DRAFT REPORT

Requirements and Implementation Plan for a Revised ESMValTool Backend

Mattia Righi (DLR, Germany), **Veronika Eyring** (DLR, Germany), **Björn Brötz** (DLR, Germany), **Axel Lauer** (DLR, Germany), **Niels Drost** (Netherlands eScience Center), **Paul Earnshaw** (MetOffice, UK), **Carsten Ehbrecht** (DKRZ, Germany), **Martin Evaldsson** (SMHI, Sweden), **David Hassell** (University of Reading, UK), **Stephan Kindermann** (DKRZ, Germany), **Mateusz Kuzak** (Netherlands eScience Center), **Bill Little** (MetOffice, UK), **Alexander Loew** (LMU, Germany), **Ben Müller** (LMU, Germany), **Ag Stephens** (STFC CEDA, UK), **Willem van Hage** (Netherlands eScience Center), **Javier Vegas Regidor** (BSC, Spain), **Thomas Voigt** (MetOffice, UK), **Jost von Hardenberg** (ISAC, Italy), and **Keith Williams** (MetOffice, UK)

First Draft: 9 December 2016

Finalization Date: 31 January 2017

Before the workshop, we encourage those attending to read through the iris user guide: http://scitools.org.uk/iris/docs/latest/userguide/index.html and, if possible, download iris (http://scitools.org.uk/iris/download.html) and follow through some of the examples.

Preface

At the ‘2nd Technical ESMValTool Workshop’ that was held 15-16 November 2016 in Munich, a decision was made that we will actively work towards a **merge of ESMValTool and Auto-Assess based on Iris**. It was agreed that for the rewriting of the ESMValTool backend as preparatory work for the merge of the ESMValTool with AutoAssess a coding workshop is needed.

It was decided that this 5-day coding workshop for rewriting the ESMValTool backend will be held at the MetOffice on 6-10 February 2017 in Exeter, UK. Participants will include scientists from the Iris and AutoAssess development teams of the MetOffice and ESMValTool developers from DLR, University of Reading, BSC, Esciencecenter, SMHI, and possibly from DKRZ, ISAC and LMU.

The goal of this report is to set the basis for this coding workshop. We aim at finalizing this report by end of January. The writing will be led by DLR and other partners who are involved in the rewrite of the backend (see above) will contribute.

The report includes (a) requirements for the revised ESMValTool backend; (b) current issues; (c) an implementation plan that should give clear guidance for the coding workshop early February 2017.

1. Introduction and motivation

The ESMValTool applies diagnostics and metrics scripts to a wide range of input data, including models from CMIP5 and other model intercomparison projects (MIPs), selected models in their native format (EMAC, GFDL, EC-Earth and, in the future, the MetOffice model), and observations.

Before the analysis and diagnostic codes can be applied, a number of common operations such as extraction, reformatting, regridding, masking, etc., need to be applied to the input data. These operations are collectively named **pre-processing** and are (or should be) performed by the backend.

The current version of the ESMValTool performs only a few of these operations in a centralized way in the backend, while others are performed at a lower level, for example within specific diagnostics. This results in several drawbacks, such as slow performance, code duplication, lack of consistency among the approaches, and limited documentation.

The goal of this document is to outline the ideas for a **revised ESMValTool backend** that should aim at overcoming some of the known issues regarding efficiency. We will start describing the current status and issues (Sec. 2), then outline the desired features (Sec. 3) and an implementation plan including a test case to be followed as a guideline for the development of the new software (Sec. 4).

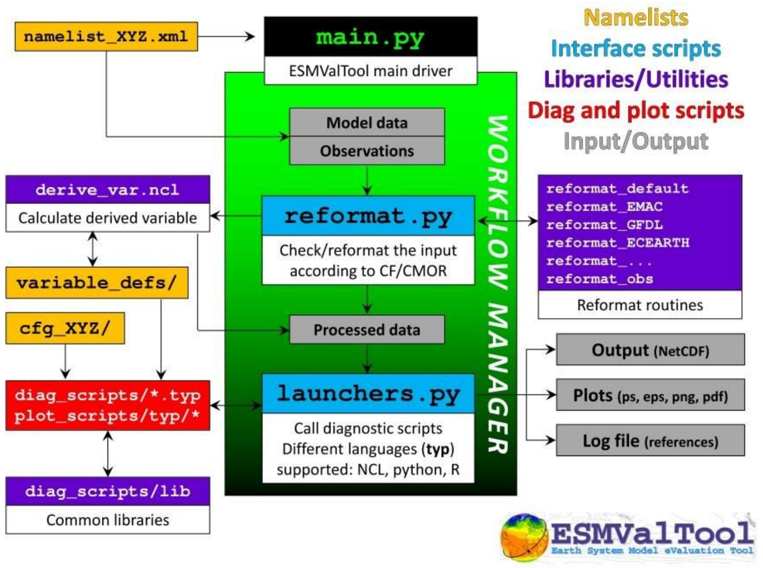

Figure 1. Current structure of the ESMValTool. From Eyring et al. (2016).

2. Currently implemented operations, backend status and related issues

The various operations required to prepare the input data are listed in Table 1. Currently, these operations are performed at **two different levels** within the tool (Figure 1):

- at the **backend level**, controlled by the `reformat.py` script in the workflow manager and mostly based on the `reformat_*.ncl` and `variable_defs/*.ncl` functions.
- at the **diagnostics level**, distributed in the various diagnostics script in `diag_scripts/` and partly centralized in the libraries in `diag_scripts/lib/`.

One of the primary goals of the revised ESMValTool backend is to move the code for these operations out of the diagnostic scripts and to **centralize** it in the backend. Having these operations centralized at a higher level will facilitate **maintenance** and **documentation**, and improve the code **readability** and customization **options** for the user. Another issue in the current implementation of the backed is that all the operations are performed in a **serial** way, resulting in a high computational **burden**, both in terms of CPU-time and memory load. Centralizing all the operations in the backend shall also facilitate code **optimization** and **parallelization**.

Operation	**Current level**	**Computational burden**	**Priority**	**Iris method**
Time extraction	Backend	Low	1	cube.extract
Check of CMOR data	Backend	Low	1	*Apply tests to cube*
CMORization of non-CMOR model data	Backend	Medium	2	*callback to correct metadata*
CMORization of observational data	External routines	Low	2	
Variable derivation	Backend + external routines	Low-Medium	1	*cube maths*
Level selection	Diagnostic	Low	2	cube.extract
Region selection	Diagnostic	Low	2	cube.intersection
Regridding and interpolation (horizontal, vertical)	Diagnostic	High	1	cube.regrid (horizontal) stratify (vertical - near completed development)
Masking	Diagnostic	Medium	1	*Cubes using masked arrays*
Basic time operations (mean, seasons, diurnal cycles, etc.)	External routines + diagnostics	Low-Medium	2	cube.collapsed cube.extract
Basic grid point operations (zonal means, cross sections, etc.)	Diagnostics	Low-Medium	2	cube.collapsed cube.extract
Multi-model statistics	Diagnostic	Medium-High	1	*cube maths* *merge cubes*

Table 1. Currently implemented pre-processing operations and their corresponding implementation level, computational burden and priority level for the implementation in the revised ESMValTool backend.

3. Requirements for the revised ESMValTool backend

Based on the current experience with the ESMValTool, the following features are considered essential requirements for the revised ESMValTool backend:

- **Independent package**: the revised ESMValTool backend shall be seen as an **independent pre-processor** which is decoupled from the rest of the ESMValTool (consisting of workflow manager, diagnostics and graphics routines). This means that it runs independently of the diagnostics and is solely controlled by the main namelist settings. The backend should read-in the original input data (as specified in the main xml namelist via the different project classes defined in `project.py`), apply the required operations and write the output to a netCDF file in the `climo/` directory, which shall be renamed `preproc_dir/` for consistency (and also specified in the main namelist as `preproc_dir`). The output file(s) from the backend will be the starting point for all diagnostics. Note that writing the pre-processed data to a file is required in order to ensure the **multi-language support** for the diagnostic scripts which use the pre-processed data for the analysis. Also note that in the ESMValTool, the pre-processing is performed on a per-variable and per-model basis, meaning that one netCDF file is generated for each variable and for each model.. This follows from the CMIP5 CMOR DRS (and other MIP projects) which store data in one file per variable. The revised ESMValTool backend should therefore act as an **independent service**. We shall also consider the possibility to accept requests in different formats and most crucially **via other API interfaces**, in addition to an xml configuration file. This way it could also be used as a separate service by any software which has not (yet) been integrated in ESMValTool, or which cannot by its nature be integrated (e.g. in CS3-MAGIC WP7 a downscaling module for precipitation will be implemented which will need to request precip fields over given areas as an input).

- **Regridding:** the regridding should be done as part of the revised ESMValTool backend and not as part of the diagnostics as is the case now. The backend needs to communicate with the ESMValTool namelists and diagnostics which expect these files to be available to them. Still the diagnostics may require similar services again (such as regridding of a derived variable) so it should be possible for the diagnostic modules to call back the backend and request it to do this service. [This needs to be discussed with a practical example and discussion is required how this could best be technically handled].

- **User controllable**: the various operations performed by the backend and the related options shall be fully controllable by the user via the ESMValTool xml namelists. In addition to the existing entries (models, diagnostics, etc.), the user shall also be able to specify, for example, the regridding method and the target grid to be used, whether a mask is to be applied and which one, whether a specific level or region is to be extracted, whether a specific time period is to be extracted, and so on. For some diagnostics, the user may also need to set the ordering of the above operations: if we continue using the SAX-parsing of the main namelist as it is done now, there is an implicit ordering in the sequence of the operations, consistent with the user intuition when reading the xml-file (left-to-right, top-to-bottom).All this information shall also be written to the log file for documentation.

- **Fast and efficient**: one of the biggest problems with the current version of the tool is the large computational time required for running even a single diagnostics (hours to days for the most demanding ones, such as `perfmetrics_main.ncl`). Parallelization is therefore a must. An easy way would be to split the pre-processing operations in a way that the processing of each model can be assigned to a different CPU. In addition, it must be possible to run several ESMValTool namelists and/or several individual diagnostics of a namelists in parallel. Already calculated files should be used as much as possible, to avoid repetition of already-performed expensive operations. Given the high volume input data and the HPC environment often available for the ESMValTool users the operations of the revised ESMValTool backend have to allow for multi-node parallelization. Connected to this requirement the ability to perform distributed analysis on separated machines should be realized. Storage can also be a limiting factor, as it is easy to fill a local quota on a shared resource.

- **Modularity**: each of the operations listed in Table 1 shall be performed by a dedicated block of code (e.g., a python class). The possibility of performing each operation as an independent job shall also be explored. This would facilitate testing, maintenance and future developments. Particularly important is the block dealing with on-the-fly CMORization-light of models and observations from their native (non-CMOR) formats, since this will continuously grow with more models and more datasets being added in the future.

Such modularity will also facilitate a flexible switching the order of operations as specified by the user in the main namelist.

- **Single language and existing libraries**: given the above requirements for efficiency, flexibility and modularity, Python was identified as the best suited language for programming the revised backend. Multi-language support in the backend is not needed, since the backend would be mostly maintained by the core development team and a small number of other ESMValTool developers who are focusing on technical improvements rather than on including more scientific diagnostics. Using python will also allow taking advantage of the numerous existing packages available for the required standard operations (IRIS, numpy, cdo, etc.). An important issue to consider is whether to use python 2 or python 3. Ideally, we should write as much code as possible compatible with 2.x and 3.x, which will facilitate future port (Python 2.x will not be maintained past 2020).
- **Well documented**: the revised ESMValTool backend shall not pre-process data as a black box. All operations and related namelist settings shall be well documented in the user's guide and appear also in the log file for enhanced transparency and reproducibility. Some information could also be appended to the global history attribute in the NetCDF file produced by the backend.
- **Functionality:** the current versions of the ESMValTool and AutoAssess include several key functionalities that the revised ESMValTool backend needs to keep. For example, the ESMValTool runs – with the help of reformatting routines – on CMIP output but additionally also on selected models in their native format (e.g., EMAC, GFDL, EC-Earth), and AutoAssess runs on the MetOffice model output but not yet on CMIP output. The revised backend should offer the same functionality as the existing ESMValTool and AutoAssess backends plus additional abilities, preferably in a way that diagnostics could remain as much as possible unchanged.
- **Computing Environment:** The revised ESMValTool backend shall run locally and stand-alone on a Linux computer (following the same system requirements of the tool itself, as given in the official User Guide) but it shall also be possible to efficiently coupe it to the Earth System Grid Federation (ESGF) and the Copernicus Climate Change Service (C3S)-Server. The Birdhouse Web Processing System (WPS) will be implemented as a compute solution at DKRZ, CEDA, and IPSL.
- **License.** The revised ESMValTool backend will be developed in a git repository at https://github.com/ESMValTool-Core/ESMValTool as open source community software under an Apache version 2.0 license. No copy-left code pieces should be included.
- **Multi-project support:** The ESMValTool development is funded by various projects, including (a) the EU Horizon 2020 Coordinated Research in Earth Systems and Climate: Experiments, kNowledge, Dissemination and Outreach (CRESCENDO) project, (b) the EU Horizon 2020 PRocess-based climate sIMulation: AdVances in high-resolution modelling and European climate Risk Assessment (PRIMAVERA) project, (c) the Copernicus Climate Change Service (C3S) Lot 1 and Lot 2 Metrics and Access to Global Indices for Climate Projections (C3S-MAGIC) projects, (d) the BMBF CMIP6-DICAD project that funds CMIP6 activities in Germany, and (e) the ESA Climate Change Initiative Climate Model User Group (ESA CCI CMUG) project. Depending on the project, technical and/or scientific development (i.e. development and inclusion of new diagnostics and performance metrics) is funded. The revised ESMValTool backend needs to serve the various projects. In terms of efficiency, the highest demands stem from the evaluation of **high-resolution** models as part of PRIMAVERA, from a **quasi-operational analysis** of CMIP models alongside the ESGF that is aimed for in C3S-MAGIC, CRESCENDO and CMIP6-DICAD.
- **Testing:** The revised backend needs to provide efficient automated testing.

- **Deliver user-friendly products from the CMIP ensemble:** the revised ESMValTool backend should be able to deliver user-friendly products such as annual, regional, seasonal and zonal means, both for individual models but also the multi-model mean (see also Table 1).
- **User interface:** the revised backend should allow to be run within a user-friendly application.

4. Implementation plan

4.1 Use cases (to be done by end of January, lead DLR)

To facilitate the implementation and, most importantly, the testing of the revised ESMValTool backend we shall first define use cases, based on the current version of the tool.

The most significant use case would be reproducing the pre-processing steps of the `perfmetrics_main.ncl` routine, since it covers all the required operations listed in Sec. 2, with the exception of the multi-model/ensemble statistics, which is however a very basic operation which can also be tested separately. We can choose several variables covering the most common cases, for example:

- a 3D variable with level selection: temperature at 850 hPa (ta-850)
- a 2D derived variable: total column ozone (toz) and short-wave cloud forcing (SW_CRE)
- a variable on a non-regular grid: sea-ice (sic)

This should also be done on the highres models to identify where the show stoppers are. A CMIP5 model output should be regridded to very high resolution and a systematic performance test how the memory goes up and the performance goes down should be done and documented.

Other use cases might be added to fully test the desired functionality of the revised backend.

4.2 Structure of the revised ESMValTool Backend

A possible implementation plan for the single node version of the revised ESMValTool backend is detailed in Figure 2. Each of the yellow boxes represents a coding block dealing with one of the operations given in Table 1. The orange cards are the ESMValTool xml-namelist settings specified by the user (some possible settings are given as an example). Some blocks require the use of external routines (represented as white rectangles), e.g. model-specific fix files to fix common format error in CMIP models (possibly also refining the current approach which is only based on project/model/variable/ensemble selection), routines to CMORize models and observations, and recipes to calculate derived variable. The order of operations given in Fig. 2 corresponds to the one currently implemented in perfmetrics_main.ncl. A concept is yet to be defined how to organize the order of sequence for operations that could/should be performed independently (e.g. regridding, masking, calculation of mean, standard deviation, etc), also allowing for a user-specified ordering.

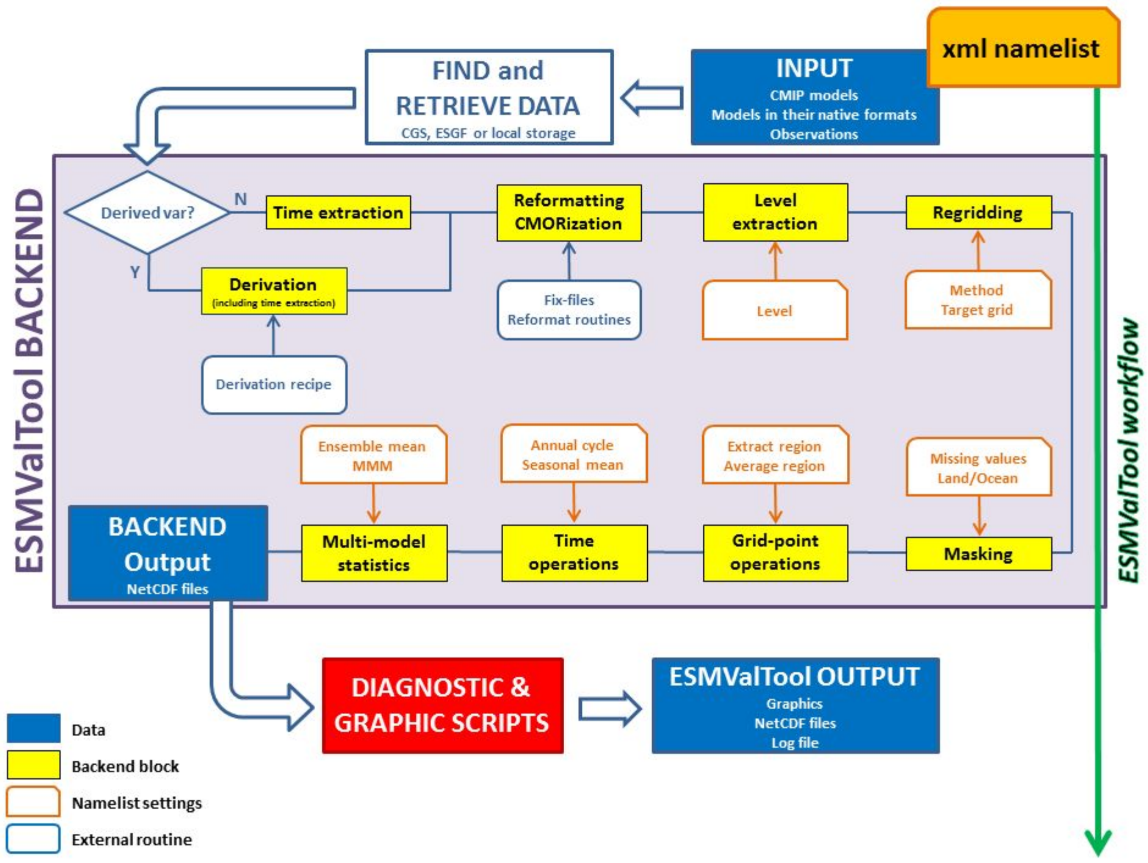

Figure 2. Schematic view of the revised ESMValTool backend.

A central workflow manager has to take care of the distribution of the tasks according to the input data and to join the final or intermediate results. A solution must be found for jobs that could potentially share data by passing objects, as this is something that can not be done using the standard schedulers used by the clusters (job wrapping?).

4.3 Structure of coding workshop

We shall organize the coding workshop in different groups (e.g. 1-3 people each group, depending on the complexity of the task) dealing with the following and writing the corresponding code:

- find and retrieve data: this shall also allow of a quick dry-run, where **all** the required input files (model and observations) are check for existence before starting the actual pre-processing. Without such feature a namelist can execute for quite some time before crashing due to a trivial missing-file error.
- variable reading (including time selection and derivation recipes): currently diagnostic scripts rely on a temporary file “<varname>_info.tmp” to be available in the `interface_data/` folder. This file is generated on the fly and contains some information, basically the variable_info-attributes, from the corresponding “variable_defs/<varname>.ncl” file. If the variable defs files are moved from NCL to Python the handling of the “<varname>_info.tmp”-files needs to be reviewed as well.

- CMORization (focusing on CMIP5 models, models in their native format and observations): we will select a few models and observations as use cases for the workshop (see also Sec. 4.1). The primary goal is to implement the functionalities currently covered by reformat_default (i.e., CMORization light), also including the re-writing of the fix-files in python possibly with a more sophisticated method to identify the dataset versions (currently based on project/model/ensemble/variable name). We need also to start implementing a few routines for reformatting the observations, as a working example. For the (more demanding) task of model-specific CMORization, this should be a responsibility of the corresponding groups. If time allows, we could start implementing this functionality for EMAC and use it as a working example for GFDL and EC-Earth groups which will eventually need to implement their routines for CMORizing their models.
- level selection and grid-point operations: this task should be relatively easy and could be managed by the same group focusing on multi-model and ensemble statistics (see below).
- regridding (including regular and irregular grids): if possible, we should try to keep regridding along the lines of existing standards, e.g., to consider including SCRIP-functionality and handling of SCRIP grid description files, to follow (or rely on) cdo regridding functionalities, to name regridded or intermediate files along the lines of CMIP DRS when it makes sense, etc. Regridding (especially for ocean grids) often requires using auxiliary files (the so-called fx-files in CMIP), which shall be made explicit in the xml namelist instead of implicitly defined in projects.py, as it is the case now. Another possibility for regridding ocean grids, is to make the tool aware of common ocean grids used in European climate models (e.g., OCRA grids, MPI, etc.): when a grid is recognized the tool will load the appropriate grid description file, weight file, etc. These auxiliary files, although binary, could be added to the repository to exemplify for users when/if they need to add similar files for their own models.
- masking (including land-sea masking and missing value filtering): auxiliary file shall be used here as well (when available), as also suggested for the regridding above. Built-in land-sea masks could also be applied but making it explicit to the users such that they are aware of such manipulation as there might be issues due to different resolutions
- multi-model and ensemble statistics: this task should be relatively easy and could be managed by the same group focusing on level selection and grid-point operations (see above).

Each block (yellow box) should be programmed in a modular way that easily allows communication with the other blocks and if possible parallel execution, and flexible definition of order of operations. Furthermore we would need:

- 1 group working on the namelist extension. This group shall get continuous feedback from the above groups on the user settings needed from their block. Several desired functionalities for the main namelist have been discussed in the past, including: definition of a XML schema for the namelist to allow check at runtime, possibility to specify the reference model in the main namelist (as for example in the perfmetrics namelist), inclusion of placeholders (for example to read two variables from different MIPs).
- 1 group dealing with the testing of the output as long as provided by the various groups. A quick testing method could be to compare the NetCDF output produced by the current version of the tool and by the backend. NetCDF files for the test cases mentioned above could also be prepared in advance.

4.4 Further details of the implementation plan

4.4.1 Further specify parallelization (NLESC, SMHI)

ESMValTool is a system comprised of a few distinct subsystems where parallelization could make sense. As parallelization is not an end by itself (rather a means to increase the speed of ESMValTool), it will take some further investigation, and probably a combination of these approached to reach our goal.

- It is especially important to let any optimizations be guided by real performance number rather than guess work, and take care not to make the implementation overly complex. For this reason we will start by setting up profiling for ESMValtool.
- As already noted by Carsten Ehbrecht, ESMValtool is currently writing temp files into a fixed folder, which is normally located within the source code. This is making it hard to run (pieces of) ESMValTool more than once, and also hinders parallelization on a higher level (e.g. if ESMValTool is run as a WPS service). We should make sure all intermediate files are written in unique, configurable, temp directories and files.
- Some of the tools/libraries/languages (for instance ncl) used in ESMValTool have (experimental) multicore support. We should make sure these are enabled (some are off by default, depend on compile settings, etc), to see if this makes a difference in our case.
- The main.py driver script contains a nested loop which is a prime candidate for task-based parallelization. (e.g. calculate each diagnostic on a different core of a multicore machine, or different node of a cluster). Creating a parallel version of main.py using e.g. Multiprocessing should be possible. This does assume most iterations of each loop are independent.
- Further expanding on the option above, since a lot of separate scripts are called from main.py, we could also use a workflow execution engine to run all these scripts. This execution engine will then take care of transferring all files, running tasks in the right order, running tasks in parallel if possible, etc. One prime candidate is to use the Common Workflow Language (http://www.commonwl.org/), as we are then able to switch execution engines without changing any code.
- We will be using Iris in the revised backend to do all the "heavy lifting" of regridding, etc. This is an excellent place for data-parallel execution. Iris uses numpy/scipy. Depending on exactly what is done, code can sometimes be sped up tremendously by using for instance cython, optimized implementations of underlying libraries, and/or OpenMP. A quick look at the iris regridding code shows there should be room for improvement as iris seems to currently use pure Python and numpy. (Please verify)
- Another option for speeding up ESMValTool would be to re-use intermediate files created by the backend, if multiple diagnostics use the same input. We should take care to not make this overly complex. Perhaps the simplest approach would be to add some pre-calculated statistics to all input data, for instance a monthly and yearly mean for fields where this makes sense.

4.4.2 Testing (LMU, NLESC, MetO)

Testing of the backend

The development of the backend should be based on a test-driven approach. Each component and routine of the backend should come with a corresponding testing routine. Best practice would be even to develop a test case first before actually implementing the code for the backend functionality (this is called Test Driven Development).

The python **unittest** module should be used for test implementation or a similar approach like chosen by the IRIS development team would be useful instead. Following the coding standards for python projects, the tests are expected in a subdirectory “test”. Filenames containing test routines shall start with “test_” in their names.

The following testing guidelines are currently used for the IRIS development:
http://scitools.org.uk/iris/docs/latest/developers_guide/tests.html

Some further information for those not having done testing in python yet:
https://docs.python.org/2/library/unittest.html

http://docs.python-guide.org/en/latest/writing/tests/

Testing of diagnostics

For code development, a unit testing scheme instead of an integration testing scheme will be implemented for selected diagnostics in the beginning. This is already existing now (22.1.2017) for selected diagnostics, but still needs to be merged with the trunk. The current testing of diagnostics is based on synthetic data. The major advantage is that this allows users also to test the correctness of a new ESMValTool installation without the need to download a large volume of reference data. The developed concept is applicable to diagnostics written in any of the supported languages.
In addition, unittests for individual diagnostics can be further developed.

4.4.3 ESGF coupling requirements

- The ESMValTool ESGF coupling module can be configured to use local ESGF data archives on the filesystem. The configuration can be adapted for each data-provider (root-folder, versioning).
- The synda ESGF data replication tool is used to maintain a local ESGF data archive. DLR is working on triggering ESMValTool diagnostic runs when new data has been replicated by synda according to a specified data selection. This is a feature required by CMIP6.
- In Copernicus a subset of ESGF data (plus observation data) is provided. This data is also replicated using synda and we can probably assume that all data for a diagnostic will be available locally. Thus the existing ESGF coupling module will be sufficient for Copernicus as well.
- The ESGF coupling module is on a branch in subversion and should be integrated in the main development trunk (or master branch on Git).
- The ESGF coupling module has its own configuration. Maybe this can be integrated in the common ESMValTool configuration.

Running ESMValTool as a Web Processing Service:

- ESMValTool installation folder needs to be **read-only**.
- Currently the ESMValTool uses files in the folder `interface_data/` for communication between scripts (probably Python/ncl?). This **will not work** when the same installation is used by several WPS processes.
- ESMValTool needs working directories for the processed files and results. The paths to the working directories is set in the `namelist.xml`. In the current WPS processes for ESMValTool a `namelist.xml` is generated with a unique working directories for each process execution. The

working directory paths are relative to a temporary folder provided by the WPS (See Example 1). This temporary folder is removed after the process execution and the process results are stored in the WPS output storage identified by a unique ID. This approach is already working but it relies on generating a `namelist.xml` for each process execution. One could simplify this by providing just a single working directory which is used as the root folder for all other working paths used by ESMValTool. One could provide the working directory as parameter on the call of the ESMValTool (command line option, python api parameter), so the `namelist.xml` does not need to configure the working directories. By this the `namelist.xml` does not need to be generated for each process run.

- Currently it is not clear what the output files of a diagnostics are and where they are written to. For each diagnostic one needs to check what it produces and one needs some kind of guessing (pattern matching) to find the outputs. It would be nice if the diagnostics would specify their outputs.

```
<wrk_dir type="path">/tmp/pywps_process_oLDQen/work/</wrk_dir>
<plot_dir type="path">/tmp/pywps_process_oLDQen/work/plots/</plot_dir>
<climo_dir type="path">/tmp/pywps_process_oLDQen/work/climo/</climo_dir>
```

Example 1. Working directories in a generated namelist.xml used by a WPS process.

4.5 List of participants for coding workshop (6-10 February 2017, Exeter, UK)

1. Mattia Righi (DLR, Germany) - Chair
2. Axel Lauer (DLR, Germany)
3. Javier Vegas-Regidor (BSC, Spain - PRIMAVERA)
4. Mateusz Kuzak (eScience Center, Netherlands – C3S-MAGIC)
5. Niels Drost (eScience Center, Netherland – C3S-MAGIC)
6. Ag Stephens (STFC CEDA – C3S-MAGIC)
7. Carsten Ehbrecht (DKRZ, Germany – CMIP6-DICAD)
8. Stephan Kindermann (DKRZ, Germany – CMIP6-DICAD)
9. Keith Williams (MetOffice, UK)
10. Bill Little (MetOffice, UK)
11. Thomas Voigt (MetOffice, UK)
12. Paul Earnshaw (MetOffice, UK)
13. Laura Dreyer (MetOffice, UK)

5. References

Eyring, V., Righi, M., Lauer, A., Evaldsson, M., Wenzel, S., Jones, C., Anav, A., Andrews, O., Cionni, I., Davin, E.L., Deser, C., Ehbrecht, C., Friedlingstein, P., Gleckler, P., Gottschaldt, K.D., Hagemann, S., Juckes, M., Kindermann, S., Krasting, J., Kunert, D., Levine, R., Loew, A., Mäkelä, J., Martin, G., Mason, E., Phillips, A.S., Read, S., Rio, C., Roehrig, R., Senftleben, D., Sterl, A., van Ulft, L.H., Walton, J., Wang, S. and

Williams, K.D., 2016. ESMValTool (v1.0) – a community diagnostic and performance metrics tool for routine evaluation of Earth system models in CMIP. Geosci. Model Dev., 9(5): 1747-1802.

A telecon will be held on 17 January 15:00-16:30 CET (Central European Time) UTC/GMT +1 hour.

Agenda for Telecon:

(1) General remarks (Veronika)
(2) Brief summary of the status of the backend report (Mattia)
(3) Identification of gaps (see also Section 4.4 Further details of the implementation plan)
(4) Status Iris / AutoAssess coupling
(5) Define Action Items until the Coding Workshop
(6) We shall organize the coding workshop in different groups (e.g. 1-3 people each group, depending on the complexity of the task), see Section 4.3. Can we already assign people for the different groups before the workshop?
(7) Discussion on critical open issues
(8) Organisational aspects of the coding workshop (Keith)

Start 9 am on the first day; please bring your passport
9 am to 6 pm and maybe a bit shorter on Friday.
One room booked; but people can be in the central part of the building to find a corner for breakout groups.