https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions
Open Access
EGU

# CP-DSL: Supporting Configuration and Parametrization of Ocean Models with UVic (2.9) and MITgcm (67w)

Reiner Jung[1], Sven Gundlach[2], and Wilhelm Hasselbring[3]

[1]Software Engineering Group, Department of Computer Science, Kiel University, Kiel, Germany
[2]Software Engineering Group, Department of Computer Science, Kiel University, Kiel, Germany
[3]Software Engineering Group, Department of Computer Science, Kiel University, Kiel, Germany

**Correspondence:** Wilhelm Hasselbring (hasselbring@email.uni-kiel.de)

**Abstract.** Ocean models are long-living software systems facing challenges with increasing complexity, architecture erosion, and managing legacy code. These challenges increase maintenance costs in development and use, which reduces the time and resources available for research. Software engineering addresses these challenges by separation of concerns and modularization. One particular approach is to separate concerns by tailor-made notations, i.e. Domain-Specific Languages (DSLs). Using DSLs, the model developer can focus on one concern at a time without the need to consider other concerns of a software system simultaneously. In ocean and climate models, DSL tooling, like PSyclone and Dusk/Dawn, is used for instance to separate scientific and technical code.

CP-DSL complements this approach with a focus on configuration and parametrization, which play an important role in ocean models, especially in parameter optimization and scenario-based simulations. CP-DSL is designed to be model agnostic and provides a unified interface to different ocean models. Furthermore, the DSL can be integrated into tools and processes used by domain experts. In this paper we report on the DSL design, implementation, and the evaluation with scientists and research software engineers. The implementation of CP-DSL is available as open source software and a replication package for configuration and parameterization of UVic and MITgcm is provided.

## 1 Introduction

Ocean modeling provides mathematical abstractions of the real ocean and builds software systems as implementations of these mathematical abstractions. The developed software systems are then called *ocean models*. Ocean models are used to study the behavior of natural processes in the ocean, like currents, $CO_2$ uptake, and growth of plankton. They are used in earth system models together with models for the atmosphere and land surface, and are themselves comprised of dedicated sub-models for transport, sediments and biological processes (Collins et al., 2005; Warner et al., 2008; Alexander and Easterbrook, 2015).

Successful ocean models are long-living complex software systems. They often start as small projects and grow over years in size and community (Aumont et al., 2015; Weaver et al., 2001). Due to the continually applied extensions and modifications over the years, ocean models face typical issues of long-living software, like code complexity, legacy code, architecture erosion; similar to software from other domains (Goltz et al., 2015). This impedes maintenance and thus hinders scientific progress as scientists spend extra time on technical problems rather than on scientific challenges.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions
Open Access
EGU

Concepts from software engineering such as modularization, separation of concerns and abstraction address these specific challenges of long-living software (Reussner et al., 2019). One approach to realize separation of concerns is the use of different levels of abstractions facilitated by Domain-Specific Languages (DSLs). Such programming approaches, like PSyclone (Adams et al., 2019) and Dusk/Dawn (MeteoSwiss, 2020a), aim to separate technical concerns of coding, like parallelization and optimization, from scientific concerns, like implementing circulation and bio-geo-chemical processes.

While the aforementioned projects PSyclone and Dusk/Dawn address the discretization and implementation of Partial Differential Equations (PDEs), we focus in the present paper on the configuration and parametrization of ocean and climate models to supplement the whole process of development and application. This simplifies the overall process by summarizing configuration and parametrization in a central specification.

We develop CP-DSL to supplement modeling and simulation languages by separating scientific model code from its configuration and parametrization in a unified manner. This approach also makes the configuration and parameterization more independent of the scientific code base and thus simplifies the transfer between different versions and models.

Configuration and parametrization are distinct concerns. Configuration addresses the selection of features and code to be used for the model, as well as, the build configuration. Parametrization addresses, among others, model resolution and time step length. However, on a technical level in ocean models, they are usually connected, as certain parameters are set in the code directly and in include files, as well as, in parameter option files and Fortran namelists. Similarly, some features are controlled via `#define` and `#undef` preprocessor directives, while others are flags to be activated and deactivated at start time. This mix of configuration and parametrization has often historical reasons, e.g., in the beginning all configurations and parameters were directly set in the code. Then, recompiling became too cumbersome for every parameter change. Thus, some parameters were externalized and made available via parameter files while others remain in the code.

The design and implementation of the CP-DSL faces various challenges:

1. The development and maintenance of a DSL in general causes additional effort, which must be balanced by the gain for the scientific modelers.
2. Ocean models vary in the approaches used to configure and parametrize them.
3. Ocean models employ varying techniques to realize configuration and parametrization, e.g. preprocessor declarations and Fortran namelists.
4. Model developers and scientific modelers use established tools and development environments when working on and with ocean models. To be able to introduce a new DSL into their environment, it should be compatible with the existing development and work processes.
5. The DSL has to provide benefits to model developers and scientific modelers without introducing new risks.
6. As ocean and climate models are extended and modified over time, the DSL must be adjustable by model developers to support new configuration and parameter settings without requiring them to adjust the code generators themselves.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

We address these challenges with a flexible and extensible DSL for parameter and configuration options of ocean models. To reduce the time and effort required for DSL development and integration, the DSL is applicable to different ocean and climate modeling projects by coherently implementing specific ocean model support through plugins. Thereby, CP-DSL should be usable as a cross project tool, like Make or as intended by PSyclone and Dusk/Dawn. Due to the interleaved nature of configuration and parametrization in ocean models, we address both concerns with one language. For code generation, we use a set of generators and transformations that can be customized to reduce the necessity for model developers to implement or change code generators themselves. Finally, to address potential deprecation concerns regarding DSLs, our generators carry comments from the DSL into the generated code and setup files. Additional information derived from the documentation may be added to explain the settings. Thus, the generated files are always human readable and documented. This allows use of the generated files, in case the DSL support is suspended for some reason. This also allows for better reproducibility, since the generated files are comprehensible.

We evaluated CP-DSL initially on the ocean models UVic (Weaver et al., 2001) and MITgcm (Artale et al., 2010). For our evaluation we included three increasingly complex ocean modeling experiments from MITgcm to analyze the feasibility of CP-DSL experimentally and the usability empirically. For feasibility, generated setup files were validated and test participants (ocean modelers) were asked to parameterize and configure scenarios. Subsequently, questionnaires were conducted with these domain experts to analyze the usability.

Before we present CP-DSL, we introduce related work in Section 2 and the domain of ocean models in Section 3. Based on the domain properties and our current case studies – UVic and MITgcm – we present language and tool requirements for CP-DSL in Section 4. We present the DSL design and its generic architecture in Section 5. The implementation choices are explained in Section 6. Section 7 reports on the empirical evaluation. Finally, we present a summary and an outlook in Section 8.

## 2 Related Work

Domain-specific Languages (DSL) are introduced to improve software engineering for computational science (Johanson and Hasselbring, 2018). In the following, we discuss related approaches to ocean modeling (Subsection 2.1) and logging / diagnosis of ocean models (Subsection 2.2).

### 2.1 DSLs for Ocean Modeling

PSyclone is an internal DSL based on in-place code transformation (Adams et al., 2019). PSyclone is an *internal* DSL whose language features are embedded into Fortran as host language. It targets the domain of numerical ocean modeling and aims to separate scientific code from technical code, e.g. parallelization, memory management and logging. PSyclone provides a set of extensions to Fortran syntax, which are replaced by standard Fortran code via transformation. The code transformation is mainly implemented in Python. PSyclone is developed by the Met Office UK, the UK weather service. It has been adapted to support the GOcean API (National Oceanography Center, 2020) which is related to NEMO (Aumont et al., 2015). Furthermore,

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

the PSyclone code generator is designed to be customized by model developers what, in principle, allows ocean modeling projects to customize their own code generation.

In contrast to PSyclone, Dusk/Dawn provides an *external* DSL (MeteoSwiss, 2020a) and a code generator and optimizer (MeteoSwiss, 2020b) developed by MeteoSwiss, the Federal Office of Meteorology and Climatology of Switzerland. As an external DSL, Dusk is not embedded into some host language; it comes with its own syntax and requires a specific code generator. Like PSyclone, Dusk separates scientific and technical code. Dusk also provides a rich syntax for mathematical formulae and provides optimization through Dawn. C++ code is generated from Dusk.

The hierarchical Sprat DSLs target marine ecosystems modeling and use a multi-layered scientific modeling approach, where each layer represents one separate concern supported by its own DSL (Johanson and Hasselbring, 2017; Johanson et al., 2017). Its four layers address deployment, simulation parametrization, ecosystem modeling and FEM PDE solver setup. The simulation specification is based on Xtext (Bettini, 2016), like our own configuration and parametrization DSL CP-DSL (see Section 5). The latter two Sprat DSLs are internal DSLs embedded in C++. Sprat provides features, such as content assistance for programmers and high level error messaging, e.g. extended checks on unit types and their correct conversion.

Configuration and parametrization of scientific models is not addressed by PSyclone, Dawn or Sprat. Thus, our CP-DSL complements these scientific modeling projects.

### 2.2 DSLs for Logging and Diagnosis of Ocean Models

A special concern of scientific models is the logging of diagnoses and scientific model states (also called history). For this purpose the XML-IO-Server (XIOS) – first developed by Meurdesoif et al. (2012) – can be utilized. It is intended as a framework and IO-server for scientific models and was first implemented in Fortran. XIOS was reimplemented in C++ providing a rich API to be used in a scientific model. It can be configured programmatically and via an XML configuration file. XIOS has been tested and used with different scientific models, including NEMO (Aumont et al., 2015) and LFric (Adams et al., 2019). Another technology for logging is CDI-pio (Kleberg et al., 2017) which is an extended version of the Climate Data Interface (CDI) (Schulzweida, 2019). CDI-pio is configured programmatically and used in various scientific models, including ICON (MPI, 2020) and ECHAM (Stevens et al., 2013). Both, CDI-pio and XIOS, are general solutions for logging, including diagnostics and scientific model state. Yet they do not specifically address diagnostics.

Ocean models, like our case studies UVic (Weaver et al., 2001) and MITgcm (Artale et al., 2010) come with their own diagnostics implementation. MITgcm uses Fortran namelists to setup diagnostics logging, while UVic uses a programmatic approach. While there exists a variety of technologies, such as XIOS (Meurdesoif et al., 2012) and CDI-pio (Kleberg et al., 2017), addressing diagnostic logging, they are diverse in the way they are setup. In some ocean models, multiple technologies are used. Therefore, CP-DSL aims to provide a unified syntax for this concern.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

## 3 Working with Ocean Models

Since CP-DSL is designed for ocean modeling, the language requirements were identified from domain experts. To achieve this, an analysis was conducted with domain experts prior to the design and implementation phase of the DSL. For the analysis of the interviews, we chose the Thematic Analysis approach tailored to our questions (Braun and Clarke, 2006). In this section, we give a brief introduction to the results concerning roles, processes, experiment types, software architectures and version/variant management that we identified in this domain.

The results are based on several interviews conducted with scientists and engineers involved in ocean model development from the GEOMAR Helmholtz Centre for Ocean Research Kiel, the German Climate Computing Center (DKRZ) and the Max Planck Institute for Meteorology (MPI). Our interview partners use a wide variety of scientific model versions and variants.

Some of the identified roles are introduced in Subsection 3.1, the modeling and simulation process in Subsection 3.2, simulation experiments in Subsection 3.3, software architectures in Subsection 3.4, handling of versions and variants in Subsection 3.5, and the configuration and parameterization of ocean models in Subsection 3.6. For more details on the thematic analysis results (not required for the present paper) refer to Jung et al. (2021b).

### 3.1 Roles and Interactions

A role conducts certain activities related to ocean models. Each role can be taken by multiple persons and one person can take several roles. In our interview analysis, we identified seven roles in the context of ocean modeling. For a detailed description, refer to Jung et al. (2021b). For this paper, we focus on three of these roles directly involved in developing and maintaining ocean models:

- The *Scientific Modeler* develops new and adapts or extends existing ocean models to address open research questions. Scientific modelers are software users parameterizing and running the ocean models to conduct research. Thereby, they are developing and checking the underlying theory and the plausibility of simulation results.
- The *Model Developer* is a software developer and responsible for transferring the ocean models into code and has to consider the scientific aspect of the code as well as technical aspects, like parallelization. Model developers also deploy the software, and submit feature or bug requests.
- The *Gatekeeper* is responsible to control contributions to the ocean models. They interact with model developers from various institutions to ensure quality standards in submissions and clean up code in collaboration with the contributors. The gatekeeper is from the organization hosting the ocean models. Therefore, they are also model developers, but are responsible for ensuring code styles and best practices in the common code base.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions
Open Access
EGU

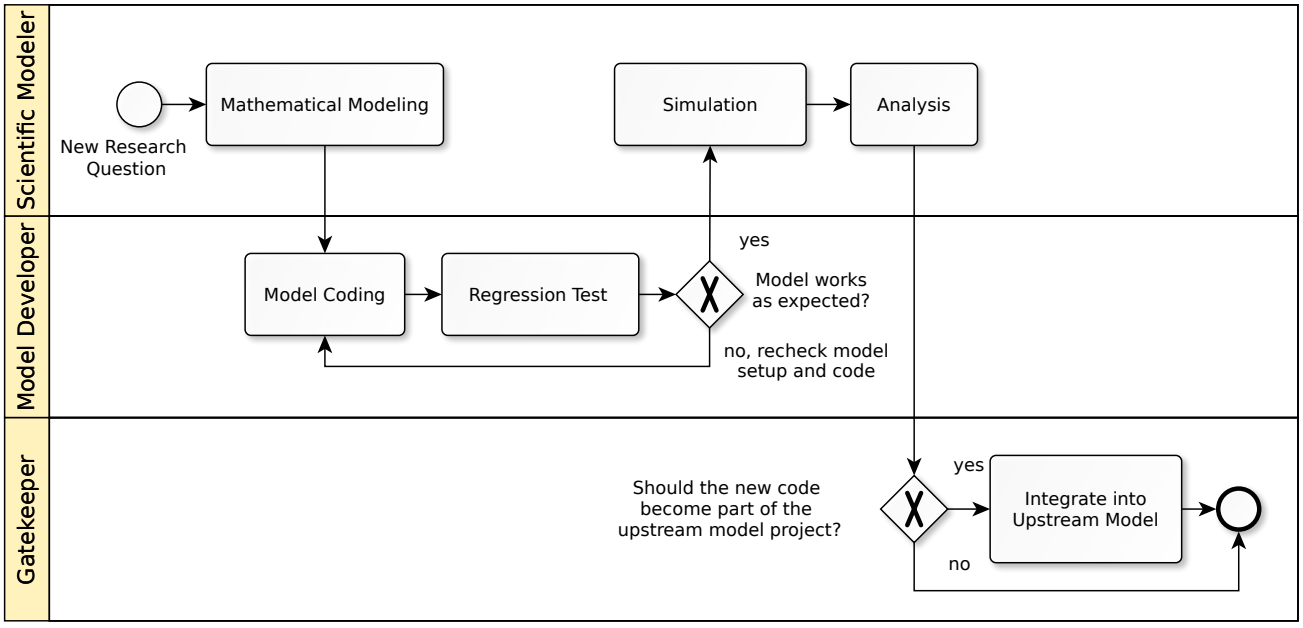


**Figure 1.** Extract of the ocean modeling and simulation process, depicting the three roles Scientific Modeler, Model Developer and Gatekeeper. The process is specified with the standardized Business Process Modeling Notation (BPMN) (Chinosi and Trombetta, 2012)

### 3.2 Ocean Modeling and Simulation Process

Based on the analyzed interview data, we identified several processes and sub-processes (Jung et al., 2021b). For this paper, we only provide an overview of the ocean modeling and simulation process. Here, the three roles introduced above are involved in the process, depicted in Figure 1.

Let us illustrate this process with an example: Assume that new observations of natural processes were discovered, such as the behavior of plankton, which now need to be integrated into the ocean model:

1. A scientist in the role of a scientific modeler develops an ocean model for the plankton, including formulating the mathematical formulas.
2. A model developer transfers this mathematical ocean model into program code, which then becomes the executable ocean model. In this step, scientific code representing the mathematical ocean model is developed and combined with technical code for the scientific model integration and parallelization. Furthermore, numerical limitations play a significant role in the implementation. This is even more important if multiple hardware platforms should be supported, which is often done via separate implementations and multiple build chains managed by preprocessors.
3. Regression tests are applied to the updated ocean model to test the results against previous ocean model runs, with specific test scenarios and a scientifically guided interpretation of the results.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

4. In case the regression tests pass, the model simulation is executed. As the development often takes place on the same hardware to avoid different behaviors caused by technical differences between computer systems, the deployment is merely configuration and parametrization of the ocean model for the intended scenarios. The scientific modeler monitors the ocean model during runtime and adjusts its parameters to support the ocean model.

5. After the ocean model has been executed, the results are collected and analyzed. Due to the size of the ocean model output, most of the data is stored on the runtime system and only selected records are retrieved in full. For version control, tracking and reproducibility, the results are stored together with the setup files. Based on the results, other research might be launched and findings are published, preferably including the code contribution (Hasselbring et al., 2020).

6. The new ocean model code might be of interest to the community. If this is the case, the contribution is submitted and integrated into an upstream ocean model. This is a sub-process governed by the gatekeeper who ensures coding standards.

### 3.3 Simulation Experiments

Scientists use the ocean models for at least three different types of experiments:

1. Scenario-based predictions, like in the assessment of climate change. For instance, various $CO_2$ profiles could be used. Usually, these scenarios are run multiple times to gain probability distributions.

2. Sensitivity studies are conducted to test the ocean model's stability/sensitivity to certain changes. These studies are used to evaluate the validity of ocean models under certain conditions and to detect tipping points.

3. Ocean model calibration aims to find and check start values by comparing ocean model output with observation data.

### 3.4 Software Architectures

Ocean models, as part of earth system models, usually comprise multiple scientific models including one for the atmosphere and one for the ocean (Collins et al., 2005). The ocean model itself includes at least a scientific model to simulate the circulation in the ocean, and a scientific model describing the behavior in the water column, e.g. sediments and plankton. As the ocean and the atmosphere interact, the scientific models have to share information. This is done by couplers which also map different spatial properties and resolutions, e.g., different grid and mesh sizes. In some cases these meshes can have varying densities. Another option for scientific model interaction is called nesting. Here a global ocean model with a wide mesh is used to cover the whole globe. For specific areas, a denser mesh may be 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 (AIRSEA Team, 2016).

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

### 3.5 Handling Versions and Variants

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

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

Meanwhile, scientists usually rely on Git to handle versions and variants and collaborate on platforms such as GitHub and GitLab to contribute code via pull requests.

### 3.6 Configuration and Parameterization of Ocean Models

Before ocean models can be compiled and executed, they need to be configured and parametrized. The *configuration* specifies which code portions are included in the resulting program, e.g. selecting program features, scientific sub-models, build configuration, adaptations to the hardware and execution environment including parallelization like OpenMP (Dagum and Menon, 1998). The *parametrization* covers all the settings and inputs necessary to run a ocean 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. Handling this aspect of ocean modeling is one of the goals of CP-DSL.

Related domains to configuration and parametrization are compilation, deployment and execution. Compilation is managed by a build tool in all analyzed ocean 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 ocean modeling, as ocean models are compiled on the machine and in the location they will be executed. However, input data files referenced in the parametrization or required by internal definitions in the ocean model must be uploaded and prepared to be usable for the ocean 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, not covered in the present paper.

## 4 Requirements on CP-DSL

Based on the domain analysis as summarized in the previous Section 3, we derived requirements for CP-DSL. Ocean model configuration and parametrization are a common issue across different ocean modeling systems. To avoid the development of multiple ocean-model-specific languages and to address a wider community, an extensible, ocean-model-independent approach is favorable. This is also the aim of some related DSL projects targeting ocean and climate models, like PSyclone and Dusk/Dawn (see Section 2).

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

Our observation with various ocean models is that the tasks of configuration and parametrization are often interleaved on a technical level. Thus, generating only parametrization, and subsequently integrating configuration settings from a different source would require code weaving of generated and handwritten code, which is a discouraged pattern in software engineering. In consequence, the DSL should address both concerns together. To avoid overly large setting files and to support reuse, the DSL should also support modularization concepts, like includes.

The development environment of scientific modelers and model developers typically includes remote development and remote execution. Coding is sometimes done locally, but may also be performed remotely through a remote terminal with text editors, like Vim and Emacs. The ocean model is then also executed and controlled via remote commands, as are the analyses of ocean model output data. Thus, the complete development environment can be remote. Therefore, scientists rely on tools which are available and usable in command shells, remotely and locally. Consequently, the CP-DSL should also be able to run both on remote (e.g., HPC) and local platforms. As mentioned in the introduction to the present paper, new tooling should be compatible with the existing development and work processes in the domain.

The toolchain of scientific modelers and model developers typically consists of Vim and Emacs as editors with some utilizing Jupyter notebooks for editing, ocean model execution, and analysis. Scientific model projects rely on Make, CMake, and custom build tools where standard built tools are insufficient. Furthermore, they use open source and proprietary compilers, e.g. the GNU Compiler Collection and the Intel Fortran compiler, whereby some are not available on every platform. Therefore, the DSL must be able to integrate with existing editors and build tools, i.e, provide extensions for existing editors and code generators usable on the command line.

Code management and sharing via tar-balls, ssh and email is mostly deprecated and replaced by Git, often with the Git Large File Storage (Git LFS) extension. The projects and developers use multiple repositories loosely following a repository hierarchy with the community repository at the top, followed by institute-wide repositories, research-group repositories and two layers of individual repositories. However, patches can also be shared between repositories on the similar level without going through the respective upstream repository. Since merge conflicts can arise in such multi-repository and multi-branch setups, good readability of files is advised to resolve potential conflicts (McKee et al., 2017). Thus, the DSL should use a plain, flat textual representation – such as YAML – instead of hierarchically nested structures – such as XML or JSON – which are difficult to merge.

Ocean models evolve over time, which requires adapting the configuration and parameterization settings for a specific model. Thus, it is advisable to provide the means to extend the supported settings without the need to rebuild the tooling for the CP-DSL. This also helps to reduce the maintenance cost of the DSL and it helps to standardize the way to introduce new settings into models.

Finally, scientists are familiar with and work in a number of programming languages such as Fortran, C, Python, MatLab, and R. Thus, syntactical structures similar to these languages can be supportive for its adoption, as this allows to draw upon existing experience.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

## 5 Design of CP-DSL

In the following, we explain the design decisions for the DSL (Subsection 5.1), followed by a presentation of the abstract (Subsection 5.2) and concrete (Subsection 5.3) syntax of CP-DSL. Subsection 5.4 explains the type semantics.

### 5.1 General Design Decisions for CP-DSL

The general design decisions for CP-DSL are derived form the elicited requirements, as presented in the previous Section 4. The CP-DSL syntax should fit to other languages that scientific modelers and model developers are familiar with. This helps to incorporate the new tools into their work environment (cf. Challenge 4 in the introduction).

To avoid nested braces, which are not a common concept in Fortran or Python, we follow syntaxes present in established configuration formats, namely YAML and to some extent Python. However, in contrast to Python, indentation is not part of the syntax of CP-DSL.

CP-DSL allows to set ocean model features, such as friction heating, and to set parameter values grouped by parameter groups. As ocean models can have global and module-related features and parameters, they can be set on a global level or in specific local modules. A module in this context can be a specific sub-model, e.g., bio-geo-chemical model and ocean circulation model, as we found in UVic or a more fine grained set of features, as in MITgcm.

For each simulation experiment, we need to define a configuration and a parametrization. Independent of a specific experiment, we declare settings that are specific to an ocean model. Thus, the parameters and configurable features are declared with CP-DSL in a Declaration Model specific to each supported ocean model, as depicted in Figure 2. The Configuration Model is independent of a specific an ocean model, it defines the settings of a concrete experiment, whereby the declarations in the Declaration Model for the specific ocean model are referenced. This way, we separate the ocean-model-independent and the ocean-model-dependent settings. If you intend to conduct the same experiment with multiple ocean models, you can use one Configuration Model and link this to the Declaration Models of the respective ocean models.

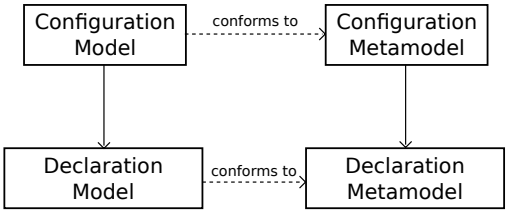


**Figure 2.** With CP-DSL, the Configuration Model is independent of a specific ocean model, while the referenced Declaration Model is specific for an ocean model, such as MITgcm and UVic. The models (CP-DSL specifications) need to conform to the corresponding metamodels (which define the syntax of CP-DSL specifications).

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions
Open Access
EGU

## 5.2 Abstract Syntax of CP-DSL

Due to the nature of the domain of CP-DSL, the terms model and modeling are used both to describe models of the ocean in ocean science and models of software in software engineering. In software engineering, models are built as abstractions of the software systems themselves. In this subsection, we present models of software: the abstract syntax of CP-DSL as *metamodels* via the standardized Unified Modeling Notation UML (UML 2.5.1).

The concrete syntax of a programming language is defined by a context-free grammar (see Subsection 5.3 below). It consists of a set of rules that define the way programs look like to the programmer. The purpose of the abstract syntax, presented here, is to have a simpler representation of what is *essential* in the syntax (Aho et al., 2007).

Figure 3 defines the top-level structure of CP-DSL specifications as a UML class diagram. The DeclarationModel *contains* module, feature and parameter specifications. With the UML, filled rhombi at the association lines define containment relationships between classes. Features can belong to the global DeclarationModel or a local module which both serve as root nodes of a feature tree. Thus, features and parameters may be specified both on the global level of the DeclarationModel and local to modules. The CP-DSL supports modules which can contain their own feature and parameter declarations.

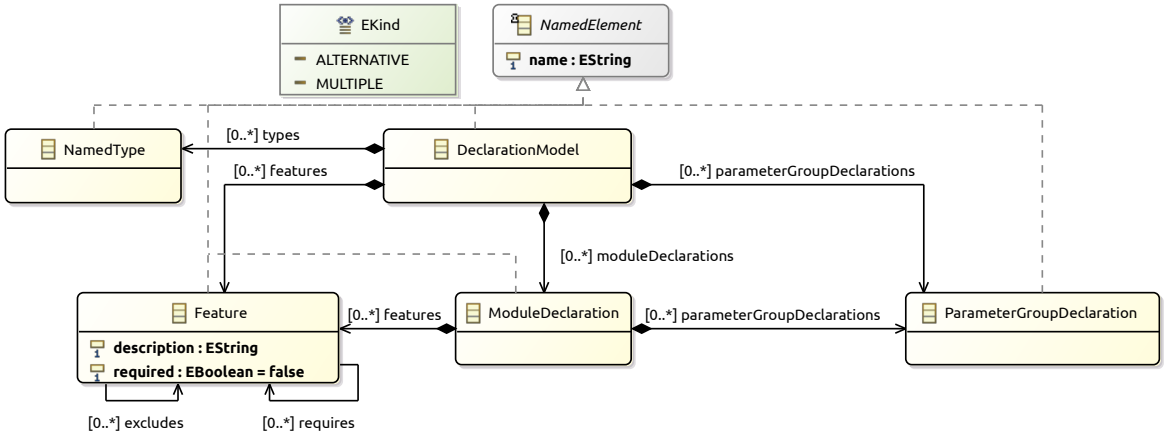


**Figure 3.** Top-level structure of CP-DSL specifications

Features allow a user to select specific code portions in the ocean model. However, some features might be alternatives to each other and features might require additional features. Thus, feature declarations in CP-DSL allow to specify such dependencies (excludes and requires associations of class Feature in Figure 3). A Feature can require 0 or more other features and can exclude 0 or more other features.

Each feature has a name (inherited from NamedElement), a description which is shown in the editor as context assist information, and a flag showing whether a feature is mandatory or optional. A feature may contain sub-features. These are grouped in a FeatureGroup, as the refined metamodel for features in Figure 4 shows. The model allows to have multiple feature groups. Each feature group can contain multiple features, which are either alternatives or can be used in parallel.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions
Open Access
EGU

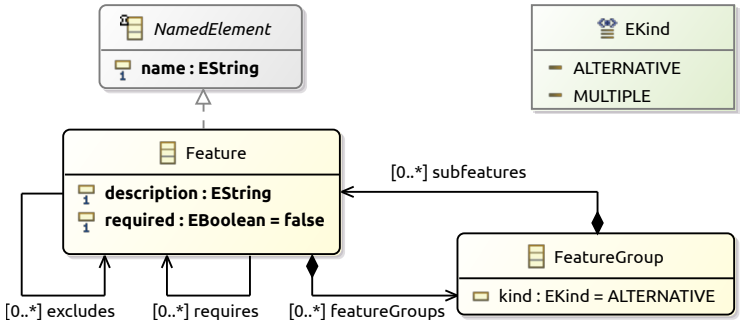


**Figure 4.** Refined metamodel excerpt for feature modeling

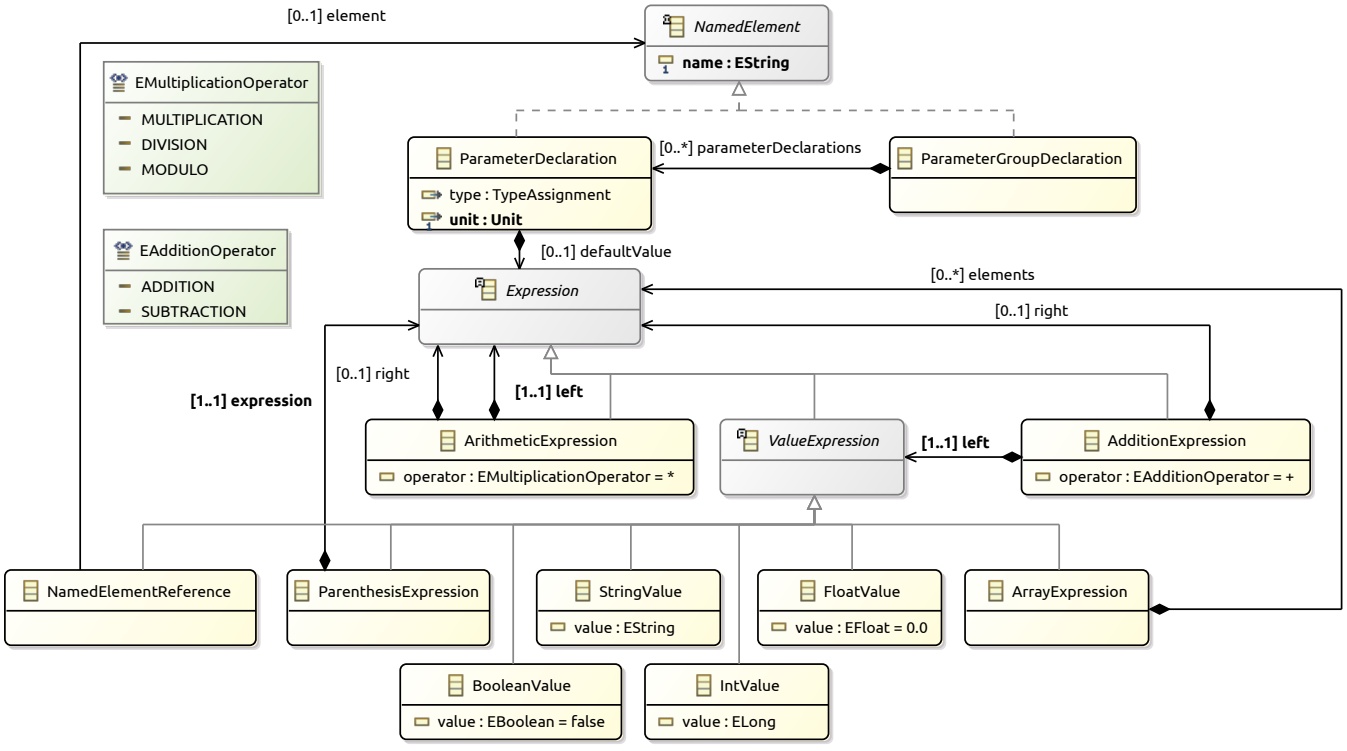


**Figure 5.** Metamodel excerpt for parameter modeling

Figure 5 depicts the metamodel excerpt for parameter declarations. A ParameterDeclaration contains the type and unit of a parameter as it is required by the specific ocean model, while in a CP-DSL configuration specification, a user can specify any compatible unit which will then automatically be normalized. Thus, the CP-DSL provides the additional benefit of checks to parameter values that might result otherwise in erroneous results which are hard to identify. The unit can either be an SI unit, like meter (m), an arbitrary amount label, like count, or coordinate values, like longitude and latitude. The use of units allows to support unit conversion. For example, if a parameter requires its value in meter (m), then the scientist can also specify the

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions
Open Access
EGU

value in km and it is automatically translated to the correct meter value in the configuration file, e.g., $2km/h$ will become $2 * 1/3.6m/s$. As ocean models may use a wide range of parameters, the parameter declarations are grouped in declaration groups (ParameterGroupDeclaration in Figure 3).

Types can be the primitive types Integer, Float, String, File, and Boolean, as well as, ocean-model-specific enumeration and array types. As Fortran before Fortran 2003 did not have enumerations, developers used various approaches to implement them. The CP-DSL supports such approaches on the code level providing enumeration values to the developer on the ocean model level.

In addition to name, type and unit, a ParameterDeclaration contains optional default values (see the containment association between ParameterDeclaration and Expression in Figure 5). Beside a plain default value, it is also possible to define expressions as defaults. This functionality is used, e.g., for grid size parameters computed based on tile sizes, number of tiles, and border points around the core grid of ocean models.

Beside the generic module syntax, we implemented a specific syntax for the concern of ocean model diagnostics. Diagnostics refer to log files that are written during a simulation to support the model developers to assess the run of the ocean model and to check whether everything is working properly. Figure 6 shows a metamodel extract for diagnostics and logging. Here, diagnostics parameters and log file information are defined.

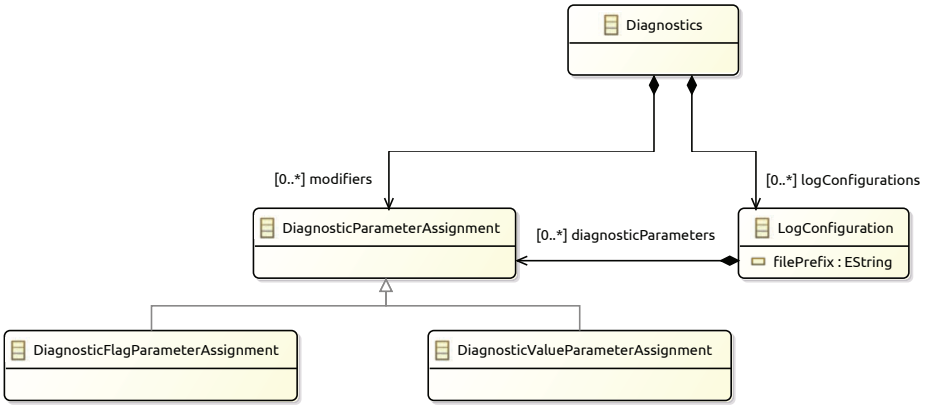


**Figure 6.** Metamodel excerpt for diagnostics and logging

### 5.3 Concrete Syntax of CP-DSL

In this section, we present an extract of CP-DSL's concrete syntax in Figure 7, which is defined as a context-free grammar using the EBNF (Aho et al., 2007). With the EBNF (Extended Backus Naur Form), optional parts of the syntax are enclosed in brackets. Here, this is the case for defining global features, parameter groups, and modules. Alternatives are specified via a |. A module can have its own set of features and parameter groups (not included in the excerpt of Figure 7). The complete grammar can be found in the replication package (Jung et al., 2021a).

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

The start for the excerpt in Figure 7 is `<model-setup>`. The `ID` left of the colon defines the name of the experiment. The second `ID` identifies the ocean model. This information is required to make the matching feature and parameter declarations available for a ocean model, and to select the right code generators for it. An example for selecting MITgcm for an experiment called barotropic gyre (Artale et al., 2010) follows:

```
barotropic_gyre : mitgcm
```

⟨*model-setup*⟩ ::= ID ':' ID ['features' ⟨*features*⟩] ['parameters' ⟨*parameter-groups*⟩] [⟨*modules*⟩]

⟨*features*⟩ ::= ⟨*features*⟩ ',' ⟨*feature*⟩ | ⟨*feature*⟩

⟨*feature*⟩ ::= ID

⟨*parameter-groups*⟩ ::= ⟨*parameter-groups*⟩ ⟨*parameter-group*⟩ | ⟨*parameter-group*⟩

⟨*parameter-group*⟩ ::= ID '{' ⟨*parameter-assignments*⟩ '}'

⟨*parameter-assignments*⟩ ::= ⟨*parameter-assignments*⟩ ⟨*parameter-assignment*⟩ | ⟨*parameter-assignment*⟩

⟨*parameter-assignment*⟩ ::= ID [ '(' ⟨*dimensions*⟩ ')' ] = ⟨*value*⟩ [⟨*unit*⟩]

⟨*dimension*⟩ ::= ⟨*dimension*⟩ | ⟨*dimensions*⟩ ',' ⟨*dimension*⟩

⟨*dimension*⟩ ::= INT | INT ':' INT

⟨*config-expression*⟩ ::= ⟨*multiply-expression*⟩ [('+'|'-') ⟨*config-expression*⟩]

⟨*multiply-expression*⟩ ::= ⟨*literal*⟩ [ ('*'|'/'|'%') ⟨*multiply-expression*⟩]

⟨*literal*⟩ ::= ⟨*primitive*⟩ | ⟨*array*⟩ | '(' ⟨*configuration-expression*⟩ ')'

⟨*array*⟩ ::= '[' ⟨*primitive*⟩ [',' primitive ]* ']'

⟨*primitive*⟩ ::= ⟨*float-number*⟩ | ⟨*int-number*⟩ | ⟨*text*⟩ | ⟨*boolean*⟩ | ⟨*enumeration-value-reference*⟩ | ⟨*parameter-value-reference*⟩

⟨*enumeration-value-reference*⟩ ::= ID

⟨*parameter-value-reference*⟩ ::= [ ID '.' ] ID [ '(' ⟨*dimensions*⟩ ')' ]

**Figure 7.** Excerpt of the context-free grammmer for CP-DSL in EBNF (Extended Backus Naur Form). The complete DSL can be found in Jung et al. (2021a).

### 5.3.1 Features

Features are used to activate and deactivate certain configuration options of an ocean model. They usually relate to `#ifdefs` in the code and allow to create different variants of a ocean model. In Listing 1, the feature for 'friction heating' is activated

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

and 'geothermal flux' is deactivated. While features are deactivated by default, CP-DSL allows to import other configurations. It is possible to deactivate imported features.

**Listing 1.** Activating and deactivating features

```
barotropic_gyre : mitgcm
features:
    ALLOW_FRICTION_HEATING,
    !ALLOW_GEOTHERMAL_FLUX
```

### 5.3.2 Parameters

Parameters, specify configuration and parametrization values, which can be scalars or arrays. CP-DSL supports multidimensional arrays. To set a single cell of an array, its cell number must be specified, e.g., `levels(1) = 5`. To fill a complete array, only the parameter name is specified and an array value of the same size on the right-hand side, e.g., `levels = [ 1,2,3,4,5 ]`. However, in some ocean models not all fields of an array need to be initialized and the underlying array declaration may not start at the target language default, i.e., 0 for C and 1 in Fortran. Thus, we allow to specify a range as dimension expression, e.g., `levels(4:5) = [ 1,2 ]`, where the first and last index are specified.

Parameters are grouped in CP-DSL to offer a lean way to put related parameters together or follow the configuration file layout of the ocean model. For example, MITgcm (Artale et al., 2010) uses different namelist files for general data setup, grid size, and the use of parallel threads. An example parameter group from MITgcm is depicted in Listing 2.

**Listing 2.** Global parameter group PARM01 of MITgcm

```
PARM01:
  viscAh: 4.E2
  f0: 1.E-4
  rhoConst: 1000.
  gBaro: 9.81 m/s^2
  tempStepping: false
```

Besides assigning individual values or arrays to parameters, the right hand side of the assignment can be a numerical expression which includes values and parameter references. In case the referenced parameter can be referred to by name, only the name needs to be specified. In cases where different parameter groups contain a parameter with the same name, the required parameter group is placed in front with the dot notation, .e.g., `Nx = SIZE.sNx*SIZE.nSx*SIZE.nPx`.

### 5.3.3 Modules

Modules represent parts or packages of ocean models that address a specific concern in an ocean model. Often these are scientific sub-models, like biogeochemical models, or specific functionality, like diagnostics. In the CP-DSL, a module consists

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


of a name and a set of features and parameter groups (see Listing 3). We chose the term *module*, as it is a known term in the domain to group functionality together.

**Listing 3.** Configuration excerpt of the cost module of MITgcm

```
module cost:
  features:
    ALLOW_EGM96_ERROR_COV

  cost_nml:
    mult_atl = 0.
    mult_test = 0.
    mult_tracer = 0.
    multTheta = 0.
```

### 5.3.4 Logging and Diagnostics

Listing 4 illustrates the use of the diagnostics syntax in the context of the MITgcm (Artale et al., 2010). The diagnostics includes general parameters to setup the diagnostics, i.e., a storage directory for all diagnostics, a file format and mask setup for the output. For each kind of log, a separate log section must be specified which defines the log file name, here `first-out.log`, the kind of log, e.g., a snapshot or an aggregation of values, a frequency and which value should be used to indicate a missing value. Finally, the fields and levels are specified.

**Listing 4.** Configuration excerpt of diagnostics of MITgcm

```
diagnostics:
  diagMdsDir: "diagnostics/directory"
  format: netcdf
  diagSt_regMaskFile: "regMask_lat24.bin"
  - log: "first-out.log"
    logmode: snap
    frequency: 10
    missing_value: 5.0
    fields(1:2) = [ SDIAG1, SDIAG2 ]
    levels(1:2) = [ 1,2 ]
```

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

### 5.4 Type Semantics

The CP-DSL defines a set of base types: Integer, Float, String, File and Boolean. In the Declaration model, enumeration types and array types can be declared. The former are used to handle discrete selection options, e.g., file formats. Arrays can be declared based on all other types. This also allows to define multi-dimensional array types.

The CP-DSL uses an ocean-model-agnostic type system that allows CP-DSL types to be mapped to target language types. The general typing rules follow the rules laid out by Pierce (2002). Thus, we only discuss key features here:

- The numerical types Integer and Float, are automatically coerced to the next fitting type in an expression. If all values in an expression are Integer, the resulting type will be Integer. However, if one value is Float, it will be Float. The associated typing rules are as follows: Assume $R$ and $T$ are either Integer or Float, and $R \leq T$, then

$$\frac{t_{left} : R \; t_{right} : T \; operator = \{+, -, *, /\}}{t_{left} \; operator \; t_{right} : T}$$

$$\frac{t_{left} : T \; t_{right} : R \; operator = \{+, -, *, /\}}{t_{left} \; operator \; t_{right} : T}$$

- In case of array types, the size of the arrays must match or one type must be a scalar. The semantics for two arrays $A$ and $B$ with operators $+$, $-$, $*$, $/$, and $\%$ are $c_i = a_i \, op \, b_i$. When one side is a scalar, it is $c_i = a \, op \, b_i$.

- The typing of parameter references is simple. They inherit their type from the parameter declaration in the Declaration model. In case of array types, the array type can be reduced by dimension specifications. In the Declaration model, array types are all specified with the index range and the number of entries. For instance, `layers(2:5)` refers to a parameter `layers` which has 4 elements ranging from 2 to 5.

The CP-DSL allows to address a sub-range of array elements, e.g., `layers(3:4)` which will range from 3 to 4. If only one element is used, only one dimension value is specified, e.g., `layers(4)`. In this case, the type of the parameter is the array type's base type – in our example Boolean. Please note that in the CP-DSL the specified range must be within the limits of the type specified in the Declaration model.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

## 6 Implementation of CP-DSL

In the following, we explain our implementation infrastructure for agile development of CP-DSL (Subsection 6.1), and the implementation of the code generators (Subsection 6.2).

### 6.1 Implementation Infrastructure for Agile Development

As implementation infrastructure, we use XText (Bettini, 2016), Xtend (Efftinge et al., 2012), and EMF (Steinberg et al., 2009) for the following reasons:

- The CP-DSL, its metamodels and code generators should be developed with continuous feedback from users, i.e., scientists, and by adding new case studies to the project. This continually creates new feature requests that must be integrated into the DSL. Hence, the development context requires an agile approach which allows to introduce changes frequently. The selected technology stack is able to provide this, as XText automatically derives syntax tree representations and metamodels from the grammar specifiation, and includes the generator template language XTend. Thus, it reduces development and maintenance costs (cf. Challenge 1 in the introduction).
- XText supports the Language Server Protocol (LSP) (Microsoft Corporation, 2016) that allows to provide DSL-supportive editor features to many editors, including Vim, Emacs, and Jupyter. This facilitates the integration in the existing development environment of scientists (cf. Challenge 4 in the introduction).
- We have previous experience in implementing DSLs for ocean science with XText/Xtend/EMF (Johanson and Hasselbring, 2017) reducing the time necessary to familiarize the involved researchers and developers with these technologies.

While such an implementation infrastructure introduces a new dependency and therefore a risk (cf. Challenge 5 in the introduction), we mitigate this, as the generated code is human readable and in case the DSL becomes deprecated, the generated artifacts can still be used.

### 6.2 Code Generation

The relationships and code transformations among these metamodels are specified as so-called *megamodels*. The overall architecture for the code generation for MITgcm is a two step process, depicted as a megamodel in Figure 8. For the megamodel notation in Figure 8, we follow the notation of Favre et al. (2012) and Jung (2016).

We distinguish Configuration and Declaration Models, as discussed in Subsection 5.1, see Figure 2. On the left of Figure 8, the Configuration and Declaration Models for a specific ocean model (here MITgcm) are shown. The models in the middle are intermediary models that can be transformed into specific configuration and parametrization files. The first step of the process utilizes a set of model-to-model transformations to create the intermediary models. The second step uses a set of model-to-text transformations to generate the required output files. We chose this two step process, as this helps with maintaining code templates and model-to-model transformations.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

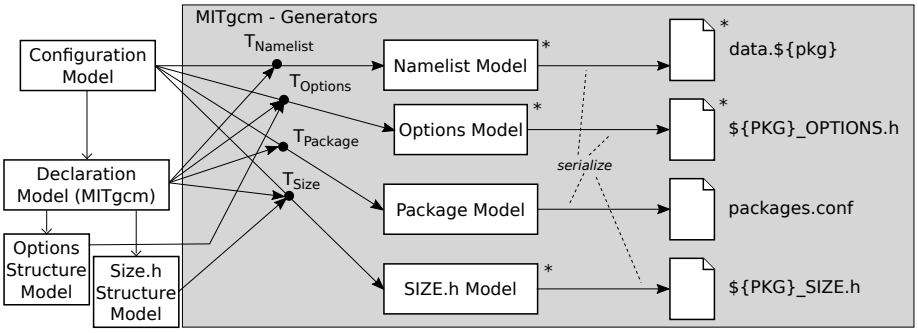


**Figure 8.** Excerpt of the megamodel of the DSL and its code generators for MITgcm; boxes represent models, arrows transformations, open arrows references between models and asterisks denote multiplicity (the notation is derived from Favre et al. (2012) and Jung (2016)).

MITgcm has four different types of configuration files and uses separate files for each module (called package in MITgcm). The transformations $T_{Namelist}$, $T_{Options}$, $T_{Packages}$ and $T_{Size}$ in Figure 8 produce output models for Fortran namelists, feature selections which can be stored in preprocessor files based `#defines` and `#undefs`, MITgcm package selection files, and MITgcm include files defining variables and sizes, respectively. As the `${PKG}_OPTIONS.h` and `${PKG}_SIZE.h` files do not follow a unique structure that can automatically be derived from parameter and feature settings alone, additional structural information is provided in multiple Options Structure Models and Size.h Structure Models (cf. Figure 8, lower left). The Declaration model contains the declaration of data types, parameters, features, and modules and refers to supplemental information, as shown in Figure 8 for the MITgcm generators.

Each ocean model has its own configuration and parametrization file types. Thus, the set of transformations and code-level models differ. Figure 8 and Figure 9 depict the transformations for MITgcm and UVic, respectively.

As depicted in Figure 9, UVic handles its configuration and parametrization in two files, where `mk.in` contains feature selections and compiler setup. `control.in` serves as runtime parametrization file in a Fortran namelist format.

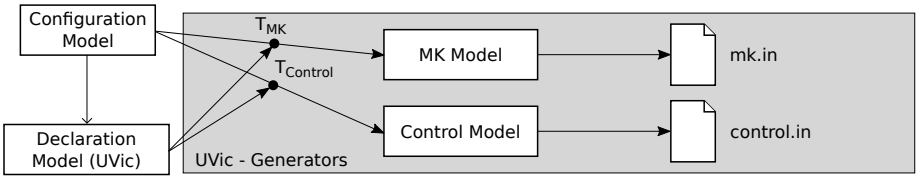


**Figure 9.** Excerpt of the megamodel of the DSL and its code generators for UVic.

Our generator architecture provides an API to integrate additional transformations for other file formats necessary to support additional ocean and climate models. We provide a template extension mechanism alongside with documentation that allows to create extensions for any scientific model with little knowledge of the overall technology.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

For model developers, who implement new features and modules for an already supported ocean model, writing new transformations can be omitted when they follow the general guidelines for their respective scientific model. They only need to add module, feature and parameter declarations to the Declaration model.

## 7 Evaluation

The CP-DSL was evaluated utilizing the JupyterLab setup as a case study for the OceanDSL Project (Jung et al., 2021c). We applied an iterative evaluation process where we evaluated CP-DSL versions via user studies and interviews with domain experts. The initial evaluation was performed with an early syntax of the DSL. The later review was conducted with domain experts based on the revised syntax, which was presented in Section 5.

### 7.1 Initial Evaluation

The first CP-DSL revision evaluation replicated three MITgcm experiments derived from the MITgcm User Manual (Alistair et al., 2018). In detail, we used the Barotropic Ocean Gyre experiment, a modified variant of the Barotropic adjustment problem based on a latitude-longitude grid, and a global ocean simulation. We tested the feasibility and usability of the DSL. First, test users were tasked to create suitable configurations for the three experiments based on their documentation. Second, they reported on the feasibility to create the desired configurations and potential issues with the DSL. The created CP-DSL files were used to generate native configuration files for each experiment and compared them to the original setup of the experiments. To gain insight into its usability, test users were asked to report on their experience in a questionnaire.

The feasibility evaluation showed that valid setup files could be generated successfully for all three scenarios. The validation was performed via a regression test against the original setup. The usability analysis is based on the feedback by the test users. While they were able to create the intended setup, they found the use of JSON-style curly braces in the first version of CP-DSL impractical, affecting readability.

### 7.2 Second Evaluation

In the second evaluation, the revised CP-DSL was presented to domain experts and utilized in a joint session to demonstrate its usage. UVic is the reference simulation by GEOMAR of the University of Victoria model. It was used together with MITgcm in the third phase of Sopran (Bange, 2016). Where UVic formed the basis for all earth system models and MITgcm was used for atmospheric and biogeochemical processes. Based on this experience, interviews were conducted to gain insights on the usability of the CP-DSL. The division in general parameters and modules was considered useful. Also the reworked YAML syntax was rated easy to understand.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021


Geoscientific Model Development Discussions

## 8 Conclusions

We present the model-independent configuration and parametrization CP-DSL for ocean models, evaluated with domain experts from ocean science. The key concepts of the DSL, e.g., feature selection, setting up parameters, and module selection are presented. CP-DSL separates the configuration and parameterization from the scientific program code and summarizes it in a unified document. We explained the general architecture of the code generation and illustrated this with the two case studies based on the earth system climate models MITgcm and UVic. The evaluation included domain experts and research software engineers. The discussion of the major requirements of the DSL itself is based on a domain analysis through interviews with domain experts and a thematic analysis of the interview results, as documented in Jung et al. (2021b). The code generation including extensibility provides model-independent CP-DSL specifications. Furthermore, support for the Language Server Protocol (LSP) enables integration into the toolchain of scientific modelers and model developers, facilitating the usability of the DSL.

As this is an ongoing research project, we aim to further extend and improve CP-DSL in close contact with users and active scientists from the domain. We initially developed the DSL for a representative subset of MITgcm ocean modeling scenarios and are currently evaluating it to be able to support all modules of MITgcm. However, the current syntax for diagnostics caused a larger comment by our domain experts, as diagnostics are an important topic in climate modeling for various purposes. A concern was raised that the current syntax for the diagnostics might either be not expressive enough or did not provide enough specific features, i.e., it seemed to be too close to the generic module setups. We will address these insights in the next revision of the DSL including a thorough investigation of different diagnostics approaches including CDI-pio (Kleberg et al., 2017) and XIOS (Meurdesoif et al., 2012).

Our current efforts also focus on supporting additional ocean models and improving the syntax and semantics of the CP-DSL. In particular, the current parameter-override semantics follows a straightforward solution where an included parameter can be redefined and thus overwritten. Since this allows accidental redefinition of parameters, we will evaluate several language solutions in the future that prevent this. For instance, the possibility to redefine or hide parameters when importing configurations and parameters. We will further evaluate the CP-DSL versions with ocean scientists and research software engineers from our partner institutions at GEOMAR, DKRZ, and the MPI.

*Code and data availability.* The CP-DSL code is available at https://git.se.informatik.uni-kiel.de/oceandsl/cp-dsl. Furthermore, a replication package is provided (Jung et al., 2021a)

*Author contributions.* Authors contributed equally to this work.

*Competing interests.* The authors declare that they have no competing interests.

*Acknowledgements.* Funded by the Deutsche Forschungsgemeinschaft (DFG – German Research Foundation), grant no. HA 2038/8-1 – 425916241.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021




## References

Adams, S. et al.: LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models, Journal of Parallel and Distributed Computing, 132, 383–396, https://doi.org/10.1016/j.jpdc.2019.02.007, 2019.

Aho, A., Lam, M. S., Sethi, R., and Ullman, J.: Compilers: Principles, Techniques and Tools, Pearson, 2nd edn., 2007.

AIRSEA Team: ARGIF – Adaptive GRid Refinement in Fortran, http://agrif.imag.fr/index.html, 2016.

Alexander, K. and Easterbrook, S. M.: The software architecture of climate models: a graphical comparison of CMIP5 and EMICAR5 configurations, Geoscientific Model Development, 8, 1221–1232, https://doi.org/10.5194/gmd-8-1221-2015, 2015.

Alistair, A., Jean-Michel, C., Stephanie, D., Constantinos, E., David, F., Gael, F., Baylor, F.-K., Patrick, H., Chris, H., Helen, H., et al.: MITgcm User Manual, Tech. rep., online, http://mitgcm.readthedocs.io/en/latest/, 2018.

Artale, V. et al.: An atmosphere–ocean regional climate model for the Mediterranean area: assessment of a present climate simulation, Climate Dynamics, 35, 721–740, https://doi.org/10.1007/s00382-009-0691-8, 2010.

Aumont, O., Ethé, C., Tagliabue, A., Bopp, L., and Gehlen, M.: PISCES-v2: an ocean biogeochemical model for carbon and ecosystem studies, Geoscientific Model Development, 8, 2465–2513, https://doi.org/10.5194/gmd-8-2465-2015, 2015.

Bange, H. W.: SOPRAN III - Abschlussbericht: 01.02.2013 - 31.07.2016, https://doi.org/10.2314/GBV:871684756, 2016.

Bettini, L.: Implementing Domain Specific Languages with Xtext and Xtend - Second Edition, Packt Publishing, 2nd edn., 2016.

Braun, V. and Clarke, V.: Using thematic analysis in psychology, Qualitative Research in Psychology, 3, 77–101, https://doi.org/10.1191/1478088706qp063oa, 2006.

Chinosi, M. and Trombetta, A.: BPMN: An introduction to the standard, Computer Standards & Interfaces, 34, 124–134, https://doi.org/10.1016/j.csi.2011.06.002, 2012.

Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., and Da Silva, A.: Design and implementation of components in the Earth System Modeling Framework, The International Journal of High Performance Computing Applications, 19, 341–350, https://doi.org/10.1177/1094342005056120, 2005.

Dagum, L. and Menon, R.: OpenMP: an industry standard API for shared-memory programming, IEEE Computational Science and Engineering, 5, 46–55, https://doi.org/10.1109/99.660313, 1998.

Efftinge, S., Eysholdt, M., Köhnlein, J., Zarnekow, S., von Massow, R., Hasselbring, W., and Hanus, M.: Xbase: implementing domain-specific languages for Java, in: Proceedings of the 11th International Conference on Generative Programming and Component Engineering (GPCE 2012), pp. 112–121, ACM, New York, NY, USA, https://doi.org/10.1145/2371401.2371419, 2012.

Favre, J.-M., Lämmel, R., and Varanovich, A.: Modeling the Linguistic Architecture of Software Products, in: Model Driven Engineering Languages and Systems, pp. 151–167, Springer, https://doi.org/10.1007/978-3-642-33666-9_11, 2012.

Goltz, U., Reussner, R. H., Goedicke, M., Hasselbring, W., Märtin, L., and Vogel-Heuser, B.: Design for future: managed software evolution, Computer Science - Research and Development, https://doi.org/10.1007/s00450-014-0273-9, 2015.

Hasselbring, W., Carr, L., Hettrick, S., Packer, H., and Tiropanis, T.: Open Source Research Software, Computer, 53, https://doi.org/10.1109/MC.2020.2998235, 2020.

Johanson, A. and Hasselbring, W.: Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment, Empirical Software Engineering, 22, 2206–2236, https://doi.org/10.1007/s10664-016-9483-z, 2017.

Johanson, A. and Hasselbring, W.: Software Engineering for Computational Science: Past, Present, Future, Computing in Science & Engineering, 20, 90–109, https://doi.org/10.1109/MCSE.2018.021651343, 2018.

https://doi.org/10.5194/gmd-2021-311
Preprint. Discussion started: 22 October 2021




Johanson, A., Oschlies, A., Hasselbring, W., and Worm, B.: SPRAT: A spatially-explicit marine ecosystem model based on population balance equations, Ecological Modelling, 349, 11–25, https://doi.org/10.1016/j.ecolmodel.2017.01.020, 2017.

Jung, R.: Generator-Composition for Aspect-Oriented Domain-Specific Languages, Doctoral thesis/phd, Faculty of Engineering, Kiel University, 2016.

Jung, R., Gundlach, S., and Hasselbring, W.: Replication Package for CP-DSL: Supporting Configuration and Parametrization of Ocean Models, https://doi.org/10.5281/zenodo.4753368, 2021a.

Jung, R., Gundlach, S., and Hasselbring, W.: Software Development Processes in Ocean System Modeling, CoRR, abs/2108.08589, http://arxiv.org/abs/2108.08589, 2021b.

Jung, R., Gundlach, S., Simonov, S., and Hasselbring, W.: Developing Domain-Specific Languages for Ocean Modeling, in: Proceedings of the 8th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS 2021), vol. 2814 of *CEUR*, http://ceur-ws.org/Vol-2814/, 2021c.

Kleberg, D., Schulzweida, U., Jahns, T., and Kornblueh, L.: CDI-pio, CDI with parallel I/O, https://code.mpimet.mpg.de/attachments/download/13746/pio_docu.pdf, 2017.

McKee, S., Nelson, N., Sarma, A., and Dig, D.: Software practitioner perspectives on merge conflicts and resolutions, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 467–478, IEEE, 2017.

MeteoSwiss: Dusk-Frontend for Dawn – Compiler Toolchain, https://github.com/dawn-ico/dusk, 2020a.

MeteoSwiss: Dawn – Compiler toolchain to enable generation of high-level DSLs for geophysical fluid dynamics models, https://github.com/MeteoSwiss-APN/dawn, 2020b.

Meurdesoif, Y., Ozdoba, H., Caubel, A., and Marti, O.: XIOS – XML-IO-Server, http://forge.ipsl.jussieu.fr/ioserver/raw-attachment/wiki/WikiStart/XIOS_IO_Workshop_Hamburg.pdf, 2012.

Microsoft Corporation: Language Server Protocol, https://microsoft.github.io/language-server-protocol/, 2016.

MPI: Modeling with ICON, https://mpimet.mpg.de/en/science/modeling-with-icon, 2020.

National Oceanography Center: GOcean, https://puma.nerc.ac.uk/trac/GOcean, 2020.

Pierce, B. C.: Types and programming languages, MIT Press, 2002.

Reussner, R., Goedicke, M., Hasselbring, W., Vogel-Heuser, B., Keim, J., and Märtin, L.: Managed Software Evolution, Springer, Cham, https://doi.org/10.1007/978-3-030-13499-0, 2019.

Schulzweida, U.: CDI - Climate Data Interface, https://code.mpimet.mpg.de/projects/cdi, 2019.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E.: EMF: Eclipse Modeling Framework, Eclipse Series, Addison-Wesley, Upper Saddle River, NJ, 2 edn., 2009.

Stevens, B. et al.: Atmospheric component of the MPI-M Earth System Model: ECHAM6, Journal of Advances in Modeling Earth Systems, 5, 146–172, https://doi.org/10.1002/jame.20015, 2013.

UML 2.5.1: OMG Unified Modeling Language Version 2.5.1, https://www.omg.org/spec/UML/, 2017.

Warner, J. C., Perlin, N., and Skyllingstad, E. D.: Using the Model Coupling Toolkit to couple earth system models, Environmental Modelling & Software, 23, 1240–1249, https://doi.org/10.1016/j.envsoft.2008.03.002, 2008.

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., and Yoshimori, M.: The UVic earth system climate model: Model description, climatology, and applications to past, present and future climates, Atmosphere-Ocean, 39, 361–428, https://doi.org/10.1080/07055900.2001.9649686, 2001.