided for each transient.

B. Coupling XML dependencies

A coupling XML file as it was produced by the GUI is shown in Fig. B1.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<coupling xmlns="http://www.w3schools.com"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.w3schools.com coupling.xsd">
   <redirect redirect_of_root="true" redirect_stdout="true"/>
   <components>
      <component id="1">
         <name>atmo</name>
         <model>ICON</model>
         <simulated>atmosphere</simulated>
         <transient_grid_refs>
            <transient_grid_ref collection_size="2" grid_ref="1" id="1" transient_ref="1"/>
            <transient_grid_ref collection_size="2" grid_ref="1" id="2" transient_ref="2"/>
            <transient_grid_ref collection_size="3" grid_ref="1" id="3" transient_ref="3"/>
            <transient_grid_ref collection_size="4" grid_ref="1" id="4" transient_ref="4"/>
            <transient_grid_ref collection_size="4" grid_ref="1" id="5" transient_ref="5"/>
            <transient_grid_ref collection_size="1" grid_ref="1" id="6" transient_ref="6"/>
         </transient_grid_refs>
      </component>
      <component id="2">
         <name>ocean</name>
         <model>ICON</model>
         <simulated>ocean</simulated>
         <transient_grid_refs>
            <transient_grid_ref collection_size="2" grid_ref="1" id="1" transient_ref="1"/>
            <transient_grid_ref collection_size="2" grid_ref="1" id="2" transient_ref="2"/>
            <transient_grid_ref collection_size="3" grid_ref="1" id="3" transient_ref="3"/>
            <transient_grid_ref collection_size="4" grid_ref="1" id="4" transient_ref="4"/>
            <transient_grid_ref collection_size="4" grid_ref="1" id="5" transient_ref="5"/>
            <transient_grid_ref collection_size="1" grid_ref="1" id="6" transient_ref="6"/>
         </transient_grid_refs>
      </component>
   </components>
   <transients>
      <transient id="1" transient_standard_name="surface_downward_eastward_stress"/>
      <transient id="2" transient_standard_name="surface_downward_northward_stress"/>
      <transient id="3" transient_standard_name="surface_fresh_water_flux"/>
      <transient id="4" transient_standard_name="total_heat_flux"/>
      <transient id="5" transient_standard_name="atmosphere_sea_ice_bundle"/>
      <transient id="6" transient_standard_name="sea_surface_temperature"/>
   </transients>
   <grids>
      <grid alias_name="grid1" id="1"/>
   </grids>
   <dates>
      <start_date>+1800-01-01T00:00:00.000</start_date>
      <end_date>+2100-01-01T00:00:00.000</end_date>
      <calendar>proleptic-gregorian</calendar>
   </dates>
   <timestep_unit>second</timestep_unit>
   <couples>
      <couple>
         <component1 component_id="1"/>
         <component2 component_id="2"/>
         <transient_couple transient_id="1">
            <source component_ref="1" transient_grid_ref="1"/>
            <target transient_grid_ref="1"/>
            <timestep>
               <source>10</source>
               <target>10</target>
               <coupling_period operation="accumulate">60</coupling_period>
               <source_timelag>0</source_timelag>
               <target_timelag>0</target_timelag>
            </timestep>
            <interpolation_requirements use_source_mask="true" use_target_mask="true">
               <interpolation method="n-nearest_neighbor" n="1" weighted="DISTANCE_WEIGHTED"/>
            </interpolation_requirements>
            <debug_mode at_source_after_interpolation="false" at_source_before_interpolation="false" at_target="false"/>
            <enforce_write_restart>false</enforce_write_restart>
            <enforce_write_weight_file filename="">false</enforce_write_weight_file>
         </transient_couple>
      </couple>
   </couples>
</coupling>
```

Figure B1: Example of a coupling XML file

For each component we have a list of N `transient_grid_refs` with consecutive IDs from 1 to N, where each `transient_grid_ref` ID may have a different `collection_size`, `grid_ref` and `transient_ref` (Fig. B2).

These lists are copied from the component XML descriptions. The `grid_ref` and `transient_ref` ids get updated (see Appendix A). Typically we will have

```
<transient_grid_refs>
   <transient_grid_ref collection_size="2" grid_ref="1" id="1" transient_ref="1"/>
   <transient_grid_ref collection_size="2" grid_ref="1" id="2" transient_ref="2"/>
   <transient_grid_ref collection_size="1" grid_ref="1" id="3" transient_ref="3"/>
   <transient_grid_ref collection_size="4" grid_ref="1" id="4" transient_ref="4"/>
   <transient_grid_ref collection_size="4" grid_ref="1" id="5" transient_ref="5"/>
   <transient_grid_ref collection_size="2" grid_ref="1" id="6" transient_ref="6"/>
</transient_grid_refs>
```

Figure B2: List of transient grid reference IDs

at least two entries in the `grid` element (Fig. B3), thus `grid_ref` becomes either 1 or 2 in this case. In the same way the `transient_ref` ids point to the appropriate entries in the transient list.

```
<grids>
   <grid alias_name="R2B04_no_land" id="1"/>
   <grid alias_name="R2B04" id="2"/>
</grids>
```

Figure B3: List of grids

In the couple (Fig. B4) the two components are listed with its component ID. Next a list of transient couples is provided which defines the source component identified by its component ID and its respective transient grid reference ID and the transient grid reference ID of the target.

```
<couples>
   <couple>
      <component1 component_id="2"/>
      <component2 component_id="1"/>
      <transient_couple transient_id="5">
         <source component_ref="2" transient_grid_ref="5"/>
         <target transient_grid_ref="5"/>
```

Figure B4: Coupling dependencies