S. Götz, L. Linsbauer, I. Schaefer, A. Wortmann (Hrsg.): Software Engineering 2021 Satellite Events,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2021 1

Developing Domain-Specific Languages for Ocean Modeling

Reiner Jung[1] Sven Gundlach[2] Serafim Simonov[3] Wilhelm Hasselbring[4]

Abstract: Climate and Ocean Modeling are complex and long-living software systems. Due to their history and continuous growth, they face challenges due to code complexity, cross-cutting concerns and limited modularization. OceanDSL aims to address these issues by supporting separation of concerns utilizing Domain-Specific Languages. While there are projects addressing numerical modeling and parallelization, we focus on configuration and parametrization as well as deployment and execution. In this paper, we document our efforts regarding configuration and parametrization as well as start the discussion how to introduce DSLs in scientific computing in an efficient and sustainable way.

Keywords: Domain-specific Languages; Model Configuration; Model Parametrization; Scientific Computing

1 Introduction

Ocean models are used to simulate ocean circulation and process in the ocean, like plankton growth and sediment development. They are part of climate models and help to understand climate changes in the past and predict future developments which may effect human ocean related activities, like fisheries and recreational use of the ocean. These models have been developed over decades and are therefore real long-living software projects. Often they started as small single use programs and have grown into larger code bases supported by communities. To the extend of our knowledge, developers have no formal training in software engineering. Usually, they have studied sciences, like geophysics, chemistry and biology. As every aging software, especially without an explicit architecture, ocean models are affected by code complexity and code degradation induced by the many code changes and alternative implementations that must be maintained (cf. section 3). This affects the time spent on extending models and in turn reduce the time available for other research tasks. Thus, it is imperative to reduce the time spent and therefore cost of model development and use. The goal of OceanDSL is to provide Domain-Specific Languages (DSLs) to target specific aspects in developing and using numerical models including configuration, parametrization, execution, as more technical domains, and bio-geo-chemical modeling, as a scientific domain [JGH19].

[1] Kiel University, Christian-Albrechts-PLatz 4, 24118 Kiel, Germany reiner.jung@email.uni-kiel.de
[2] Kiel University, Christian-Albrechts-Platz 4, 24118 Kiel, Germany sven.gundlach@email.uni-kiel.de
[3] Kiel University, Christian-Albrechts-Platz 4, 24118 Kiel, Germany stu126367@mail.uni-kiel.de
[4] Kiel University, Christian-Albrechts-Platz 4, 24118 Kiel, Germany hasselbring@email.uni-kiel.de

Currently, we focus on two major challenges within this goal: (a) Understanding the domain of ocean modeling, as this affects how the DSLs must be tailored and implemented to ensure their usefulness to scientists; (b) specifying model configurations, like feature selection and parametrization of models as well as the relationship to development and usage processes around ocean models.

In this paper, we report shortly on key properties of the domain of ocean modeling and discuss requirements for DSLs with a focus on configuration and parametrization in section 4. We introduce the current state of our configuration and parametrization DSL and the ideas behind its current structure in section 5. We provide an overview of related activities planned in our project and in other DSL developments in the domain in section 6. Finally, we discuss aspects and concepts missing from the DSL and potential solutions (section 7). This should also serve as a starter to an open discussion on envisioned and existing language features. Furthermore, we want to discuss aims to maintain DSLs over longer time periods, especially with varying funding support.

2 Development Approach

We follow, at large, the DSL development process laid out by Mernik et al. [MHS05]. This process consists of five phases: Decision, Analysis, Design, Implementation, and Deployment. The Decision phase focus on getting a glimpse of the domain to decide whether DSL development is advisable, which has been completed by granting research money. In the Analysis phase, domain knowledge is gathered, usually, based on existing domain artifacts, questionnaires and interviews. Furthermore, we collected other DSLs used or developed for the ocean modeling community. However, configuration and deployment are not addressed by these DSLs or are incomplete, i.e., not compilable. Based on the domain knowledge, we drive the Design, Implementation and Deployment phases iteratively adding features to the DSLs. The deployment phase consists of two evaluation methods, (a) we test the DSL based on ocean model experiments used as case studies, and (b) we involve scientists to use the DSL in their work environment.

Interviews To gain domain knowledge, we focused on interviews with potential DSL users. We used on *semi-structured interviews*, as they require no preparation by the interviewee and provide guidance for the interviewer. Thus, we can control the interview and gain statements on the topics we are interested, while simultaneously are flexible enough adapt to emerging ideas and concepts during an interview. We limited the length of the interviews both to reduce the effort for interviewees, making it more likely to participate in the interview and limit the amount of time required to analyze the interview recording. Our *interview guide* focused on the processes involved in developing ocean models, the development and execution environment, and the involved social interactions and project structure.

Thematic Analysis While often informal approaches are used to gain insights in interview data during requirements engineering, we aimed to reduce personal biases and extract systematically concepts and ideas. Thus, we looked into potential methods, e.g., Grounded Theory and Thematic Analysis. The former is a rather complex approach comprising a specific methods which are often implemented incomplete or even misused in software engineering [SRF16]. Thus, we selected Thematic Analysis (TA) [BC06]. Thematic Analysis is an qualitative method to analyze text and other media. It originates from psychology and sociology. It is flexible, lightweight, and can be customized to our needs.

For our analysis, we first transcribed the interviews, codified the expressions in the transcriptions and categorized them. This is an incremental process, as the understanding of the texts grows. A category can be a grouping of codes, like *Editors* and *Model Contribution*. Based on the categories, a set of themes are derived. A theme can be seen as a narrative of topic within the interview, e.g., *Modeling Process* and *Roles and Interactions*. In some cases a category can directly be marked as a theme, in others multiple categories contribute to one theme. The building of categories and themes is an interactive and iterative process. A brief summary of some domain properties are presented in section 3.

3 Ocean Modeling

Ocean modeling is a complex domain involving development processes, customs, and collaboration approaches. In this paper, we will give a brief introduction on four roles and interactions, an abbreviated development and execution process relating to these roles, the experiment types, and a short introduction to the architecture and the handling of versions and variants in this domain.

Roles and Interactions In our analysis, we identified seven distinct roles in context of ocean modeling. For this paper, we focus on four central roles involved directly with the development and maintenance of models.

- The *Scientific Modeler* develops new and adapts or extends existing mathematical models to address newly discovered properties of nature.
- The *Model Developer* is tasked with transferring these mathematical models into code and has to consider the scientific aspect of the code and the technical aspect, like parallelization and numerical limitations.
- The *Modeling RSE* is often a former scientist with extensive experience in the first two roles. He or she supports the model developer, communicates best practice, guidelines and the envisioned architecture.
- The *Gatekeeper* is a special type of Modeling RSE and tasked to control contribution to community models. They interact with Model Developers and Modeling RSE from

various institutions to ensure quality standards and cleanup the code in conjunction with the contributor. In contrast to Modeling RSEs who are often from the same organization, the Gatekeeper is from the hosting organization and handles contributions form different research institutions.

Modeling and Execution Process Based on the interview data and our analysis, we discovered several processes and sub-processes. For this paper, we provide an overview of the modeling and execution process, depicted in Figure 1.

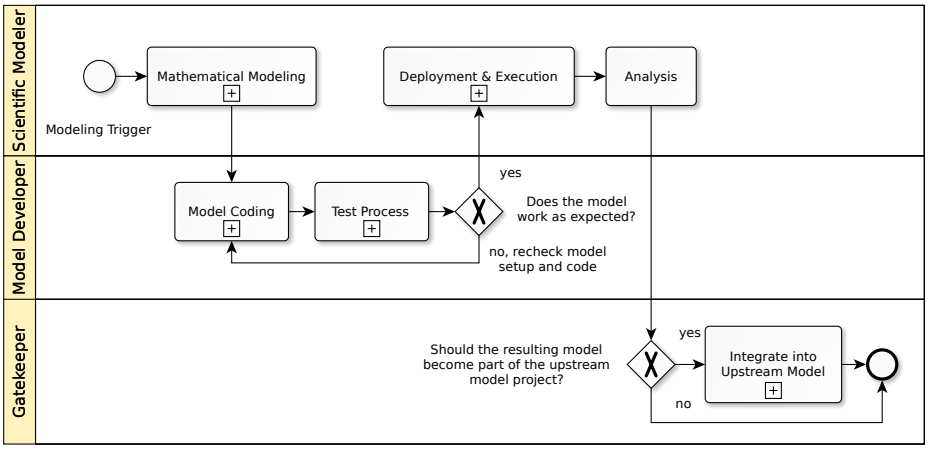

Fig. 1: Abbreviated Modeling and Execution process, depicting three roles involved in modeling. The Modeling RSE is omitted, as he/she only provides assistance to the Model Developer.

Here, three of the roles, introduced above, are present in the process. The Modeling RSE has usually more a consulting role in specific tasks and is therefore not part of Figure 1.

In the beginning of the modeling and execution process, new knowledge on natural process has been discovered, like behavior of plankton, which must now be integrated in the ocean model. First, a scientist in the role of a Scientific Modeler develops a mathematical model for the plankton. Second, a Model Developer transfers this model into program code. In this step scientific code representing the mathematical model and technical code for model integration and parallelization are combined. Furthermore, numerical limitations play a significant role in the implementation. Third, the resulting model is tested based on previous model runs, with specific testing scenarios and scientifically guided interpretation of the results. Fourth, in case the model behaves as intended, the model gets deployed and executed. As the development often takes place on the same hardware as the scientific model runs to avoid different behaviors based on technical differences between computer systems, the deployment is merely configuration and parametrization of the model for the intended scenarios. Fifth, after the model has been executed, the results are collected and analyzed.

Execution and analysis are controlled by the role of the scientific modeler. Based on the results other research might be launched and findings are published, sometime including the code contribution. However, this is not always the case. Finally, the new model code might be of interest to the community. In that case, the contribution must be integrated in the upstream model. This is a sub-process governed by the Gatekeeper who might modify the code contribution to align it with coding standards. To ensure the scientific properties of the model, the Model Developer is also involved in this sub-process.

Models and Experiments Our interviewees use a wide variety of models and model versions. For our research, we currently focus on MIT General Circulation Model (MITgcm) [Ar10] from the Massachusetts Institute of Technology (MIT) and the Earth System Climate Model, called UVic, from the University of Victoria (Canada) [We01]. These two are executable on PCs, while other models, like Nucleus for European Modeling of the Ocean (NEMO) [Au15] and Icosahedral Nonhydrostatic Weather and Climate Model (ICON) [MP20] require High-Performance Computing (HPC) facilities to run.

Scientists use these models for at least three different types of experiments: (a) Scenario-based predictions, like in the assessment of climate change. Here different CO_2 profiles are used. Usually, these scenarios are run multiple times to gain probability distributions. (b) Sensitivity studies where the main goal is to test the model's stability/sensitivity to certain changes. These studies are used to evaluate the validity of models under certain conditions and to detect tipping points. (c) Model calibration aims to find and check start values by comparing model output with observation data.

Architecture Earth system climate models comprise several models including one for the atmosphere and the ocean. The ocean model itself comprises at least a transport model simulating the movement of bodies of water and models describing the water column, e.g., sediments and plankton. As the ocean and the atmosphere interact, the models have to share information. This is done by couplers which also map different spatial properties. Spatial models, like ocean and atmosphere models, use different principles to cover the entire globe.

Earlier models use a grid based layout. These have the downside that the distance between grid points vary over the globe, i.e., at the poles grid point converge while at the equator they are the furthest apart. Thus, today triangle based meshes are used and sometimes finer meshes around coastlines and areas of special interest.

Another option is to overlay certain areas. Here a global model with a wide mesh is used to cover the whole globe, but for a small area a denser mesh is overlaid. The denser and wider meshes are then either coupled using a coupler to provide interpolation or these are directly integrated, as it is the case with NEMO utilizing AGRIF [AI16]. The latter is called nesting.

Handling Versions and Variants Scientists rely on versions and variants of their models. Versions play a role for publications and to document experiments. They can be used to replicate model runs. These versions do not only contain the model code, but often also configurations and parametrizations as well as input data used for the experiment.

In contrast variants occur when models are prepared for specific experiments or moved between hardware platforms which require regularly adjustments to the implementation due to concurrency issues, different numeric behavior between HPC installations and different versions of used libraries.

Unfortunately, scientists do not consequently distinguish versions and variants. In some cases variants grow and change on different platforms in separate source trees. Over time sharing code and patches becomes problematic. Fortunately, nowadays scientists mostly rely on Git to handle versions and variants, where still different source trees can be located on different machines, but code is shared via pull requests. In case of larger modeling efforts, there exists an hierarchy of code repositories. On the lowest level are the repositories controlled by a single scientist. That scientist might have one major repository on his workstation and downstream repositories on different HPC installations. Upstream from the scientist are the repositories of the research group or collaboration, the institute and – in case of a community model – the community repository.

Configure and Parameterize Models Before models can be compiled and then executed, they need to be configured and parametrized. The *configuration* determines which code portions are included in the resulting program, i.e., select program features, sub-models and adaptations to the hardware and execution environment, including parallelization and Interprocess Communication (IPC). The *parametrization* covers all the settings and inputs necessary to run a model for a specific experiment. However, not all parameters can be set at runtime. Thus, they are set at compile-time, like mesh and grid sizes. Our configuration and parametrization DSL targets this aspect of modeling.

Related domains to configuration and parametrization are compiling, deployment and execution. Compilation is handled by a build tool in all analyzed models. Typical standard build tools are Make and CMake, supplemented by additional dependency management scripts written in Bash, Korn Shell and Python. Deployment, i.e., moving a compiled artifact and all necessary files to an execution environment is not a typical procedure in modeling, as models are compiled on the machine and in the location they are going to be executed. However, input data files referenced in the parametrization or required by internal definitions in the model must be downloaded and prepared to be usable for the model. Furthermore, depending on scenario requirements, different configuration and parameter settings as well as input data must be used. These requirements are addressed by a separate deployment DSL. However, they rely on the ability to modularize the configuration and parametrization.

4 Domain-specific Requirements

The development of a configuration and parametrization DSL for ocean models must adhere to specific requirements induced by the domain and the way scientists work with models. The development and runtime environment have to be considered including languages and tooling. The long live-span of the models and the necessity that the tooling must be usable for a similar long time important aspects for the users.

Larger models are usually written in Fortran and C/C++. Currently, models use Fortran 90 or newer, while initially have been written in older Fortran revisions, e.e., Fortran 77. Developers make use of the C-preprocessor to handle variants in the code, for example, to activate certain sub-models or model features. Some smaller models have been written in Python often utilizing special Python extensions written in C/C++, like FeNICS [Al15].

Depending on the used languages and execution context scientists use different editors and IDEs. Larger models are often edited remote in a shell. Thus, IDEs like Eclipse, with rich UIs are not used, as they are slow when used remote and hard to install on HPC platforms. In some cases, scientists have used PyCharm to develop models in Python locally. However, in most cases scientists use Vim/Vi and Emacs to edit their code. Compilation is done using the build system provided by the model. In some cases these are special self-developed build scripts and in other cases Make and CMake are used as build system.

A key issue with introducing new tooling is the long live-span of models and therefore the need to have tooling which is maintained or at least available in a working condition for a similar long time. Thus, any new tool must provide the means to be maintainable and/or usable without changes for decades.

5 Configuration and Parametrization DSL

In this section, we introduce shortly the two case studies used in our research, followed by a brief discussion of the language structure and the tool integration.

Supported Models We use two model projects as case studies: MITgcm [Ar10] and UVic [We01]. MITgcm is a well documented and modularized model, developed by the Massachusetts Institute of Technology. Thus, technical properties can be understood quickly. It provides a wide variety of test cases and tutorial examples which we use as scenarios. The UVic model has a more complex code structure which was referred to by scientists as Spaghetti code. However, it is a model used by collaborating scientists, which we can consult on scenarios and serve as users for the language evaluation.

Language Structure The DSL must be able to include or inherit parts of a configuration. Moreover, it must be able to support different models to provide a unified experience to

Scientific Modelers. Still it must be as specific as possible for each model to complicated methods to reference model identifiers or do not provide any support the the Scientific Modeler. In case the DSL is too generic, Scientific Modelers will have no advantage in using the DSL which will render it useless. Therefore, we aim to identify common properties in configuration and parametrization in models.

Through our analysis of models, we identified at least two general structural properties. Models allow to activate or deactivate model features, like different friction behaviors. Or they allow to use different sub-models, like different bio-geo-chemical models and they include different analytical and service components, like logging and diagnostics.

Depending on the model, there can be global features as well as sub-model and component specific features. Furthermore, there can be global and model/component specific parameter settings. Therefore, the DSL allows to select features on a global level and together with parameters as well as select modules, representing sub-models, packages and components, which can then have their own feature and parameters. At least in MITgcm, parameters are grouped in specific parameter lists. For the time being, we replicate this behavior in our DSL, as the documentation for MITgcm refers to these parameter groups and it is beneficial when the group names are reflected in the configuration files. In Listing 1, an excerpt of the configuration file for the barotropic_gyre example is shown.

```
include size

barotropic_gyre : mitgcm

features ALLOW_FRICTIONHEATING

PARM01:
    viscAh = 4.E2
    f0 = 1.E-4
    beta = 1.E-11
    rhoConst = 1000.0
    gBaro = 9.81

module cost:
    features ALLOW_EGM96_ERROR_COV
    cost_nml:
        mult_atl = 0.
        mult_test = 0.

diagnostics:
    diagMdsDir = "some-dir"
    format = netcdf
    diagSt_regMaskFile = "regMask_lat24.bin"
```

```
set_regMask(1:3) = [ 1, 1, 1 ]
val_regMask(1:3) = [ 1., 2., 3. ]

"first-out.log":
    logmode = snap
    frequency = 10
    missing_value = 5.0
    fields(1:2) = [ SDIAG1, SDIAG2 ]
    levels(1:2) = [ 1,2 ]
```

Listing 1: Excerpt of a configuration file `barotropic_gyre.oconf` including global settings and module specific settings

The Listing starts with an include from another configuration named `size` that could contain any number of settings which are included in the resulting configuration. In case a setting is defined in `size` and redefined in the `barotropic_gyre.oconf`, the latter setting is used.

The second line names the experiment, here `barotropic_gyre`, and states that a configuration for the MITgcm model must be generated. This loads a parameter declaration model in the background which restricts which settings are present for configuration.

Features and parameters can be specified on the global level, here `ALLOW_FRICTIONHEATING` and `PARM01` group, and on a module level, here for the module `cost`, which uses feature `ALLOW_EGM96_ERROR_COV` and sets some cost values.

The last section in the configuration excerpt, depicted in Listing 1, is called `diagnostics`. It declares what and how to log runtime variables of the model. In principle, `diagnostics` could have been modeled like any other module. However, we want to provide a more concise and restricted grammar for this aspect. This has the benefit to provide focus for the Scientific Modeler and allows to introduce specific checks and constraints for settings. We can also hide technical aspects, like Fortran file numbers, from the user. Whether such specific language features are appreciated by Scientific Modelers has to be determined.

Tool Integration Scientific Modelers and Model Developers have expressed their dislike of IDEs, as most IDEs are not available on HPC or are hard to set up, are not performant and provide minimal benefits in their context compared to simpler editors. In some cases IDEs were not able to support specific libraries used by Model Developers. Thus, our DSL, which has been developed using XText, providing an editor and generators for Eclipse, must be usable with Vim and Emacs. Therefore, we utilize the Language Server Protocol (LSP), introduced by Microsoft in 2016 [Co16]. LSP defines an interface for editors to query a language server on syntactical and semantical checks for a given code fragment. It does not handle compilation. Thus, we provide a command line compiler which can be integrated in any build system, like an additional compiler or preprocessor. As Jupyter is gaining popularity in sciences and especially in modeling, we also utilize the LSP extension

for Jupyter to provide editor support in Jupyter. We complement the editor with a special kernel that generates proper configuration and parametrization files for the used model.

6 Research Agenda

Our research agenda includes the development of DSLs supporting configuration, parametrization, deployment and execution control, modeling bio-geo-chemical models and setup for transport models. In the near future, we focus on the configuration and parametrization DSL. We will realize all language features to cover the domain and provide generators for the two models MITgcm [Ar10] and Uvic [We01]. To make the DSL accessible to scientists, we will provide fully functional extensions for at least Vim or Jupyter relying on LSP and an Eclipse integration as feature reference. Alongside working extensions, we will provide a working bundle for BinderHub [Bi17] and installation instructions.

There are other projects introducing DSLs to the numerical modeling community. Here we are especially interested in PSyclone [Ad19] and Dusk/Dawn [Me20] which are developed by the Meteorological Office, UK and MeteoSwiss, respectively, as well as, the forerunner of OceanDSL, the Sprat approach [JH17]. PSyclone is essentially a configurable, in-place code transformation platform. Its aim is to provide special new statements which are embedded in Fortran code. The transformation then translates these special statements into Fortran code and stores the result in a pure Fortran file. This concept is called embedded DSL and adds new syntactical structure to Fortran. Dusk, in contrast, is an external DSL which uses the Dawn compiler backend to produce C/C++ code. Finally, the Sprat approach includes internal and external DSLs to address numerical modeling and parametrization.

Both projects address transport and mathematical modeling, which could in principle be used to for bio-geo-chemical models. Thus, they are related work and can serve as a source of ideas as platform by using their infrastructure, and as an option to contribute code to addressing the future maintenance aspect.

Finally, we aim to evaluate our DSL in 2021 together with our partners. We will rely on working infrastructure, as discussed above. First, we introduce the tooling to the potential audience in an online tutorial open to our partners and interested scientists. Second, we will apply our tooling to their current research, replicating their tasks on our infrastructure. This is necessary to identify potential issues and obstacles users may face. Third, we will support scientists in using our tooling to perform their modeling efforts.

7 Conclusion

In this paper, we presented our efforts to develop a configuration and parametrization DSL for numerical models, especially addressing the ocean. We characterized the domain and core requirements of the DSL, sketched the basic building blocks of the DSL in its

current form and presented our research agenda in context with the configuration and parametrization DSL.

For this workshop we are interested in discussing core issues of the language design, maintenance and social aspects of the domain and the introduction of the DSL.

Language-oriented questions are:

- Should we use curly braces, rely on indentation like Python or follow a syntax similar to YAML? Model Developers and Scientific Modelers are familiar with Fortran, CPP, C, Fortran namelists, and Python.
- Currently, the modularization of configuration files relies on the `include` statement. Values that are redefined in a file overwrite included definitions. However, this might not be the safest strategy. Potential solutions would include the necessity to state the intention with an overwrite modifier, mark redefinitions as errors and require to explicitly define which values must be defined in the importing file, similarly to an interface.
- Shall we introduce more specialized sections, like diagnostics, e.g., IO handling, or better cover everything with the more generic module structure?

Regarding maintenance and social aspects, we have the following questions:

- OceanDSL is a research project and as such limited in resources and time. Thus, maintenance must be done in subsequent projects or by interested users. While we aim to make maintenance as easy as possible, time must be spent to update and maintain the language in future. How can institutions or individuals be motivated to invest time and money to ensure further development?
- Key to get the DSL accepted by users are benefits for the user. Typical arguments against a DSL are the necessity to learn a new language, limited support by tools, hard to integrate in the existing workflow and the risk of losing support in future. We are seeking measures to address these issues and arguments that reduce the anxiety towards new tooling. One measure is to allow people to transition back to previous means. Still, more details on an introduction strategy are desired.

Acknowlegement Funded by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation) – HA 2038/8-1 – 425916241.

References

[Ad19] Adams, S.; Ford, R.; Hambley, M.; Hobson, J.; Kavčič, I.; Maynard, C.; Melvin, T.; Müller, E.; Mullerworth, S.; Porter, A.; Rezny, M.; Shipway, B.;

Wong, R.: LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models. Journal of Parallel and Distributed Computing 132/, pp. 383–396, 2019.

[AI16] AIRSEA Team: ARGIF – Adaptive GRid Refinement in Fortran, 2016.

[Al15] Alnæs, M. S.; Blechta, J.; Hake, J.; Johansson, A.; Kehlet, B.; Logg, A.; Richardson, C.; Ring, J.; Rognes, M. E.; Wells, G. N.: The FEniCS Project Version 1.5. Archive of Numerical Software 3/100, 2015.

[Ar10] Artale, V.; Calmanti, S.; Carillo, A.; Dell'Aquila, A.; Herrmann, M.; Pisacane, G.; Ruti, P.; Sannino, G.; Struglia, M.; Giorgi, F.; Bi, X.; Pal, J.; Rauscher, S.: An atmosphere–ocean regional climate model for the Mediterranean area: assessment of a present climate simulation. Climate Dynamics 35/5, pp. 721–740, 2010.

[Au15] Aumont, O.; Ethé, C.; Tagliabue, A.; Bopp, L.; Gehlen, M.: PISCES-v2: an ocean biogeochemical model for carbon and ecosystem studies. Geoscientific Model Development 8/8, pp. 2465–2513, 2015.

[BC06] Braun, V.; Clarke, V.: Using thematic analysis in psychology. Qualitative Research in Psychology 3/2, pp. 77–101, 2006.

[Bi17] BinderHub: BinderHub, 2017.

[Co16] Corporation, M.: Language Server Protocol, 2016.

[JGH19] Jung, R.; Gundlach, S.; Hasselbring, W.: OceanDSL – Domain-Specific Languages for Ocean Modeling and Simulation, 2019.

[JH17] Johanson, A.; Hasselbring, W.: Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment. Empirical Software Engineering 22/4, pp. 2206–2236, Aug. 2017.

[Me20] MeteoSwiss: Dawn – Compiler toolchain to enable generation of high-level DSLs for geophysical fluid dynamics models, 2020.

[MHS05] Mernik, M.; Heering, J.; Sloane, A. M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37/4, pp. 316–344, Dec. 2005.

[MP20] MPI: ICON – ESM, 2020.

[SRF16] Stol, K.; Ralph, P.; Fitzgerald, B.: Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In. IEEE/ACM 38th International Conference on Software Engineering (ICSE), pp. 120–131, 2016.

[We01] Weaver, A. J.; Eby, M.; Wiebe, E. C.; Bitz, C. M.; Duffy, P. B.; Ewen, T. L.; Fanning, A. F.; Holland, M. M.; MacFadyen, A.; Matthews, H. D.; Meissner, K. J.; Saenko, O.; Schmittner, A.; Wang, H.; Yoshimori, M.: The UVic earth system climate model: Model description, climatology, and applications to past, present and future climates. Atmosphere-Ocean 39/4, pp. 361–428, 2001.