Update of ESiWACE Application Software Framework prepared for the demonstrators

Deliverable D3.2

This project has received funding from the European Union's
Horizon 2020 Research and Innovation Programme
under Grant Agreement No 675191

About this document

Work package in charge: WP3 Usability
Actual delivery date for this deliverable: 5 March 2018
Dissemination level: PU (for public use)

Lead author:
Max Planck Institute for Meteorology (MPI-M), Sergey Kosukhin, Reinhard Budich
Other contributing partners:
Barcelona Supercomputing Center (BSC), Kim Serradell, Oriol Mula-Valls
The University of Reading (UREAD), Grenville Lister

Rewier:
German Climate Computing Center (DKRZ), Chiara Bearzotti

Contacts: esiwace@dkrz.de
Visit us on: www.esiwace.eu
Follow us on Twitter: @esiwace

Index

1. Abstract

Software implementations of modern climate and numerical weather prediction models and their execution workflows require many pieces of software to be correctly installed and configured. Various mathematical, input/output, parallelisation and other software libraries are necessary to build and run executables of the model; plotting, statistics, file format conversion and other packages are required for post-processing; scripting language interpreters and filesystem utilities are essential for controlling different stages of the execution workflows and gluing them together. All these generate long lists of software dependencies that have to be resolved and accounted for during the deployment of the complete software stack required for numerical experimentation. This process is further complicated by the fact that the models are research tools and hence are under continuous development. It is often the case that introduction of changes in the model's source code becomes an essential part of a research workflow. Thus, the software environment for the research workflow does not have to only fulfil the runtime requirements of the model execution workflow but also the requirements posed by software development needs. On top of that, given the resource requirements of some scientific projects, a single machine is not always enough to accommodate all required model runs. Thus, it becomes mandatory to make the software stack portable to whatever supercomputing environment becomes available for the numerical experiments.

This work aims to systematise requirements of a typical climate modelling workflow to the software environment, to identify common problems on the way to meeting them, and attempts to remedy the situation by providing recommendations on what solutions can be applied to make the preparations of the software environment for Earth system modelling easier.

There are no indications that the requirements to the software environments of the upcoming extreme scale HPC will decrease. On the contrary, the main challenges related to the increasing amount of input and output data of the models, workload balancing, and fault tolerance (see ESiWACE deliverable D3.8[1]), will be exacerbated, requiring more and more sophisticated solutions, the implementation of which will inevitably lead to the further complications of the already difficult problem of the software dependencies resolution. At the same time, most experts assume that the basic concepts underlying the existing approaches to the software deployment and maintenance will be inherited in the next generation of HPC solutions. Thus, the investment into the development of the automatization tools for software installations promises long term benefits for the whole HPC community.

2. Conclusion and Results

The main result of the deliverable is the publication of the s**econd version of the White Paper**: "Application Software Framework: A White Paper". The white paper is available on the ESiWACE website https://www.esiwace.eu/results/misc/the-application-software-framework/view
Please note that we do not consider this version as the final one and plan to develop it further until the end of the project.

[1] https://www.esiwace.eu/results/deliverables

3. Project objectives

This deliverable contributes directly and indirectly to the achievement of all the macro-objectives and specific goals indicated in section 1.1 of the Description of the Action:

Macro-objectives	**Contribution of this deliverable?**
Improve the efficiency and productivity of numerical weather and climate simulation on high-performance computing platforms	No
Support the end-to-end workflow of global Earth system modelling for weather and climate simulation in high performance computing environments	Yes
The European weather and climate science community will drive the governance structure that defines the services to be provided by ESiWACE	Yes
Foster the interaction between industry and the weather and climate community on the exploitation of high-end computing systems, application codes and services.	Yes
Increase competitiveness and growth of the European HPC industry	No

Specific goals in the workplan	**Contribution of this deliverable?**
Provide **services** to the user community that will impact beyond the lifetime of the project.	Yes
Improve **scalability** and shorten the time-to-solution for climate and operational weather forecasts at increased resolution and complexity to be run on future extreme-scale HPC systems.	Yes
Foster **usability** of the available tools, software, computing and data handling infrastructures.	Yes
Pursue **exploitability** of climate and weather model results.	No
Establish governance of common software management to avoid unnecessary and redundant development and to deliver the best available solutions to the user community.	Yes
Provide **open access** to research results and **open source** software at international level.	Yes
Exploit **synergies** with other relevant activities and projects and also with the global weather and climate community	Yes

4. Detailed report on the deliverable

The objective of this deliverable was to update and to refine the first version of the White Paper delivered at the beginning of the project. Thus, our research was focused on further analysis and systematisation of the requirements of Earth system modelling workflows, as well as identification of common issues and limitations related to the deployment and maintenance of software stacks on large supercomputing facilities. The main conclusion that we made based on the results of the second phase of the research was that users, in order to be productive, have to share the responsibility for completeness and usability of the software environment. To achieve that, they have to get a better understanding of the main issues related to the software deployment and to learn about the existing approaches to its automatization. Thus, providing this information has also become one of the objectives of the Whitepaper.

The updated version of the Whitepaper also takes into account the change in the project priorities towards high-resolution demonstrators. At the example of the models that were selected as demonstrators, we have illuminated the relevance of the issues highlighted in the document, as well as presented the applicability of the provided approaches to their solution.

5. References *(Bibliography)*

References can be found in the White Paper.

6. Dissemination and uptake

6.1 Dissemination

The White Paper is in Open Access and available here: https://www.esiwace.eu/results/misc/the-application-software-framework/view

6.2 Uptake by the targeted audience

As indicated in the Description of the Action, the audience for this deliverable is:

x	The general public (PU)
	The project partners, including the Commission services (PP)
	A group specified by the consortium, including the Commission services (RE)
	This reports is confidential, only for members of the consortium, including the Commission services (CO)

This is how we are going to ensure the uptake of the deliverables by the targeted audience
The White Paper going to be made available in Zenodo for granting full access to communities beyond the ESiWACE community. Information about revision and updates of the document is disseminated through the regular channels: ESiWACE newsletter and mailing lists, project webpage, workshops and citations.

7. The delivery is delayed: ☐ Yes [✓ No]

8. Changes made and/or difficulties encountered, if any

None.

9. Sustainability

9.1. Lessons learnt: both positive and negative that can be drawn from the experiences of the work to date

Supercomputer systems that are used for ambitious Earth system modelling show significant differences in purpose and size, resulting in substantial qualitative differences in their design, architecture and interfaces. While most of their complexity is hidden from the users by their *clustering middleware* – the software that allows treating an HPC as one large computing unit –, users still have to be aware of some special aspects of distributed computing environments to be able to employ them effectively. To achieve that, they have to get a better understanding of the main issues related to the software deployment and to learn about the existing approaches to its automatization. Users educated this way than have to interact with system administrators. Supporting Spack installations will benefit both groups in the following way:

- Users will be able to easily customise the software environment on their own, thus being more productive and reducing the workload put on system administrators.
- Spack ensures the completeness of descriptions of software dependencies and requirements, thus mitigating possible communication problems between software developers, users, and system administrators.
- Spack allows for formal description of the software stack available on a supercomputer, which simplifies identification of the list of missing software dependencies of a modelling workflow.
- Spack helps system administrators to easily test various usage scenarios of the basic elements of the software environments, such as compiler toolchains and MPI libraries.

All these aspects are especially important in solving the problem of providing a complete, well tested and reproducible software environment for the demonstrators.

9.2 Links built with other deliverables, WPs, and synergies created with other projects

This deliverable reports on the update of the Whitepaper delivered as D3.1 in the WP3 "Usability". The update is mainly driven by the experience gained in the course of testing the deployment procedures (D3.5) of the software stack required for the demonstrators (D3.7) on different supercomputers (D3.8). This task implied close collaboration with the WP2 "Scalability" team, who provided us with a real use case of the results of our research activity on one hand, and received our support in solving issues related to the deployment and configuration of the software stack required for the demonstrators on the other hand.

Ideas and methods set forth in the Whitepaper were well received by the ICON Community, who decided to put the development and support of the automatized software deployment tools on the roadmap of the infrastructure development of ICON model, one of the demonstrators.

The Earth Science department at Barcelona Supercomputing Center is also very much satisfied with the described solutions and plans to apply them for the deployment of the models they work with (EC-Earth, also one of the demonstrators, and MONARCH), especially on "external" HPC facilities, which usually do not provide the complete software stack required for their experiment workflows.

10. Dissemination activities

Type of dissemination and communication activities	Number	Details	Total funding amount	Type of audience reached In the context of all dissemination & communication activities	Estimated number of persons reached
Participation to a workshop	1	Presentation of S. Kosukhin, Software stack deployment for ESM, ESiWACE and IS-ENES2 Joint final workshop on IS-ENES2 Workflow Solutions in Earth System Modelling and on Meta-Data Generation during Experiments, Costa da Caparica, 27-29 September 2016	See costs declared in form C of MPI-M	Scientific community, Higher education, Industry,	30
Participation to a conference	1	Presentation of K. Serradell (BSC) S. Kosukhin (MPI-M), Software stack deployment for Earth System Modelling using Spac k, PRACE Days 2017 Barcelona, 15-18 May 2017	See costs declared in form C of MPI-M and BSC	Scientific community, Higher education, Industry	20
Publication of a report	1	Publication of the “Application Software Framework: A White Paper” on the website of the project https://www.esiwace.eu/results/misc/the-application-software-framework/view	See costs declared in form C of partners involved.	Scientific community, Higher education, Industry	
Participation to a workshop	1	Presentation of L. Kornblueh, S. Kosukhin, Spack for ICON; ICON Developers Meeting, 28.09.2017, Löwenstein	See costs declared in form C of MPI-M	Scientific community, Higher education, Industry	38
Participation to a workshop	1	Presentation of S. Kosukhin, Spack for ICON: Status Update, ICON Infrastructure Meeting, 09.02.2018, Karlsruhe	See costs declared in form C of MPI-M	Scientific community, Higher education, Industry	14
Organisation of a workshop (planned)	1	We have also submitted (not reviewed yet) an application to organize a tutorial "HPC Software Management with Spack" at ISC High Performance 2018, 24-28 June 2018, Frankfurt (https://www.isc-hpc.com/). Authors: M. Kuhn (Universität Hamburg), S. Kosukhin (Max Planck Institute for Meteorology), G. Becker (Lawrence Livermore National Laboratory), M. Culpo (EPFL).	See costs declared in form C of MPI-M	Scientific community, Higher education, Industry	

Intellectual property rights resulting from this deliverable
Not applicable.

ESiWACE
Application Software Framework: A White Paper

March 2018

ESiWACE has received funding from the European Union's
Horizon 2020 Research and Innovation Programme
under Grant Agreement No 675191

Contents

Document change history

Date	Version	Contributor	Summary of changes
2018-03-02	2.2.3	Sergey Kosukhin	Small edits
2018-03-01	2.2.1	Sergey Kosukhin	Small edits
2018-03-01	2.2.RGB	Reinhard Budich	Typos, Re-ordering, Narrative
2018-03-01	2.2	Sergey Kosukhin	Amendments
2018-02-28	2.1.RGB	Reinhard Budich	Typos, Re-ordering, Narrative
2018-02-28	2.1	Sergey Kosukhin	Amendments
2018-02-27	0.13.RGB	Reinhard Budich	Comments, typos, flow
2018-02-15	0.13	Sergey Kosukhin	Systematization of the text
2016-03-31	0.12	Reinhard Budich	Added ToC, prepared for submission to ESIWACE web site
2016-03-24	0.11	Sergey Kosukhin	Added document change history table
2016-03-24	0.10	See list of authors	First public version

Table of abbreviations

Abbreviation	Explanation
CMIP	Climate Model Intercomparison Project(s)
DoW	Description of Work
ESiWACE	Centre of Excellence in Simulation of Weather and Climate in Europe
ESM	Earth System Model (Earth System Modelling)
IS-ENES2	The second phase project of the distributed e-infrastructure of models, model data and metadata of the European Network for Earth System Modelling
HPC	High Performance Computer (Computing)
NWP	Numerical Weather Prediction
OS	Operating System
PRACE	Partnership for Advanced Computing in Europe
SW	Software
WF	Workflow

Introduction

This white paper sets out to describe the software stack needed to run climate and weather prediction models for research purposes[1]. In this context *climate models* are the numerical realisation of models of the climate as they are used, for example, in the context of the Climate Model Intercomparison Projects (CMIP)[2]. Such realisations are expressed as *source code*: algorithmic translations of formulae into computer readable form, engaging programming languages, which translate the formulas into arithmetic expressions executable on processing elements of modern computers, typically High Performance Computers (HPC).

Modern climate models and their workflows require many pieces of software to be correctly installed: the Application Software Environment for multi-model simulations. Various mathematical, input/output, parallelization and other *software libraries* are necessary to build and run model executables; plotting, statistics, file format conversion and other packages are required for *pre- and post-processing*[3]; *scripting language interpreters* and *filesystem utilities* are essential for linking and controlling different stages of the execution workflows. All these ingredients generate a multitude of software dependencies that have to be resolved and accounted for during the deployment of the software stack required to run a numerical experiment (see Fig. 1).

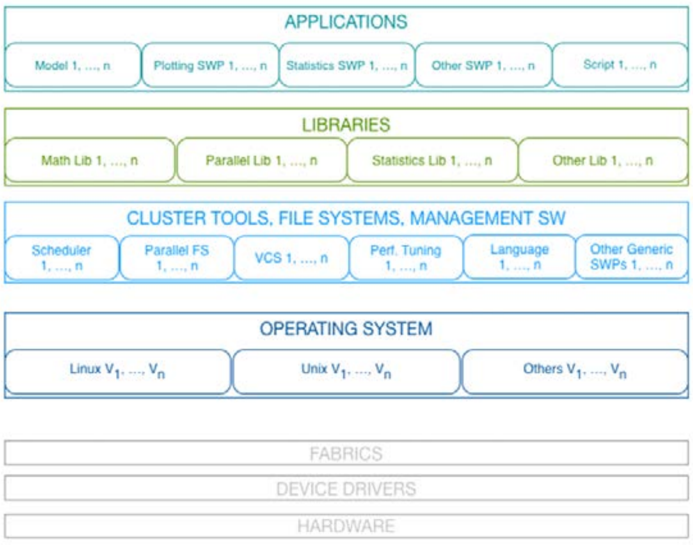


Fig. 1. Software stack

The deployment process is further complicated by the fact that the models are research tools and hence are under continuous development. It is often the case that introduction of changes in the model's source code becomes an essential part of a research workflow. Thus, the software environment for the research workflow must

[1] Where we talk of climate models on this text, we include NWP models for research purposes as default.
[2] https://www.wcrp-climate.org/wgcm-cmip/wgcm-cmip6
[3] We understand the term "post-processing" as a summary of processes that makes the model output easier to perceive, interpret, store, transfer or use as an input for another post-processing procedure. „Pre-processing" here means the preparation of data as input for the models. Not to be confused with preprocessing of the source code, as it is introduced later in this paper.

not only fulfil the runtime requirements of the model execution workflow but also the requirements posed by software development needs. So, not only immediate software dependencies of the models that were mentioned above, but also *version control systems*, *compilers*, *debuggers*, *performance tuning* and other development tools are also part of the software stack that is required to support the lifecycle of a numerical experiment.

On top of that, given the resource requirements of some scientific projects, a single machine is not always enough to accommodate all required model runs. Thus, it becomes mandatory to make the software stack portable to whatever available environment, varying from personal computers to PRACE[4] supercomputers. Therefore, *cluster management tools* and even the *operating system* need to be viewed as part of the complete software stack.

The project ESiWACE[5] (centre of **E**xcellence in **Si**mulation of **W**eather **A**nd **C**limate in **E**urope) aims at a systematic study of the reasons for the deployment difficulties, and attempts to remedy the situation by providing recommendations on what common flaws in the design of the systems might be avoided, and what strategies and methods best applied to make deployment easier, and such create better usability of the model software packages.

Motivation

The Description of Work (DoW) of the *Usability* work package of the ESiWACE project describes the motivation for the White paper in detail:

"Today, it is realized that sophisticated and flexible workflow solutions are increasingly important in production environments. However, the emerging solutions are still far from universal and currently rare in the research environment. IS-ENES2 has established a growing appetite for a step change in capability of workflow solutions in the research environment and this proposal is able to capitalize on recent investments at NIWA[6], the MetO[7], MPG[8] and others aimed specifically at this user base.

ESiWACE has the ambition to significantly improve the interaction between those with deep computing knowledge and those with the best scientific ideas. This way we will drive research in workflows solutions which offer a much greater potential for performance optimization in the non-computer- architecture-minded sense, as does the standard way of experiment design and execution. This will allow for considerable advances in a number of areas:

Large difficulties exist to organize and carry out multi-model ensembles (see projects like PRIMAVERA[9], CRESCENDO[10]) ESIWACE will develop an environment to remedy this situation, including education of young researchers.

[4] http://www.prace-ri.eu
[5] https://www.esiwace.eu
[6] The National Institute of Water and Atmospheric Research, New Zealand
[7] Met Office, United Kingdom
[8] Max-Planck-Gesellschaft zur Förderung der Wissenschaften e.V. / Max-Planck-Institut für Meteorologie, Germany
[9] https://www.primavera-h2020.eu
[10] https://www.crescendoproject.eu

- *The complete stack from the science application down across the complete system and data handling software to the hardware is much more heterogeneous than it is healthy for the communities involved. ESiWACE will provide some counterbalance against the commoditization trend currently observable in the computing industry by testing solutions, proposing and proliferating standards, and educating young scientists in their use.*
- *Information about best practice and working examples is often missing. ESiWACE will improve this with its dissemination methods.*
- *Involvement with solution providers is low. Providing a funded platform to engage and exchange with industry, also directly with ETP4HPC*[11] *by involving an SME*[12] *is a new approach. ESiWACE will gain the attention of the computing industry via greater and better co-ordinated engagement for the ESM community. With this activity ESiWACE will be very involved with the milestones of ETP4HPC, "Programming Environment", from 2016/17 on.*
- *Increased complexity of earth system model suites and the need to automate more data pre- and post-processing means that there is the urgent need to find tools to free the scientists from the increasing burden as HPC resources grow. ESiWACE' activity on meta-scheduling, like the provisioning of the Cylc*[13] *workflow engine, suitable for research and production environments and specifically developed for the climate and weather communities, provides the opportunity to give step-change improvements in the management of complex workflows.*
- *Dissimilar and disparate working environments and software stacks are a hindrance not only for multi-site, multi-model high-resolution full complexity Earth system model experiments, but also for the individual researcher needing to be flexible in terms of the usage of his computational and storage resources across different (topical or PRACE) sites, and for the software engineer in need of benchmarking his model or tool across different platforms. A huge potential lies in the provisioning of recommendations for shared common environments and software stacks across sites and architectures in terms of usability and maintainability.*
- *Rational scheduling of simulations based upon concrete parameters of the according experiments has the potential to exploit machines and resources much more elegantly than possible currently, and will be supported by ESiWACE through training and services for provisioning of technical support."*

Objectives

The above considerations led to the following objectives[14]:

- *Support scientific excellence through provision of effective HPC and big data infrastructures by allowing scientists to more easily design and carry out simulation campaigns that seamlessly exploit the existing multi-model framework, including the inherent value of model diversity.*

[11] http://www.etp4hpc.eu
[12] https://en.wikipedia.org/wiki/Small_and_medium-sized_enterprises
[13] https://cylc.github.io/cylc
[14] We do not include here the tasks dealing with scheduling and co-design.

- *Considerably improve the ease-of-use of the software, computing and data-handling infrastructure for ESM scientists from the applications through the software stack to the hardware.*
- *Reduce the skills gaps at individual centres by sharing best practice through worked examples using use-cases derived from user-driven engagement, the need to prepare the Extreme Scale Demonstrators[15] (EsD), and governance.*

The software stack

Most of the issues related to the software stack and its usability in HPC environments are induced by their hardware structure. In the following we outline the aspects that do not allow users to treat an HPC facility as a single computing unit.

Hardware Considerations

The main computational part of a climate modelling workflow typically implies the use of a *supercomputer*. The majority of modern supercomputers are implemented as *clusters*: loosely coupled parallel computing systems consisting of explicitly distinguishable computing elements, i.e. *nodes*, connected with a high-speed network. Nodes of a cluster have individual memory and instances of the operating system, but usually share a file system.

Modern supercomputers, Tier-0 PRACE machines, for example, comprise thousands of nodes. Such quantitative complexity inevitably leads to qualitative differences in the structure and the interface of a system. The most visible one from the user perspective is that nodes of a supercomputer are divided into groups, each having its own function.

Login (or front-end) nodes serve as access points for users and provide an interface to the computational resources of a cluster. They are not intended for resource-demanding jobs but for "basic" tasks such as data uploading, file management, script editing, software compilation, etc. Paradoxically enough, the most valuable resource – the manpower – is spent on those "basic" tasks, which makes the usability of the software environment of login nodes one of the most important factors for the overall effectiveness of the system.

Computing (or compute, or back-end) nodes are the essence of a cluster. Orchestrated by special software, *clustering middleware*, and communicating with each other over a fast network connection, they jointly run resource-intensive *jobs* submitted by the users from login nodes.

In general, computing nodes are identical to each other because this simplifies workload balancing and increases maintainability. However, their hardware characteristics often significantly differ from the characteristics of the login nodes. Although there are obviously good reasons for this, since login and compute nodes are designed for different purposes, such differences can decrease the usability of the system by introducing non-transparent transition from the user to the code execution environment. For example, the discrepancy between the instruction sets supported by the processors of login and computing nodes might enforce users to take

[15] See the Use Cases below

additional steps to ensure that the code they compile on a login node can be executed on a computing node.

Depending on the characteristics of the workload of HPS systems, computing nodes can be organized in partitions (or "islands") of different architectures. Just to mention the most obvious, there can be homogeneous (CPU only) or heterogeneous (GPUs, FPGAs, HDA, and others, and their combination) layouts for nodes.

These HPC systems typically have many users, often from different scientific domains. This makes it very challenging to fulfil all their special needs with respect to software environments. Providing such different environments is further impeded by the existence of a gap (see Fig. 2) between the areas of expertise and responsibilities of the users on the one hand, and system administrators of the computing facilities on the other.

Software

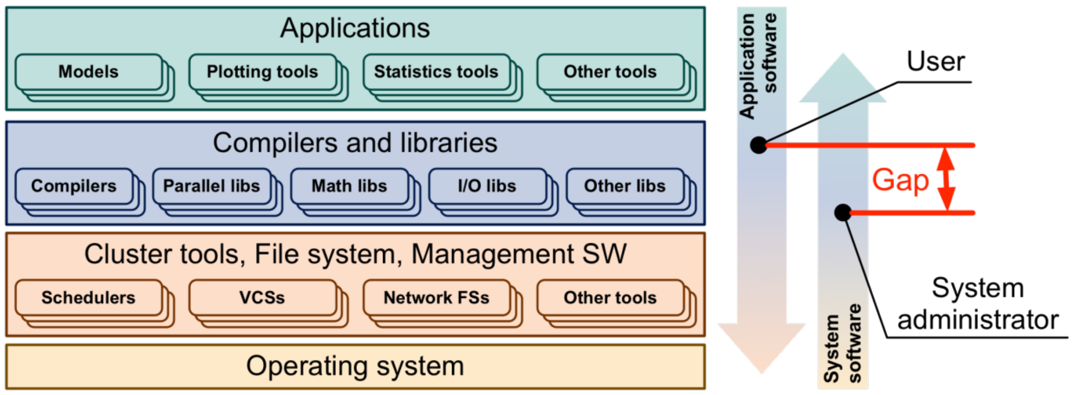


Fig. 2. Transition in the area of responsibilities of users and system administrators

Users are usually familiar only with the *application layer* of the software stack and are not aware of all the aspects of its integration with the *system software*. System administrators, in turn, have the knowledge required for software deployment but cannot foresee all possible requirements and usage scenarios of the applications.

Climate Modelling Workflow

The typical climate modelling workflow is shown in Fig. 3. In this paper, we mainly concentrate on the part in the green circle, the processing workflow; and here on the process to "prepare model experiment".

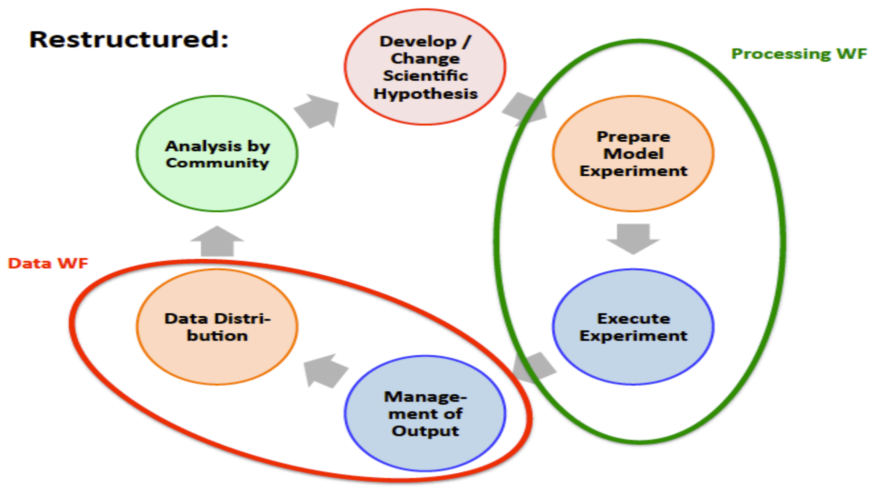
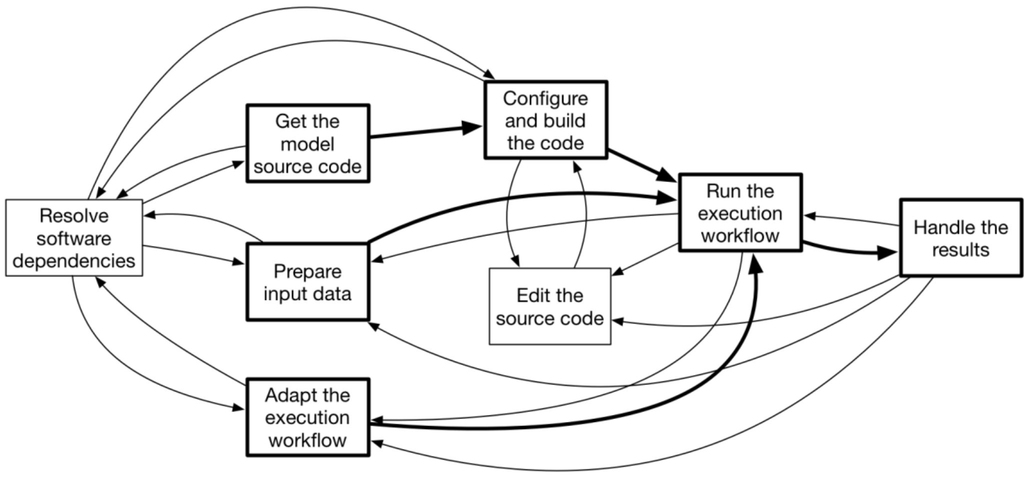


Fig.3: Cycle of Life of a workflow in weather and climate research

The following analysis of the technical part of a typical climate-modelling workflow intends to reduce the gap by providing system administrators of HPC facilities with information on common requirements of the ESM community to the software stack on the one hand, and by familiarizing the users with objective limitations of the software environments and the ways to adjust them to their needs on the other hand.

The technical workflow

A typical research workflow comprises designing the experiment from the scientific idea to the publication of the results, be it as papers, or, becoming more and more important, as the resulting data. There are many steps in the research workflow that imply direct interaction with the software environment in order to solve a particular scientific and, thus, the resulting technical problem (see Fig. 4).


Get the model source code
Configure and build the code
Resolve software dependencies
Run the execution workflow
Handle the results
Prepare input data
Edit the source code
Adapt the execution workflow
Fig 4. The technical part of the research workflow: bold elements represent the idealized case, in which a user goes from left to right. But in the real cases has to make steps back to fix problems identified on the later stages

In the idealized case the technical part of the research workflow is relatively simple and straightforward: the researcher needs to ***get the source code of the model***, ***build*** it, ***prepare input data*** (model configuration files, initial and boundary conditions, etc.), ***adapt*** the script of ***the execution workflow*** to the requirements of the experiment, ***run the execution workflow***, and ***handle the results*** (analyse, visualize, etc.). It should be noted that a lot of efforts from scientific programmers and system administrators are put into attempts to keep it this way. The real cases are often more complex though. *The first reason* for that is the iterative nature of the process: problems identified at each stage might introduce unexpected detours in the progress and turn the ideal into a real case by fixing the problems in a less than optimal way, to say the least. *The second reason* is that the software environment often (and almost always in the case of a new environment that never hosted and ESM experiment before) needs to be adapted to the requirements of the workflow: missing software libraries and tools need to be installed, compiler toolchains need to be tuned, job submission interface needs to be tested, etc. The last interactive stages of the post-processing, such as custom-tailored visualization and analysis of the simulation results, for example, might require very particular software. In such cases, users often move to their personal computers, where they have almost full control over the software environment and thus are able to have the exact tools they need to be productive. There is a significant limiting factor for such transition between machines though: the typically huge amount of data that need to be transferred enforces users either to compromise between usability and performance or to take actions to get the required software dependencies to be installed on the supercomputer.

In the following, we will lead the reader through the technical part of the workflow from Fig. 4, commenting on the activities involved.

“Resolve software dependencies”

It is unlikely that a climate modeller will encounter an operating system which is not some flavour of Linux (RedHat, Centos, SUSE, Debian, Ubuntu, Mint, etc.) or Unix (IBMs AIX, BSD/OS, Sun/Oracle Solaris, etc.). But there are many different distributions, which may contain proprietary software and provide different levels of support. Supercomputers may also run special operating systems on their compute nodes – for example, Cray machines may run Compute Node Linux optimised for stability, robustness, scalability, and performance.

Climate models from different institutions have different library dependencies. Nevertheless, there are generalities which cover many modelling systems. Maintenance of libraries is an essential component of an HPC service, where effort is devoted to ensuring that libraries are consistent with updates in system software. Determining the set of dependencies required for a climate model mostly is not a simple task and ensuring that the dependencies are satisfied on a given platform may also present difficulty. Immediate software dependencies of the climate models can be divided into the following categories, where this is probably not a comprehensive list:

- ***Parallelisation:*** climate models require an implementation of MPI[16]; several are in common use, including MPICH[17] and Open MPI[18]; some of the models

[16] https://en.wikipedia.org/wiki/Message_Passing_Interface
[17] https://www.mpich.org
[18] https://www.open-mpi.org

or post-processing tools might also require OpenMP support, which however is usually provided by compiler toolchains by default.

- ***Input/output (I/O):*** the most commonly used binary formats that are used for input and output data storage are: NetCDF[19] (the modern version of the standard is a particular case of HDF5[20]) and GRIB (both the first and the second versions)[21] (GRIB API[22], which later evolved into ecCodes[23]); climate models generally create vast amounts of data. Modern climate software generally uses asynchronous I/O (like XIOS, CDI-PIO) which can be configured to have minimal impact upon computation time
- ***Scientific libraries like*** fast Fourier transforms (FFTs), parallel random number generators, linear algebra routines (LAPACK, BLAS) or spectral decomposition are heavily used by models, and often are neither bug-free in their interplay, nor are they free of side-effects.
- ***External model components and couplers like e.g.*** OASIS or YAXT can result in problems, too.

System administrators often provide common software to the users, usually by means of tools that let users select among multiple versions of software installed on a system: Environment Modules[24], Lmod[25] are just two such systems. Unfortunately, the provided software is not always enough: either a library (see Fig. 5) or its particular version is often missing.

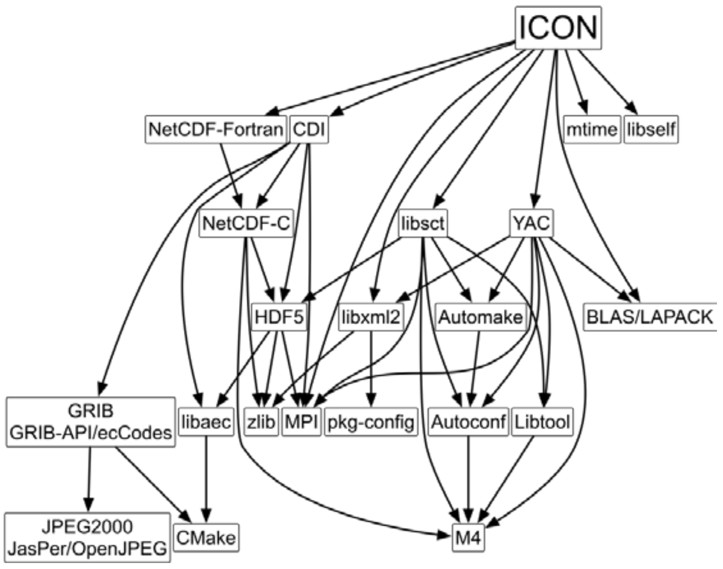


Fig 5. Software dependency tree of ICON model

One way for the users to solve the problem of dependencies is to ask system administrators to install the necessary piece of software. But both the usually excessively high workload of system administrators, and the fact that the requirements to the software environment change unexpectedly often due to the mentioned iterative nature of the workflow, lead to delays before the environment

[19] http://www.unidata.ucar.edu/software/netcdf
[20] https://support.hdfgroup.org/HDF5
[21] https://en.wikipedia.org/wiki/GRIB
[22] https://software.ecmwf.int/wiki/display/GRIB/Home
[23] https://software.ecmwf.int/wiki/display/ECC/ecCodes+Home
[24] http://modules.sourceforge.net
[25] https://lmod.readthedocs.io

can be used productively, often for an unacceptably long period. In such situations users attempt to install the software they need by themselves, which quite often fails for missing access rights due to security considerations. A good approach to overcome these difficulties is to employ a package manager. The most notable ones are EasyBuilds[26] and Spack[27]. The latter has received support from the ESiWACE project (see the Handbook[28]) and currently is at the state that allows it to install most of the software used by the ESM community automatically.

”Get the model source code”

The source code of the models is often distributed via version control systems. Although Git becomes more popular in the climate community, Apache Subversion (SVN) is still heavily used. Thus, availability of the *up-to-date versions* of client programs of both of the systems we consider to be very important.

“Configure and build the source code”

Software that requires performance as climate models do is normally written in compiled languages (Fortran, C, C++). The translation of source code into binary code is usually performed in the following sequence of steps: *configuration*, (source code) *preprocessing*[29], *compilation*, and *linking*. The latter three are usually implicit for users and from their perspective are performed at once during the *building* phase.

The goal of the configuration phase is to gather information required for the building phase. This information includes paths to external libraries, arguments and flags to be passed to a compiler toolchain (preprocessor, compiler, linker), which features and components of the software being built must be enabled or disabled, and so on. The configuration phase is usually automatized by means of a script, an integral part of the source code, associated with a *build script generator*, a program that generates (one or more) files to be used at the building phase by a *build automation tool*[30].

Build automation software:
The most notable build script generation tool, at least in Unix/Linux environments, are Autotools[31] and CMake[32]. Both of them implement means to explore software environment and to generate instructions for the building phase. The wide variety of compilers and environments made it inevitable for both of the systems to have extensive databases of heuristics that enable them to guess the necessary information. Although they work pretty well for the GNU Compiler Collection[33] (GCC) and standard Unix/Linux environments, their methods are not always good enough to handle the cases of more complex environments of supercomputers. The main difference is that the software and hardware environment in which the software is built (development environment) is often not the same as the environment in which the software runs (runtime environment). This is due to the already mentioned hardware and software differences between login and computing nodes.

[26] http://easybuilders.github.io/easybuild
[27] https://spack.io
[28] https://www.esiwace.eu/results/misc/handbook-for-system-administrators/view
[29] Should not be confused with the process of generation of input files for the models, see above
[30] https://en.wikipedia.org/wiki/List_of_build_automation_software
[31] https://www.gnu.org/software/automake/manual/html_node/Autotools-Introduction.html
[32] https://cmake.org
[33] https://gcc.gnu.org

Another reason for better support of the GCC and standard Linux environments by the build script generators is a much larger user community that contributes to their development. The HPC community is smaller, often has to use proprietary solutions, and thus strongly depends on the vendors. The only way to remedy the situation is to draw vendor's attention to this issue.

The result of the configuration phase is a concrete set of instruction for the compiler toolchain, often in the format of so-called Makefiles. At the building stage, the Makefiles are processed by a special program (GNU Make[34] is the one used most) in order to generate the binaries and/or libraries. Among other things, the Makefiles contain either information on the order in which the source file must be compiled, or instructions on how to identify the order. This is another issue that requires special attention. Most of the code developed by ESM community is written in FORTRAN. The problem is that the compilation order is based on the intra-dependency of the source files (e.g. source code in file *A* uses functionality implemented in file *B*). In the case of FORTRAN, identification of such dependencies requires an additional code preprocessing stage. Although the modern version of CMake can partially handle this task, a significant part of the ESM code had been developed before its mature enough versions became available. This leads to the emergence of a wide variety of custom solutions, each having its own software requirements. The general approach to meeting most of the requirements is to ensure the availability of the up-to-date versions (the custom solution are still developed and maintained) of the most popular script language interpreters: Perl, Python, Bash.

Compilers

Most models require FORTRAN and C compliers appropriate for the hardware on which the model will run, but will not run efficiently without having undertaken the effort to tune compiler flags. Some models simply will not run without a particular compiler. In addition, to achieve reproducibility of the results one has to be careful with the selection of compiler flags.

"Run the execution workflow"

We define *execution workflow* as a sequence of *automated* steps that initialize the runtime software environment, perform model runs and run post-processing operations.

Script interpreters

Up-to-date versions of the script interpreters (Python, Perl, Bash) are essential not only for the building stage. Scripts are also heavily used for the workflow coordination and post-processing. The latter often requires extensions of the frameworks that come along with the interpreters. The list of such extensions is provided in the Handbook.

Job scheduling systems

The nature of the climate run and the HPC resource on which the run is performed inevitably leads to the need for a batch job submission system, whereby jobs prepared for submission are managed by the scheduler in order to ensure efficient

[34] https://www.gnu.org/software/make

usage of the shared HPC resource. While all performing similar roles, the details of each differ; typically, an HPC service will support only one scheduler. Schedulers used at many sites include PBS, Loadleveller (IBM), SLURM, Oracle Grid Engine, TORQUE.

Meta-schedulers

Ever increasingly complex workflows call for the use of management software to schedule tasks according to rules determined to ensure the correct ordering and triggering of events. A climate-suite workflow might include data acquisition, data preparation, code extraction, code compilation, mirroring of data and binaries, running the integration, and processing the data generated (data manipulation and transfer). The system must manage workflow on several machines, handle failures and restarts gracefully, and be relatively simple to configure and run. The meta-scheduler may run on the HPC where climate integrations take place or it may run remotely. Several meta-schedulers are in use of which we describe three here:

Autosubmit is a solution created at IC3's Climate Forecasting Unit (CFU) to manage and run the group's experiments. Lack of in house HPC facilities has lead to a software design with very minimal requirements on the HPC that will run the jobs. It is capable to run experiments in clusters or supercomputers with PBS, Slurm, SGE or LSF schedulers. It also can run jobs on any Linux machine that can receive SSH connections.

Prior to version 3, Autosubmit had a fixed workflow matching the one used at IC3 to run EC-Earth experiments. On the new version this limitation has been removed and now it has a limited workflow definition capacity that allows running other models such as WRF or NMMB. Autosubmit is currently being developed at BSC Computational Earth Sciences group.

Cylc suite engine is a workflow engine and meta-scheduler. It specializes in cycling workflows such as those used in weather forecasting and climate modelling, but it can also be used for non-cycling suites. It was created by NIWA and currently is developed by NIWA and UK Met Office.

ecFlow is a workflow package that enables users to run a large number of programs (with dependencies on each other and on time) in a controlled environment. It provides reasonable tolerance for hardware and software failures, combined with good restart capabilities. It is developed and used at ECMWF to manage around half their operational suites across a range of platforms.

ecFlow submits tasks (jobs) and receives acknowledgements from tasks when they change status and when they send events. It does this using commands embedded in the scripts. ecFlow stores the relationship between tasks and is able to submit tasks dependent on triggers.

Criteria	**Autosubmit**	**Cylc**	**ecFlow**
Seniority	2011	2010	2011
Original authors/sponsors	IC3, BSC	NIWA, MetOffice	ECMWF
License	GNU GPL v3	GNU GPL v3	Apache License v2.0

“Prepare input data” and “get the results”

In the typical climate modelling workflow as depicted in Fig. 3 this part is typically located more in the data “bubble”, but obviously also part of the “prepare model experiment” step. Huge datasets (especially in the case of the high-resolution demonstrators) that are passed through the workflow enforce the practice of processing data close to its storage location. For the pre-processing stage this means that the input data for the experiments will be prepared close to the original raw data and then uploaded to a supercomputer. For the post-processing stage this means that the output data of the models need to be processed on the same machine, which was used for computations. The latter requires the availability of analysis and visualisation tools.

Analysis and visualization

Users have their preferred analysis tools and there is a distinction between those commonly used in weather and climate communities. Analysis software should include: IDL, Matlab, Python, R, CF-Python, grib_api, CDO, NCO, IRIS, SciPy, Matplotlib. Some require licences (IDL, Matlab) and specific features may be version dependent or require extra licenses.

“Adapt the execution workflow” and “edit the source code”

On one hand, the major part of the model development is performed in well-known and customized software environments (e.g. user workstations). On the other hand, the code often needs to be changed “in the field”, on a supercomputer to be used for computations: adaptation of execution scripts, optimizations, finding bugs in a particular environment, etc. Even simple text editors with code highlighting are not always available out-of-the-box.

From the software perspective it is important to have at least some knowledge of different programming languages and scripting. Whereas the climate and NWP models are written in programming languages, the steering of the simulations with the models is performed by scripts. These scripts do not only invoke the execution of the simulations with the compiled models, they also invoke custom-made or off-the-shelf tools for the preparation and the post-processing of the simulations.

Users need to be aware of the two main programming language types: compiled languages and interpreted languages. These two families have different characteristics and are targeted to different types of applications. It is important that the user knows that they require different knowledge and handling, and that they will need both of the language types.

Debugging and optimization

Software as complex and configurable as climate models will fail on occasion and for efficient diagnosis and bug fixing, a modern parallel debugging facility is essential. Each utility (e.g. DDT, TotalView) has its own requirements. However, these tools tend to have a quite expensive price that depend on the number of processors used to run the code.

While there are climate modellers who may have little interest in or need to optimize the HPC performance of their integrations, doing so is increasingly important as models run at ever higher resolution and fidelity. Performance-analysis tools (CrayPat, Vampir, Scalasca, BSC Performance Tools, Intel tools, etc.) greatly enhance the opportunities to detect poorly performing code and assist in resolving bottlenecks in communications, computation and I/O.

Performance measurement tools
Tools like Extrae and Paraver are used to analyse the performance and the behaviours of the selected codes on HPC clusters. Extrae is the package devoted to generate Paraver trace-files for a post-mortem analysis. Extrae is a tool that uses different interposition mechanisms to inject probes into the target application so as to gather information regarding the application performance. Paraver is a very flexible data browser for trace files generated by Extrae. Paraver was developed to respond to the need to have a qualitative global perception of the application behaviour by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems.

Use cases (high-resolution demonstrators)

One of the key objectives of the ESiWACE project is *"…the establishment of so-called demonstrators of atmosphere-only, ocean-only and coupled ocean-atmosphere simulations, which will be run at the highest affordable resolutions (target 1 km) to estimate the computability of configurations that will be sufficient to address key scientific challenges in weather and climate prediction."* At the example of the demonstrators we will, in the following, illuminate the "common part strategy" we follow, see table. The software stack necessary to deploy and run the demonstrators comprise the specific modelling software – which we will not comment – and a large number of common tools – which we comment – and which can be rolled out to the target HPC systems as part of our common strategy engaging Spack.

Demonstrators

The following models have been selected as demonstrators:

1. Very high resolution atmosphere-only and ocean-only demonstrators
    - IFS: Integrated Forecast System (developed by ECMWF)
    - ICON: Icosahedral non-hydrostatic general circulation model (developed by DWD, MPI-M and DKRZ)
    - NEMO: Nucleus for European Modelling of the Ocean (developed by IPSL)
2. High resolution coupled atmosphere-ocean demonstrators
    - EC-Earth: European consortium developing a coupled climate model. The configuration analysed uses IFS for atmosphere, NEMO for ocean and OASIS for coupling.
    - ICON-ESM: as coupled climate model using its own ocean component ICON-OCE and the YAC coupler.

Software dependencies

The model source code is, as explained before, obviously model specific – currently: This might change with the introduction of common physics libraries like those developed in ISENES2 for radiation. Also, other configurations of coupled models could comprise e.g. different atmosphere, but the same ocean models.

The common tools include:

Bash: All the models presented in this document are using bash or scripting related languages in different stages of the execution. We consider such tools as standard on Unix/Linux systems and cover any specificities with Spack.

Python and Perl: These two tools are required also in many steps of the models' workflows. The models' workflow scripts are implemented using these two languages, therefore the corresponding interpreters are required to run them. The interpreters are available under names Python and Perl in the Spack repository, which also contains a collection of packages that extend their basic functionality.

Fortran and C compilers: Many Linux/Unix systems come with gfortran/gcc by default, which work well for the models. High Performance Computing facilities may provide other compilers to achieve better performance. The Fortran compiler must support an auto-double (e.g. -r8) capability as some source code files are in fixed format Fortran. OpenMP parallelism depends upon each model. Spack specifications are available for some of the components of the demonstrator models.

MPI implementation: Must be available on the system of choice. It can either be vendor supplied or one of the freely available versions, such as MPICH or OpenMPI. The compatibility of each MPI implementation with the model should be checked with the model developers. Spack can take care of the respective dependencies.

Performance measurement tools: Extrae and Paraver are not required to run the models, but are important in the contest of the work related to the demonstrators; we recommend their use.

Installation

Most of the software packages used can be installed using Spack, work is in progress for the others. This reduces the complexity of the installation task. Spack manages the dependencies, compilers and modules.

Software dependencies of the demonstrators and their execution workflows

Tool/ Library	EC-Earth	ICON-ESM	Website	Package in Spack repository
BLAS	✔	✔	http://www.netlib.org/blas/	openblas
CDO	✔	✔	https://code.mpimet.mpg.de/projects/cdo	cdo
FCM	✔		http://metomi.github.io/fcm/doc/	N/A
GRIB-API	✔	✔	https://software.ecmwf.int/wiki/display/GRIB/Home	grib-api
HDF5	✔	✔	https://support.hdfgroup.org/HDF5/	hdf5

Tool/ Library	EC-Earth	ICON-ESM	Website	Package in Spack repository
LAPACK	✔	✔	http://www.netlib.org/lapack/	openblas
LIBXML2		✔	http://xmlsoft.org	libxml2
NETCDF	✔	✔	http://www.unidata.ucar.edu/software/netcdf	netcdf netcdf-fortran
OASIS	✔		https://verc.enes.org/oasis	N/A
SZIP	✔	✔	https://www.hdfgroup.org/doc_resource/SZIP/	szip
XIOS	✔		http://forge.ipsl.jussieu.fr/ioserver/	N/A
YAC		✔	provided with the ICON model	N/A
ZLIB	✔	✔	http://zlib.net	zlib

Conclusions

Supercomputer systems have to be used for ambitious Earth system modelling. They show significant differences in purpose and size, resulting in substantial qualitative differences in their design, architecture and interfaces. While most of their complexity is hidden from the users by their *clustering middleware* – the software that allows treating an HPC as one large computing unit –, users still have to be aware of some special aspects of distributed computing environments to be able to employ them effectively.

In this paper we use earlier findings in terms of the specification of an Application Software Environment for Multi-Model Simulations (D3.1) and ESM System Software Stack Recommendations (D3.2 to D3.5) and describe the process in more details, that is necessary to establish and deploy such environments in a way they are useable for both development and experimentation.

Our research was focused on further analysis and systematisation of the requirements of Earth system modelling workflows, as well as identification of common issues and limitations related to the deployment and maintenance of software stacks on large supercomputing facilities. The main conclusion we can draw based on the results of the second phase of the research is that users, in order to be productive, have to share with system administrators the responsibility for completeness and usability of the software environment. To achieve that, they have to get a better understanding of the main issues related to the software deployment and to learn about the existing approaches to its automatisation. Thus, providing this information to developers and experimenters has become one of the major objectives of the Whitepaper.

Users educated this way than have to interact with system administrators. Supporting Spack installations will benefit both groups in the following way:

- Users will be able to easily customise the software environment on their own, thus being more productive and reducing the workload put on system administrators.
- Spack allows for formal description of the software stack available on a supercomputer, which simplifies identification of the list of missing software dependencies of a modelling workflow.

- Spack can also help system administrators to easily test various usage scenarios of the basic elements of the software environments, such as compiler toolchains and MPI libraries.

Based on our current view and developments, the Spack approach seems to be valid and useful for the ESiWACE demonstrators and future high-resolution modelling.