Deutscher Wetterdienst
Wetter und Klima aus einer Hand

DWD

DKRZ
DEUTSCHES KLIMARECHENZENTRUM

KIT
Karlsruher Institut für Technologie

natESM

Federal Ministry
of Research, Technology
and Space

# YAC in general
## and
# its usage in ICON

Moritz Hanke (DKRZ)

# Overview

- Coupling library (not a framework)
- C, Fortran, and Python interface
- Features
  - Coupling of 2D fields on the sphere
  - All commonly used global and regional grids
  - Arbitrary decompositions
  - Parallel online weight computation
- Development:
  - started as joined project between DKRZ and MPIM
  - now developed by DKRZ
  - 2011: start of development
  - 2014: YAC 1
  - 2021: YAC 2
  - 2023: YAC 3
  - latest: v3.8.0

# Overview cont.

- Interpolation
  - Conservative (1st and 2nd order), N-Nearest-Neighbour, Linear, Bernstein-Bézier polynomial, RBF, Creep fill, Run-off-mapping, … (see [1])
  - Interpolation stack concept (see [2])
- Links
  - Download
    https://gitlab.dkrz.de/dkrz-sw/yac
  - Documentation
    https://dkrz-sw.gitlab-pages.dkrz.de/yac/
- Support
  - Issues in gitlab
  - Contact direct contact (hanke@dkrz.de, Mattermost: @k202077)

[1]: https://dkrz-sw.gitlab-pages.dkrz.de/yac/d0/da2/interp_methods.html
[2]: https://dkrz-sw.gitlab-pages.dkrz.de/yac/db/dbc/interp_stack.html

# Concepts

- Instance
  - Basic object to which all others are associated to
  - Default instance available
  - Is associated with with MPI communicator
    - By default MPI_COMM_WORLD is used
    - Can be generated explicitly via MPI-handshake
- Component
  - Subgroup of processes (e.g. ICON-atmosphere)
  - Process can be part of more than one component
- Grid
  - Each process defines its part of the grid (vertices coordinates and cell-connectivity)
  - There can be more than one grid on a process
  - Grid can be shared by components
  - Global ids identifies unique vertices/cells/edges across processes
- Mask
  - Core-mask deactivates duplicated and degenerated vertices/cells/edges
  - Field-mask marks vertices/cells/edges that contain valid data (e.g. land-mask)
- Points
  - Coordinates and location (vertex, cell, or edge) at which a field can be defined (e.g. cell center)
- Field
  - Associates a data field (e.g. sea-surface-temperature) with a grid, (field mask,) and a points
  - Collection size sets number of vertical levels or ”categories”
    (e.g. ocean_sea_ice_bundle; ice thickness, snow thickness, ice concentration)
- Interpolation
  - Basically a distributed matrix-vector-multiplication
  - (Weight files contains global weight matrix)
- Couple
  - Configures the data exchange between a source and a target field
  - Defined by source/target component/grid/field, coupling period, time reduction operation, lag, interpolation stack (+ some optional parameters)

# Basic usage

I. Initialise YAC
   - Usually involves splitting of MPI_COMM_WORLD

II. Definition of local configuration on each process (via UI)
   - Components, Grids, Masks, and Fields

III. Definition of global configuration (via UI or configuration-file)
   - Can be done by any number of processes
   - Calendar, start- and end-date of run, couples, debugging options

IV. End of definition
   - Synchronises definitions between all processes
   - Generates interpolations

V. Data exchange
   - Asynchronous put-operation for sending
   - Blocking get-operation for receiving (async. also available)

VI. Finalisation

(https://dkrz-sw.gitlab-pages.dkrz.de/yac/da/d1b/usage.html)

# Configuration file

- Defines global configuration
- Arbitrary number of files can be read by arbitrary processes
- YAML (full support for v1.2) or JSON format
- Allows flexible configuration of couples without need for recompiling
- Side note:

  - Function `InterpolationStack.from_string_json` can generate an interpolation stack from a JSON-formated string to be used in the UI

(https://dkrz-sw.gitlab-pages.dkrz.de/yac/dd/dfa/yaml_file.html)

# Configuration file example (atmo<->wave)

```
components:
  atm: &atm
    component: atmo
    grid: icon_atmos_grid
  wave: &wave
    component: wave
    grid: icon_waves_grid
  atm2wave: &atm2wave
    source: *atm
    target: *wave
    time_reduction: none
    src_lag: 1
    tgt_lag: 1
  wave2atm: &wave2atm
    source: *wave
    target: *atm
    time_reduction: none
    src_lag: 1
    tgt_lag: 1
interp_stacks:
  hcsbb_interp_stack: &hcsbb_interp_stack
    interpolation:
      - bernstein_bezier
      - nnn:
          n: 4
      - fixed:
          user_value: -999.9
```

```
timestep_unit: ISO_format
calendar: proleptic-gregorian

coupling:
  - <<: [ *atm2wave, *hcsbb_interp_stack ]
    coupling_period: ${couplingTimeStep}
    field: [zonal_wind_in_10m,
            meridional_wind_in_10m,
            fraction_of_ocean_covered_by_sea_ice]
  - <<: [ *wave2atm, *hcsbb_interp_stack ]
    coupling_period: ${couplingTimeStep}
    field: [roughness_length]
```

**(Remark: Start and end time of the run is defined by ICON via the UI.)**

# Example (generator.py and plotter.py)

```
import os
import yac
import numpy as np
import matplotlib.pyplot as plt

yac.def_calendar(yac.Calendar.PROLEPTIC_GREGORIAN)

instance = yac.YAC()

instance.def_datetime("2020-01-01T00:00:00", "2020-01-02T00:00:00")

comp = instance.def_comp("generator")
assert comp.size == 1

bounds = [0.5*np.pi, 1.5*np.pi, -0.25*np.pi, 0.25*np.pi]
resolution = (360,180)
lon = np.linspace(bounds[0],bounds[1],resolution[0])
lat = np.linspace(bounds[2],bounds[3],resolution[1])
grid = yac.Reg2dGrid("gen_grid", lon, lat)
points = grid.def_points(yac.Location.CORNER, lon, lat)

collection_size = 1
field = yac.Field.create(
  "noise_field", comp, points, collection_size,
  "3", yac.TimeUnit.HOUR)

instance.enddef()

while field.datetime < instance.end_datetime:
  print(f"generating {field.name} at {field.datetime}")
  field.put(np.random.rand(collection_size, grid.nbr_corners))
```

```
import os
import yac
import numpy as np
import matplotlib.pyplot as plt

yac.def_calendar(yac.Calendar.PROLEPTIC_GREGORIAN)

instance = yac.YAC()

comp = instance.def_comp("plotter")
assert comp.size == 1

bounds = [0, 2*np.pi, -0.5*np.pi, 0.5*np.pi]
resolution = (360,180)
vlon = np.linspace(bounds[0],bounds[1],resolution[0])
vlat = np.linspace(bounds[2],bounds[3],resolution[1])
grid = yac.Reg2dGrid("plot_grid", vlon, vlat)
clon = (0.5*(vlon[:-1]+vlon[1:]))
clat = (0.5*(vlat[:-1]+vlat[1:]))
points = grid.def_points(yac.Location.CELL, clon, clat)

collection_size = 1
field = yac.Field.create(
  "noise_field", comp, points, collection_size, "6", yac.TimeUnit.HOUR)

instance.read_config_yaml("coupling.yaml")

instance.enddef()

while field.datetime < instance.end_datetime:
  print(f"plotting {field.name} at {field.datetime}")
  field_data, info = field.get()
  for i in range(field_data.shape[0]):
      plt.imshow(field_data[i,:].reshape(len(clat),len(clon))[::-1,:],
                 extent=[clon[0],clon[-1],clat[0],clat[-1]])
      plt.title(f"{field.name} - {field.datetime}")
      plt.colorbar()
      plt.savefig(f"./{field.name}_{field.datetime}.png")
      plt.clf()
```

# Example cont. (coupling.yaml and usage)

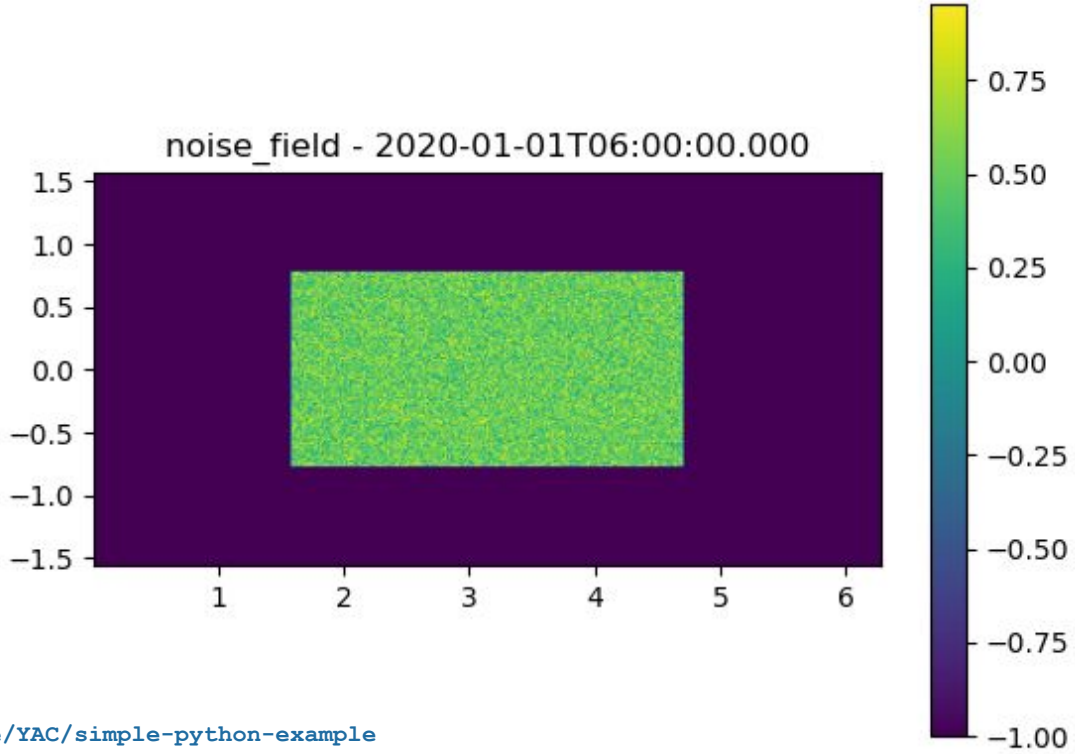
```
timestep_unit: hour
coupling:
  - src_component: generator
    src_grid: gen_grid
    tgt_component: plotter
    tgt_grid: plot_grid
    field: noise_field
    coupling_period: 6
    interpolation:
      - average
      - fixed:
          user_value: -1.0
```

```
$ export PYTHONPATH=$YAC_BUILD_DIR/python:$PYTHONPATH

$ mpirun \
    -n 1 python3 generator.py : \
    -n 1 python3 plotter.py

generating noise_field at 2020-01-01T00:00:00.000
plotting noise_field at 2020-01-01T00:00:00.000
generating noise_field at 2020-01-01T03:00:00.000
generating noise_field at 2020-01-01T06:00:00.000
generating noise_field at 2020-01-01T09:00:00.000
generating noise_field at 2020-01-01T12:00:00.000
plotting noise_field at 2020-01-01T06:00:00.000
generating noise_field at 2020-01-01T15:00:00.000
generating noise_field at 2020-01-01T18:00:00.000
plotting noise_field at 2020-01-01T12:00:00.000
generating noise_field at 2020-01-01T21:00:00.000
plotting noise_field at 2020-01-01T18:00:00.000
```

# Example cont. (output)




**Example repository:**
**https://gitlab.dkrz.de/YAC/simple-python-example**

# YAC in ICON

- ICON is SPMD (most other models are MPMD)
- src/parallel_infrastructure/mo_mpi.f90
  - Splits MPI_COMM_WORLD into communicator for ICON, (ComIn), and (YAC) using MPI handshake [1]
- Component driver (e.g. `SUBROUTINE construct_atmo_model(...)`)
  - Initialised and finalises YAC instance

[1]: https://gitlab.dkrz.de/dkrz-sw/mpi-handshake

# YAC in ICON cont.

- src/namelists/mo_coupling_nml.f90
  - Reads in ICON coupling configuration during initialisation
  - Example:

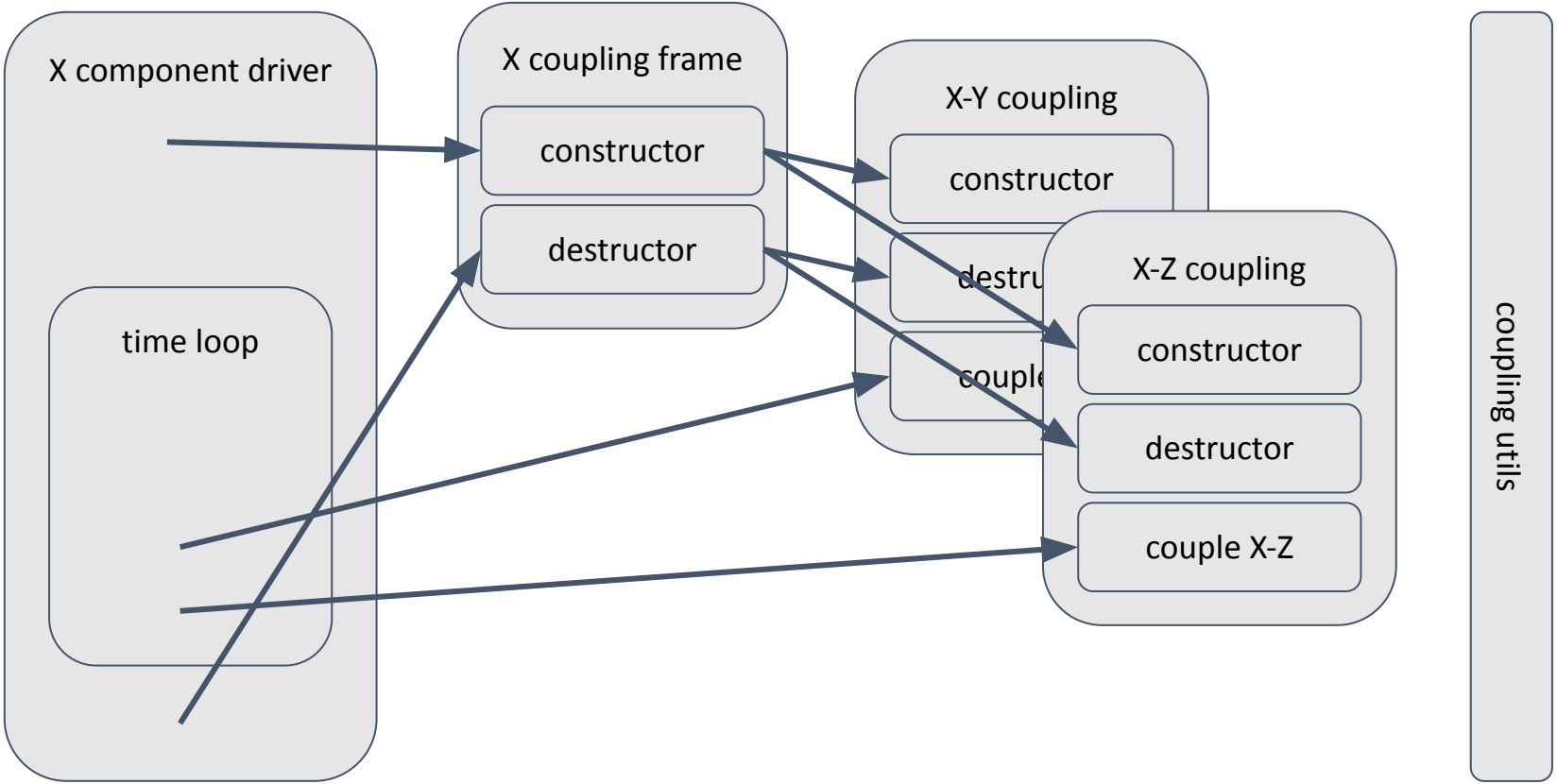
```
&coupling_mode_nml
coupled_to_ocean     = .true.
coupled_to_hydrodisc = .true.
/
```

- src/configure_model/mo_coupling_config.f90
  - Query routines for coupling mode
    (e.g. `LOGICAL FUNCTION is_coupled_to_hydrodisc()`)

# YAC in ICON cont.

- src/coupling/mo_X_coupling_frame.f90
  - Constructor and destructor of coupling in component X (atmo, ocean, wave)
    - Defines local configuration (component, grid, and points)
- src/coupling/mo_X_Y_coupling.f90
  - Constructor and destructor of coupling with component Y (atmo, ocean, wave, cleo, hydrodisc, …) in component X (atmo, aes, nwp, ocean, wave)
    - Defines local configuration (mask and field)
  - Routine for data exchange (e.g. `couple_wave_to_atmo`)
- src/coupling/mo_coupling_utils.f90
  - Coupling interface routines (`cpl_*`)
    - Provide easy access to YAC-routines (`yac_*`)
- src/coupling/mo_output_coupling.f90
  - Module that allows access to all var_list fields to external components via YAC

# YAC in ICON cont.



# Questions?