New Python AMIP reader for EC Earth 4

OASIS to pyOASIS conversion
New more flexible Python AMIP reader

Computacional Earth Sciences - Performance Team
BSC Summer Internship Programme

Intern: **Rodrigo Martín**
Supervisor: Mario Acosta
From 21/06/2021 to 29/08/2021
Academic year: 2021/2022

Abstract

The new flexible and highly configurable Python AMIP reader development for EC Earth 4 is documented here as well as a light analysis of the old Fortran implementation with an explanation of main differences when converting Fortran code from OASIS to pyOASIS. The new Python implementation has demonstrated equivalent results and comparable performance to the old Fortran AMIP reader while having a largely improved *namelist* configuration interface with huge flexibility for reading and sending any coupling field. This new version of the AMIP reader also maintains backwards compatibility with the old *namelist* configuration files in EC Earth v3.3.3.1 and EC Earth v3.2.2 (PRIMAVERA production).

Index of contents

My work in context

My internship is framed in the field of Earth Sciences. I have been part of the Performance group in the Computational Earth Sciences team of the Earth Sciences department under the BSC Summer Internship Program 2021, working on the conversion from Fortran to Python of the AMIP reader for the EC Earth 4 project using the pyOASIS API from the CERFACS OASIS library [1]. All the info about the development can be found in the issue ec-earth4#13.

AMIP

EC-Earth (*Earth System Model for the Climate Model Intercomparison Project*) is a model of the Earth that is developed collaboratively in the European Earth System Model consortium, whose current version is EC Earth 3. The model is composed of different modules that describe atmosphere, ocean, sea ice, land surface, dynamic vegetation, atmospheric composition, ocean biogeochemistry and the Greenland ice sheet. The basic configuration of EC-Earth consists of the IFS atmosphere module and the NEMO ocean module (which includes the LIM3 sea ice module). The coupling of variables between the modules is done through the OASIS3-MCT coupler, which provides a means to exchange variables coupled in the different modules. The replicability tests consist of comparing the executions of the same code in coupled (CMIP) and only atmosphere configurations (AMIP) carried out by different institutions using their respective platforms.

The AMIP reader provides realistic sea surface temperature and sea ice data from 1979 to near present. These data are obtained from reading already defined netCDF files. It is not meant to be used for climate change prediction, an endeavor that requires a coupled atmosphere-ocean model (e.g., see AMIP's sister project CMIP[2]). As can be seen in Figure 1, this allows scientists to focus on the atmospheric model without the added complexity of ocean-atmosphere feedback (CIMP).

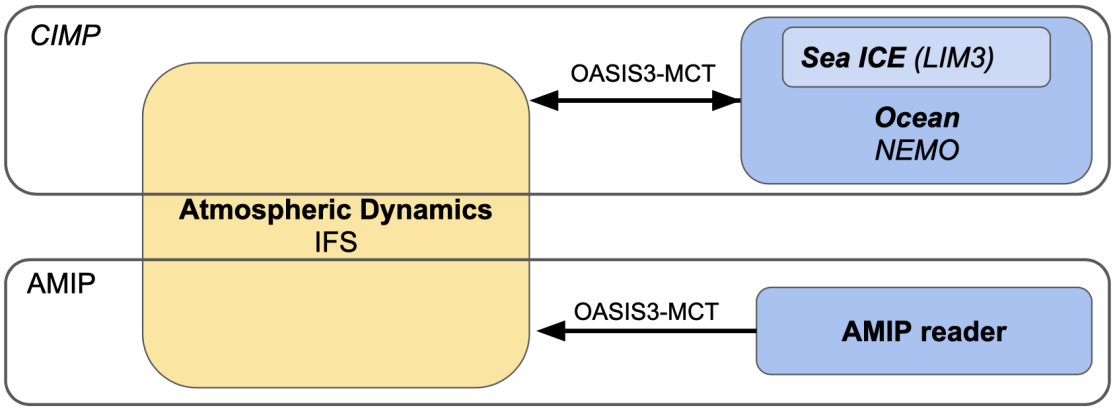

Figure 1. Fragment of the coupling links between components in AMIP vs CIMP.

My objectives

The main objective was to switch the current implementation of the AMIP reader from Fortran to Python. In this process, it was necessary to convert all coupling exchange code from OASIS3-MCT to pyOASIS, which is currently in development. In addition, adding extensibility to the AMIP reader interface taking advantage of Python's flexibility was a secondary objective, as well as refactoring the current implementation improving its reusability and maintainability. All objectives were accomplished successfully and will be discussed in the following pages.

Old Fortran implementation

In order to carry out a successful refactor, I created a few activity diagrams of the original Fortran implementation. Some of these diagrams can be seen in the figure 2 and show a messy implementation with the use of a lot of different module global variables, which is probably caused by the superposition of one code patch after another; the entry and exit points for each subroutine are not clear and the naming is a bit unconventional. As this implementation is too complex for a simple component like the AMIP reader, a complete code refactor is necessary (leaving the coupling interface as it is).

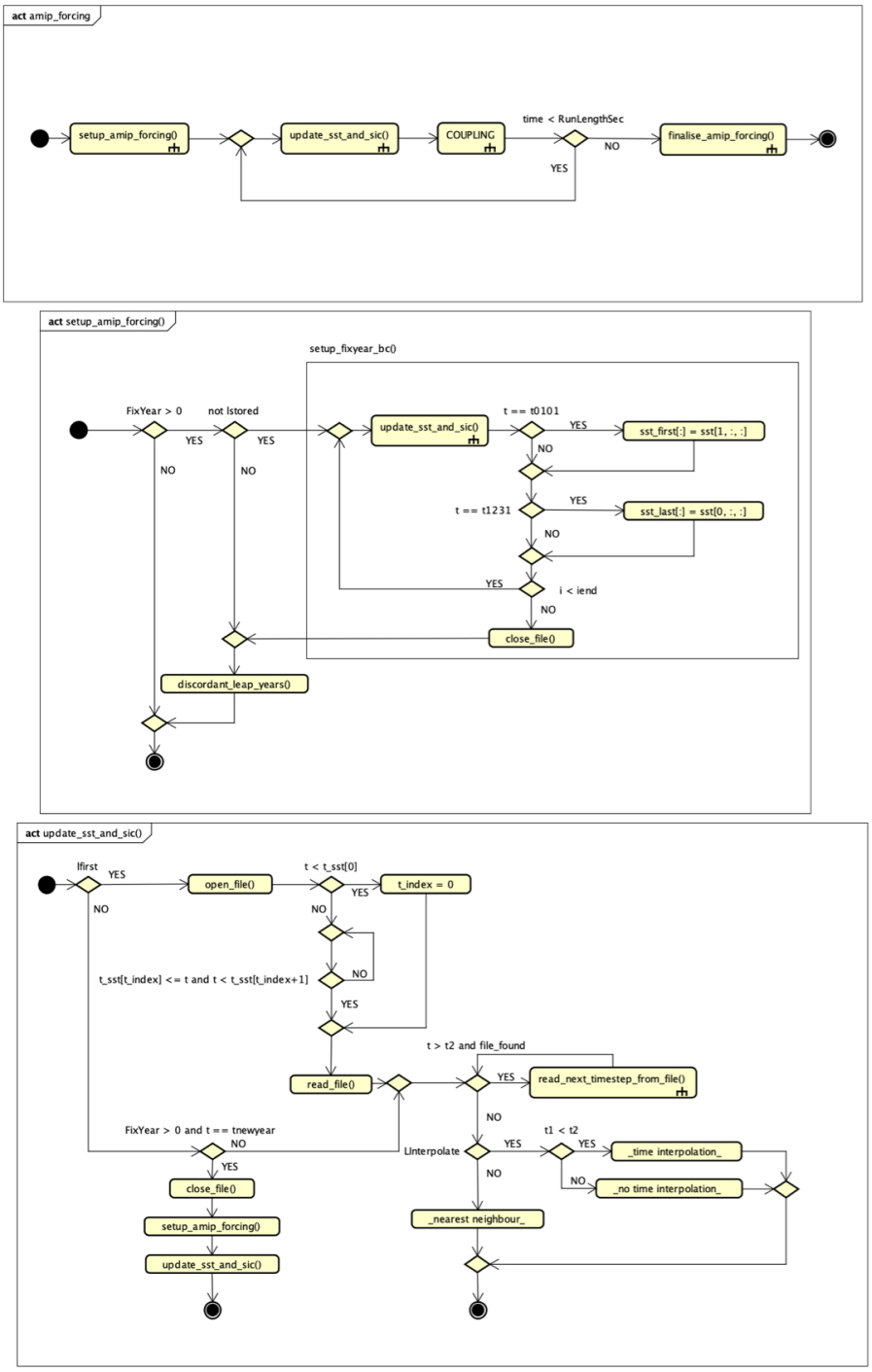

Figure 2. Activity diagram of the Fortran implementation, complete diagram on ec-earth4#13.

OASIS to pyOASIS conversion

In order to carry out a first conversion, easier to understand, develop and debug, I created a basic pyOASIS "prototype-model" from the *spoc_communication* example from OASIS3-MCT4.0 repository[3]. This toy model consists of two components ('ocean' and 'atmos') and implements basic coupling exchanges between them and its development is kept in the ES GitLab in the repository *rmarti1/oasis-toy-model-python*[4].

While performing this conversion, I identified the most common differences between Fortran and Python when converting from OASIS to pyOASIS, which can also be found in the issues #1 and #2 of the repository:

#1: Python vs. Fortran integer division

Fortran integer division results in the truncated output of the operands. However, since Python3, Python automatically casts the result of the division between two integers to float. This difference can be addressed in the Python code by casting the result in Python **of each division** like so: *int(a/b)* or use the integer division operator: *a//b*.

#2: Fortran column-major in netCDF files

While writing two components' grids, one written in Python and the other in Fortran, and using netCDF files in the process, it is important to keep in mind that the Fortran **approach to array-like structures is column-major**, while Python's default approach is row-major. This difference can be overcome by subjecting one of the languages to the other. netCDF files stored in Fortran will be written in a column-major fashion, so in order to retrieve the data in Python and be able to process and store it with the same format and shape, we must transpose the arrays by *numpy.transpose(a)*.

Performance differences between pyOASIS and OASIS

Taking advantage of the simplicity of the "prototype-model" I was able to create a new branch in the same repository serving as an example on the performance difference between pyOASIS and OASIS, as well as OASIS performance analysis itself when sending two coupling fields at the same time.

The "prototype-model" has been modified from the master branch. It consists of two components ('ocean' and 'atmos') and implements coupling exchanges only from 'ocean' to 'atmos' at the same time on every iteration of two coupling fields of the same size. 'ocean' runs first in Fortran and then in Python, always communicating with 'atmos' running in Fortran.

The execution of the model outputs almost the same results as shown in figure 3. It looks like it does not depend on the coupling field itself, but in what order you call put for each one of them. Python and Fortran display the same behavior, indicating that pyOASIS does not provoke a noticeable performance penalty. However, the incremental runtime trend along the simulation might indicate room for improvement for OASIS performance as a whole.

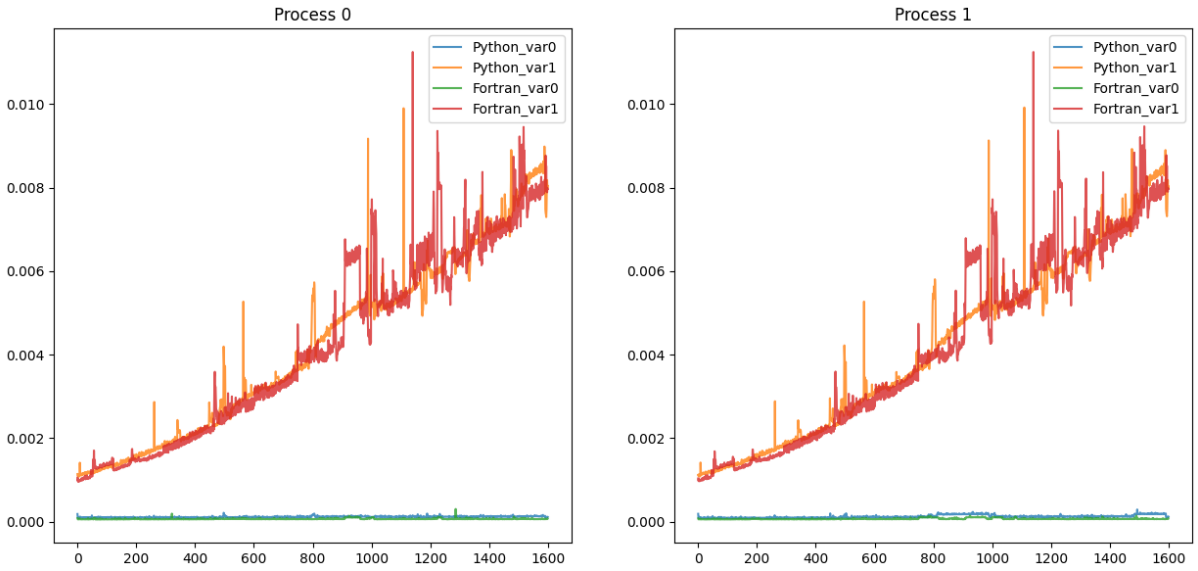

Figure 3. Execution time for the subroutine *OASIS_VAR.put()* for each variable in Python and Fortran.
Source: rmarti1/oasis-toy-model-python#performance branch.

New Python AMIP reader

All the documentation below is directly taken from the *README.md* file from the new Python AMIP reader repository: *rmarti1/python-amip-reader* [5], which also contains tests for running both implementations (Fortran code is taken from EC Earth v3.3.3.1) using the current setup for IFS and AMIP from EC Earth v3.3.3.1. It is highly recommended to check the documentation in the repository in order to get the most up-to-date and intra-linked documentation.

The new Python AMIP reader is a flexible, highly-customizable and ready out-of-the-box component with the following features:

- Read and send any field by including it in the new namelist file and making sure the input netCDF files follow a common structure. The following fields have been tested:
    - SST (Sea Surface Temperature) and SIC (Sea Ice Concentration) constructed mid-month fields (exactly as the AMIP reader from EC Earth v3.3.3.1).
    - SST (Sea Surface Temperature) and SIC (Sea Ice Concentration) constructed daily fields (exactly as the AMIP reader for PRIMAVERA from EC Earth v3.2.2 (PRIMAVERA production)).
    - Gridded emissions data as in these TM5 routines from EC Earth in *tm5mp/proj/co2emission_read__co2.F90*.
- Backwards compatibility with all the namelist files used by the old Fortran AMIP reader.
- Dynamic *grids.nc* and *masks.nc* definitions for OASIS based on the input netCDF files. It will not override any existing grid with the same name (as stated in the OASIS 4.0 user guide), providing backwards compatibility with all current implementations.

All the source code for the new implementation can be found under the folder *sources/amip-forcing/src/python/* of this repository.

Python dependencies

The new Python implementation relies on the Python modules *f90nml*, *netCDF4* and *numpy*. In order to install these dependencies in your local machine you may execute the following command.

```
pip3 install f90nml netCDF4 numpy
```

Or if running in MN4 (assuming the modules *impi/2017.4* and *mkl/2017.4* are loaded):

```
module load python/3.6.1
module load netcdf/4.2
```

Usage

The new Python AMIP reader gets all its configuration from a new *namelist.amip* file. However, it can also be used as a drop-in replacement for the Fortran implementation without any additional configuration for reading and sending the SST and SIC fields for both EC Earth v3.3.3.1 and EC Earth v3.2.2 (PRIMAVERA production). See the backwards compatibility section for more info.

In order to take advantage of all the new functionalities and have more control over the coupling fields and input data, it is highly recommended to use the new *namelist.amip* configuration structure. Although not necessary, it is also recommended to avoid setting the AMIP grid in the

grids.nc, *masks.nc* and *areas.nc* files in advance, as the new AMIP reader is able to write them at runtime.

New namelist file

Although the new AMIP reader works with the old namelist.amip configuration files, the new namelist configuration file provides greater flexibility to the user when reading and operating with the input data. Here is a scheme for the new namelist generator bash script taken from *namelist_python.amip.sh*:

```
cat << EOF
!--------------------------------------------------------------------------
&NAMAMIP
!--------------------------------------------------------------------------
    RunLengthSec = ${leg_length_sec}
    TimeStepSec  = ${cpl_freq_amip_sec}
    StartYear    = ${leg_start_date_yyyymmdd:0:4}
    StartMonth   = ${leg_start_date_yyyymmdd:4:2}
    StartDay     = ${leg_start_date_yyyymmdd:6:2}
    FixYear      = ${ifs_cmip_fixyear}
    # Vars(1,:) = <id> <grid_name> <oasis_name> <file_pattern> <netcdf_variable>
<yref_min> <yref_max> <timestep> <interpolate> <scale_factor> <offset> <min>
<max>
    # ...
    # Vars(n,:) = ...
    LDebug       = false
!--------------------------------------------------------------------------
/
EOF
```

The main difference between the old and the new namelist files is the substitution of the fields exclusively defined for SST and SIC grids (*FileListSST*, *FileListSIC* and *LInterpolate*) for an agnostic array of fields *Vars(:,:)*. *Vars(:,:)* shape is (n,13), being n the number of fields to exchange and each coupling field declaration must follow the structure below:

Index	Name	Type	Description
0	id	string	Unique identifier for the field.
1	grid_name	string	Grid name (must match one of the grids declared in the namcouple file).
2	oasis_name	string	Variable name (must match one of variables under the grid_name declared in the namcouple file).
3	file_pattern	string	File pattern of the input netCDF files. I.e: *'HadISST2_prelim_0to360_alldays_sst_[year].nc'*. All patterns must be between brackets '[]' and currently the only accepted pattern is: *year*.
4	netcdf_variable	string	Variable to read in the netCDF files.
5	yref_min	int32	Reference year for the time variable in the netCDF files.
6	yref_max	int32	The last year of the netCDF files.
7	timestep	'monthly' | 'daily'	The step in number of days of the time variable in the netCDF files.
8	interpolate	true | false	Whether to interpolate or not (disabled if the timestep is set to daily).
9	scale_factor	float64	Scale factor applied after reading the input data.
10	offset	float64	Offset applied after reading the input data.
11	min	float64 | None*	Minimum value used to clip the input data before submitting the field.
12	max	float64 | None*	Maximum value used to clip the input data before submitting the field.

* In order to set a section as None you must leave blank the content between its comas. I.e: for setting the last section (max) as None in AMIP_sst, the field Vars(1,:) ends with "..., 273.15, 271.38, ,". See the example of the new namelist file for the complete configuration.

Below is an example of the old *namelist.amip* file used for AMIP forcing in EC Earth v3.3.3.1 next to the new equivalent one created by the script above:

Example of the old namelist file

```
!-----------------------------------------------------------------------
&NAMAMIP
!-----------------------------------------------------------------------
    RunLengthSec = 5097600
    TimeStepSec  = 86400
    StartYear    = 1991
    StartMonth   = 01
    StartDay     = 01
    FixYear      = 0
    FileListSST  =
'tosbcs_input4MIPs_SSTsAndSeaIce_CMIP_PCMDI-AMIP-1-1-3_gn_187001-201706.nc'
    FileListSIC  =
'siconcbcs_input4MIPs_SSTsAndSeaIce_CMIP_PCMDI-AMIP-1-1-3_gn_187001-201706.nc'
    LDebug       = false
    LInterpolate = true
!-----------------------------------------------------------------------
/
```

Example of the new namelist file

```
!-----------------------------------------------------------------------
&NAMAMIP
!-----------------------------------------------------------------------
    RunLengthSec = 5097600
    TimeStepSec  = 86400
    StartYear    = 1991
    StartMonth   = 01
    StartDay     = 01
    FixYear      = 0
    Vars(1,:) = 'AMIP_sst_monthly', 'AMIP', 'AMIP_sst',
'tosbcs_input4MIPs_SSTsAndSeaIce_CMIP_PCMDI-AMIP-1-1-3_gn_187001-201706.nc',
'tosbcs', 1870, 2016, 'monthly', true, 1, 273.15, 271.38, ,
    Vars(2,:) = 'AMIP_sic_monthly', 'AMIP', 'AMIP_sic',
'siconcbcs_input4MIPs_SSTsAndSeaIce_CMIP_PCMDI-AMIP-1-1-3_gn_187001-201706.nc',
'siconcbcs', 1870, 2016, 'monthly', true, 0.01, 0, 0, 1,
    LDebug       = false
!-----------------------------------------------------------------------
/
```

Backwards compatibility

The new AMIP reader is backwards compatible with both EC Earth v3.3.3.1 and EC Earth v3.2.2 (PRIMAVERA production) namelist files. It achieves this by internally converting the configuration from the old namelist files to the new representation when reading the namelist file (source code in function *read_namelist()* in *amip_utils.py*). Here is the internal conversion of the SST coupling field when using each old namelist file:

EC Earth v3.3.3.1 (AMIP forcing)

Property	Value
id	'AMIP_sst_monthly'
grid_name	'AMIP'
oasis_name	'AMIP_sst'
file_pattern	*FileListSST* content from the old namelist file.
netcdf_variable	'tosbcs'
yref_min	1870
yref_max	2016
timestep	'monthly'
interpolate	*LInterpolate* content from the old namelist file.
scale_factor	1
offset	273.15
min	271.38
max	None

EC Earth v3.2.2 (PRIMAVERA production)

Property	Value
id	'AMIP_sst_daily'
grid_name	'PSST'
oasis_name	'AMIP_sst'
file_pattern	*AmipFileRoot* content from the old namelist file appending '_sst_[year].nc'.
netcdf_variable	'sst'
yref_min	1850
yref_max	2016
timestep	'daily'
interpolate	False
scale_factor	1
offset	0
min	None
max	None

Add a new field

Adding a new field solely consists on adding the input netCDF files to the runtime folder of the program and adding a new entry to the *Vars(:,:)* field in the namelist following the structure mentioned above (see the new namelist filesection).

Input netCDF files

All input netCDF files must contain the following variables (assuming the dimensions time, lat and lon are defined):

Variable	Type	Shape	Description
time	double	(time)	Days since the reference year defined as yref_min in its namelist entry.
lat or latitude	double	(lat)	Latitude.
lon or longitude	double	(lon)	Longitude.
Input variable name	double or float	(time, ..., lat, lon)	The variable name must match the netcdf_variable value in its namelist entry.

Hardcode an operation (inner workings)

If it is necessary to hardcode an operation with respect to one of the coupling fields, it is highly recommended to edit the code within the *AMIPVar* class. Right now, there are already hardcoded operations used to match the old Fortran implementation or read an specific index of the input netCDF variable. Here are some examples:

```
logging.debug('{}: no time interpolation t,t2 {} {}'.format(self.id, t_local,
self.t2))
# Hardcoded to match Fortran's AMIP clipping avoidance
if self.oasis_name == 'AMIP_sst':
    var_min = np.NINF


# This section can be "hardcoded" by the user
# Hardcoded CO2 emissions data. Read only sector id 1: Energy
if self.var_name == 'CO2_em_anthro':
    raw_field = raw_field[1]
```

While the Python implementation was originally inspired by the Fortran implementation, it has now been refactored to take advantage of Python's OOP and simpler modules. For a deeper understanding, in figures 4 and 5 are the simplified sequence and class diagrams of the current implementation.

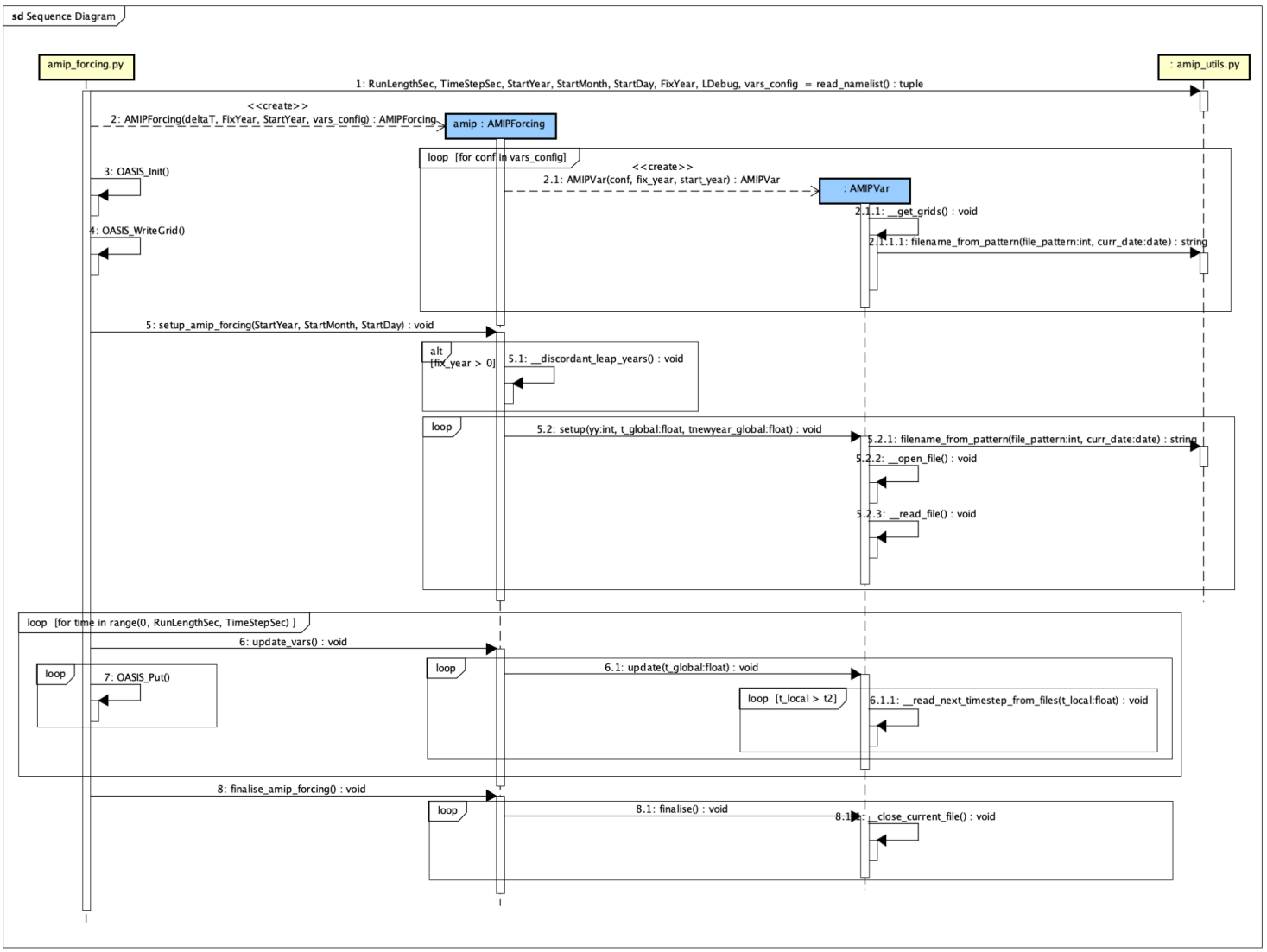

Figure 4. Sequence diagram of the Python AMIP reader implementation.

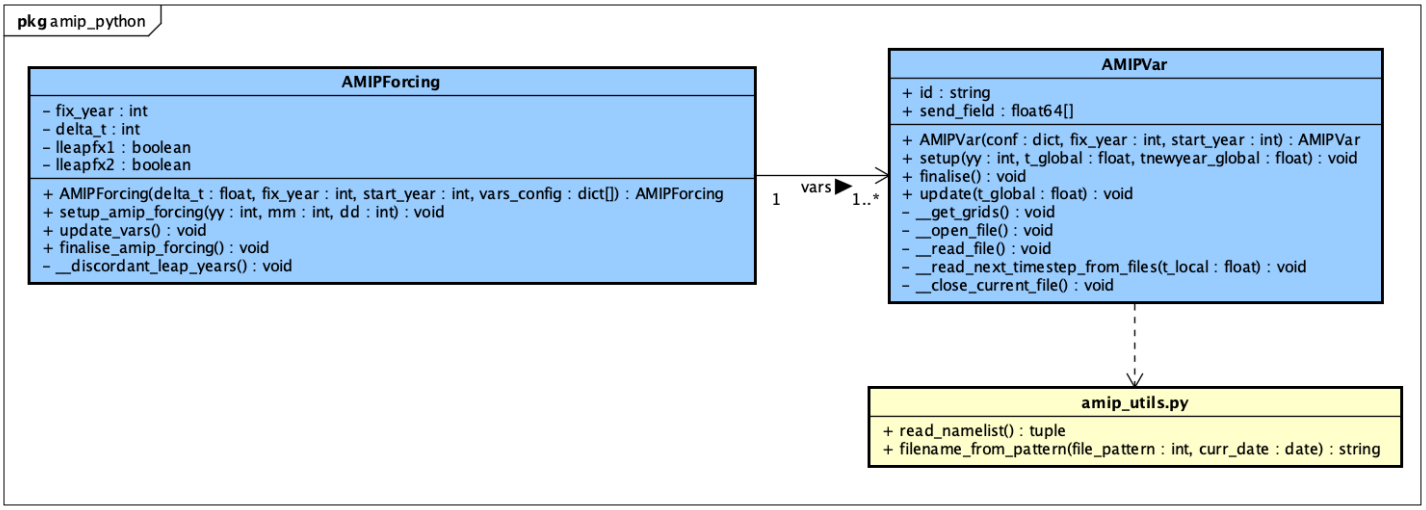

Figure 5. Class diagram of the Python AMIP reader implementation.

Run an example

The repository's *README.md* contains all the information about running an example. It consists on executing one of the implementations (Fortran or Python) along with a mock component acting as IFS. This mock component simulates IFS reception behavior as designed in EC Earth v3.3.3.1. The detailed specification can be found below, with the coupling interval and run length taken directly from the *namelist.amip* file.

```
# Constants
NAMELIST_FILE_NAME = 'namelist.amip'
L080_NX = 35718 # number of LAND grid cells at T159 resolution
L128_NX = 88838  # number of LAND grid cells at T255 resolution
L256_NX = 348528 # number of LAND grid cells at T511 resolution
```

For a more detailed explanation on how to run an example in a local machine or MareNostrum 4, please refer to section *Run an example* in the repository's *README.md* file.

Visualize netCDF output files as GIF

In the new Python AMIP reader repository there is also the possibility to visualize the results from an experiment and save them in a GIF file. First, you must first fix the grid and time axis of the EXPOUT generated files and then run the script *nc_to_gif.py*. The exact process can be found under the section *Visualize netCDF output files* in the repository's *README.md* file.

The figures 6 and 7 are visualization examples of the EXPOUT generated files of a run from 1990-01-01 to 1991-01-01 with interpolation activated and L128 IFS grid.

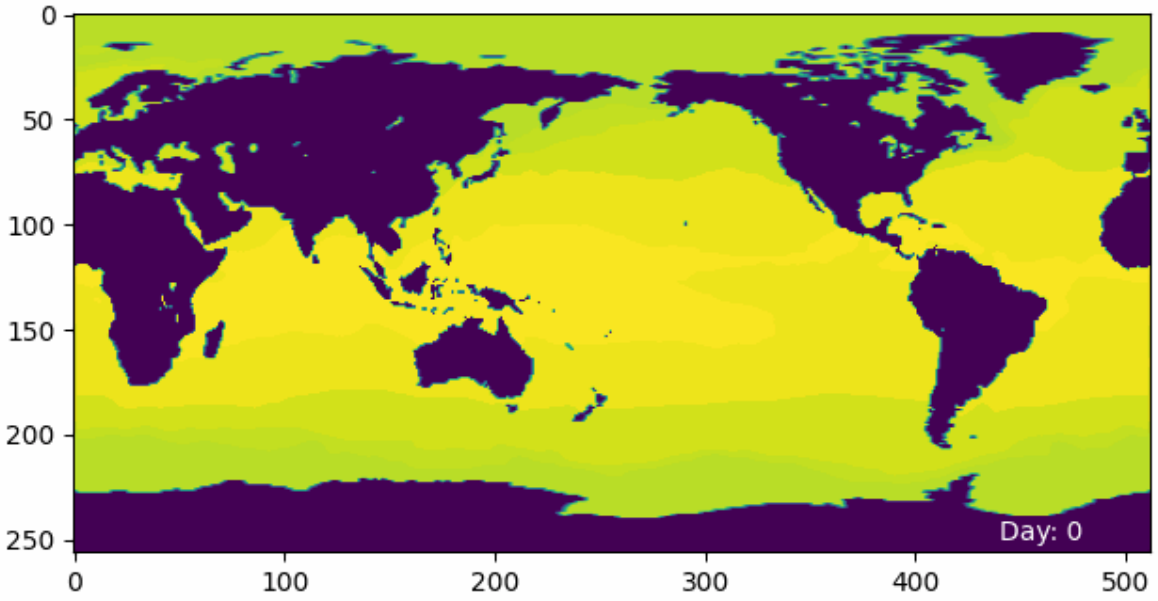

Figure 6. A_SST.nc output from Python's implementation. Source: ec-earth4#13.

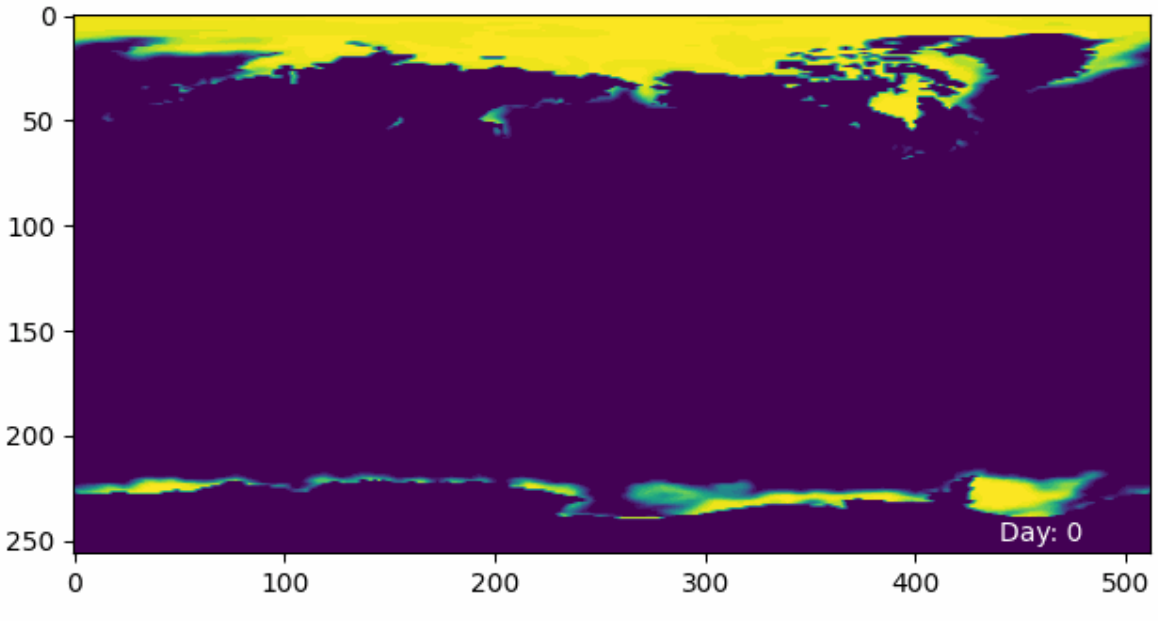

Figure 7. A_Ice_Frac..nc output from Python's implementation. Source: ec-earth4#13.

Conclusion

During my internship, my mission was to switch the current implementation of the AMIP reader from Fortran to Python. In this process, it was necessary to convert all coupling exchange code from OASIS3-MCT to pyOASIS. A second objective was to add extensibility to the AMIP reader interface taking advantage of Python's flexibility as well as to refactor the current implementation improving its reusability and maintainability. As shown above, all objectives were accomplished successfully.

One of the main difficulties was matching exactly the same results in Python and Fortran due to differences in floating point precision differences. I finally switched all Fortran code to *float64* to obtain the maximum precision possible, which is checked by the automated tests in the repository. This way, if it is necessary to reduce the precision bits, we are sure both implementations will keep being equivalent.

It should be noted that the final Python implementation has comparable performance to the old Fortran implementation. This serves as proof that it is a common misconception that Python has always worse performance than other compiled programming languages such as Fortran. The coupler provided by pyOASIS, which is just a Python interface to the Fortran OASIS implementation, adds imperceptible overhead as described in Performance differences between pyOASIS and OASIS section, the math handling modules like *numpy* are mostly made up of ultra-optimized C code and Python's file management system works flawlessly with the most up-to-date techniques.

As for the insecurity provided by the dynamic typing, since Python 3.5 it is possible to add *type hints*[6] to the code, which can be checked by any linting software and work as type checking. The current AMIP reader Python implementation does not provide any typing because of backwards compatibility, but it could be added later with no modification to the code logic.

In conclusion, thanks to its flexibility and comparable performance, Python has proven to be a good substitute for Fortran in small components such as the AMIP reader. Even compiled Python code could be used in larger and more complex components in the EC Earth ecosystem.

References

[1] OASIS - CERFACS. Available: https://portal.enes.org/oasis

[2] CIMP6 - EC-Earth. Available: http://www.ec-earth.org/cmip6/

[3] OASIS3 - MCT GitLab repository. Available: https://gitlab.com/cerfacs/oasis3-mct

[4] ES GitLab - Fortran to Python OASIS conversion. Available: https://earth.bsc.es/gitlab/rmarti1/oasis-toy-model-python

[5] ES GitLab - Python AMIP reader. Available: https://earth.bsc.es/gitlab/rmarti1/oasis-toy-model-python

[6] Support for type hints - Python. Available: https://docs.python.org/3/library/typing.html