Development of DSL for Bio-Geo-Chemical (BGC) Models

Faiz Ahmed

Master's Thesis
October 24, 2022

Dr.-Ing. Reiner Jung
Department of Computer Science
Kiel University

Advised by
Prof. Dr. Wilhelm Hasselbring

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel,

Abstract

To develop Bio-Geo-Chemical (BGC) models, scientists depend on General Purpose Language (GPL), some special tools, and technology experts. Furthermore, since it is a repetitive task, they need to adjust the implementation each time they change model specifications. So the entire development process is time-consuming and error-prone. Here, Domain Specific Language (DSL) comes into the picture; a well-designed DSL can automate the whole model development process, have a shorter turnaround, and be less prone to human error. Moreover, DSL expresses the original problem more naturally for domain and technology experts and automatically provides a graphical representation of the model. This paper gives a DSL called *Biogeochemical Domain Specific Language (BGC-DSL)* for BGC model scientists and also describes the development details. Before starting the development, we understand the domain by conducting interviews and analyzing BGC papers. Then, based on the acquired knowledge, we implement the DSL and finally evaluate by utilizing some models from papers.

Contents

Contents

Chapter 1

Introduction

BGC models are part of Ocean models (numerical models of ocean properties and their circulation) and they simulate *water columns*[1]. They simulate how abiotic (organic matter, living things, zooplankton[2], phytoplankton[3], etc.) and biotic (climate, sunlight, temperature, humidity, soil, etc.) variables interact through time and across space to determine rates of biogeochemical fluxes [Ber+20]. The models are being used practically everywhere related to Biogeochemistry [Wik22]. For example, some scientists are using it to determine biochemical key fluxes in the ocean ecosystem [KS15], and others are using it to evaluate the mitigation of greenhouse gas emissions from managed grasslands [Sán+18].

Though these models are purely domain-specific, for the time being, scientists are using General Purpose Languages (GPLs) to implement these models. Besides notable upsides of GPLs like programming flexibility, available virtually on every computer, programmer availability, and broadly applicable across domains, they have some significant downsides, like converting software requirements specifications into executable source code is complicated, error-prone, ambiguous, hard to understand for new developers and domain experts. Moreover, the BGC models use a plethora of mathematical equations which we translate into code. Though some languages like MATLAB and Fortran are suitable for mathematical equations, they are complex and do not provide domain-specific errors.

On the other hand, in Domain Specific Languages (DSLs), domain experts can write a precise specification themselves and do not depend on developers to translate any specifications to the runtime, which reduce the development time. A well-designed DSL can be much easier to program with than a traditional library. Easier in a sense that, programmer of a DSL does not need to know anything other than the domain, which enhance programmer productivity. DSLs also improve the efficiency of the overall software development process, because in DSLs domain specifications are clearly defined, these specifications become the center of communications across the whole development process and these specifications (i.e., DSL content) are also turned automatically into working code by using the *code generator*.

Moreover, Johanson and Hasselbring [JH17] conducted an online survey with embedded controlled experiments among ecologists to assess the correctness and time spent by the participants when using a DSL for ecosystem simulation specifications compared with a GPL-based solution. They conclude that:

> We observe that (1) solving tasks with the DSL, the participants' correctness point score was —depending on the task— on average 61 % up to 63 % higher than with the GPL-based solution and their average time spent per task was reduced by 31 % up to 56 %; (2) the participants subjectively find it easier to work with the DSL, and (3) more than 90 % of the

[1] https://en.wikipedia.org/wiki/Water_column
[2] https://biologydictionary.net/zooplankton/
[3] https://oceanservice.noaa.gov/facts/phyto.html

> subjects are able to carry out basic maintenance tasks concerning the infrastructure of the DSL used in our treatment, which is based on another internal DSL embedded into Java [JH17].

The domain of our thesis is *BGC models* and the target is developing DSL in this domain. To do so first we study different topics like *thematic analysis*, *programming languages*, *grammar*, *why to develop DSL*, and *steps to develop DSL*, and tools like *Eclipse*, and *Xtext* in Chapter 2.

Since we are developing an alternative to using GPLs, it is important to study those already existing DSLs that work as an alternative to GPLs. Chapter 3 discusses some DSLs like *PSyclone*, *ExaSlang*, and *Dawn*. Since mathematical equations are the core part of BGC models, in the same chapter we discuss some GPLs like *MATLAB*, *Octave*, *Wolfram Mathematica*, *R*, and *Fortran* which we normally use to express mathematical notations.

Next, we conducted several interviews with the specialists in BGC models, based on the questions in the *interview guide* (see Appendix A). In the interviews, scientists discussed a lot of things about BGC models, like *types of BGC models*, *categories of researchers*, *the kind of equations they use*, *working environment*, *tools*, and *programming languages*. Interviews were recorded and transcribed, and then used to find potential *themes* on them by using *thematic analysis*. Chapter 4 describes details about those interviews and the findings.

In the interviews, we asked the scientists for the papers and to understand the core building blocks we analyze three BGC models from those papers. The first model namely *C–N–P regulated ecosystem model* is from the paper of Kreus et al. [Kre+15], the second model is called *Optimality-based non-Redfield plankton–ecosystem model* from the paper ofPahlow et al. [Pah+20] and the third model *Modeling carbon overconsumption and forming extracellular particulate organic carbon* is from the paper of Schartau et al. [Sch+07]. In Chapter 5 we discuss the graphical and mathematical notations used in those models.

Then we start the implementation of Biogeochemical Domain Specific Language (BGC-DSL), which has two parts, one is *global* and another is the *compartment*. The *global* section is optional and contains constants common to compartments. Each *compartment* has its *state* variables, local constants, connections, and statements to update *state* variables. Chapter 6 discusses each part of the DSL implementation with examples.

Finally, in Chapter 7 we evalute the BGC-DSL by utilizing the model implementation of *C–N–P regulated ecosystem model* and *Optimality-based non-Redfield plankton–ecosystem model*. In the end, we conclude that our DSL can define BGC models directly or indirectly on the level of abstraction used in the BGC papers.

Chapter 2

Foundation

The domain of our thesis is BGC models, to know this domain we have several sources like interviews, BGC papers and source codes. To pursue our goal, first, we need to analyze those sources and then implement the DSL, based on the analyzed result. To do so, we need knowledge about *thematic analysis*, languages & grammar from *formal language theory*, DSL development procedure, and tools. In this chapter, we discuss these terms in detail.

2.1 Thematic Analysis

A *theme* caputures something important about the data in relation to the research questions and represents some level of patterned response or meaning within the data set. So, themes are strongly releated with research questions. The keyness of a theme is not necessarily dependent on quantifiable measures but rather on whether it captures something important in relation to the overall research question. Data set (interview guide, recorded interviews, projects etc.) is the primary requirement. To determine a theme, a rich description of that data set is important. Thematic analysis will provide a more detailed and nuanced account of one particular theme within the existing data set [BC06]. Ways of themetic analysis:

2.1.1 Inductive versus theoretical thematic analysis

Inductive (or *bottom up*) approach means theme strongly related to data themselves. In contrast, a theoretical thematic analysis would tend to be driven by the researchers theoretical or analytic interest in the area, and is thus more explicitly analyst driven [BC06].

2.1.2 Sementic or latent themes

With a semantic approach, the themes are identified within the explicit or surface meanings of the data, and the analyst is not looking for anything beyond what a participant has said or what has been written. In contrast, a thematic analysis at the latent level goes beyond the semantic content of the data, and starts to identify or examine the underlying ideas, assumptions, and conceptualizations - and ideologies - that are theorized as shaping or informing the semantic content of the data [BC06].

2.1.3 Epistemology: essentialist or realist versus constructionist thematic analysis

Thematic analysis can be conducted within both realist or essentialist and constructionist paradigms, although the outcome and focus will be different for each. The research epistemology guides what you can say about your data, and informs how you theorize meaning. With an essentialist or realist approach, you can theorize motivations, experience, and meaning in a straightforward way. In contrast, from a constructionist perspective, meaning and experience are socially produced and reproduced, rather than inhering within individuals [Bur15].

2.1.4 Phases of thematic analysis

Phases are not unique to all thematic analysis, some of the phases are similar to the phases of other qualitative research. The process starts when the analyst begins to notice, and look for, patterns of meaning and issues of potential interest in the data this may be during data collection. According to Mernik, Heering, and Sloane [MHS05], there are six phases of analysis. Moreover, analysis is not a linear process of simply moving from one phase to the next. Instead, it is more recursive process, where movement is back and forth as needed, throughout the phases.

Familiarizing with data It is vital to immerse oneself in the data to familiar with the depth and breadth of the content. Transcribing is the very first step to take to do further analysis. Once the data has already been, or will be, transcribed, it is important for oneself to spend more time familiarising with the data, and also check the transcripts back against the original audio recordings for accuracy [MHS05].

Generating initial codes This phase begins after generating an initial list of ideas about what is in the data and what is intersting about them. This phase then involves the production of initial codes from the data. Codes identify a feature of the data (semantic content or latent) that appears interesting to the analyst, and refer to the most basic segment, or element, of the raw data or information that can be assessed in a meaningful way regarding the phenomenon [Boy98].

Searching for themes When all data have intially coded and collated, and there is a long list of different codes that have been identified across the data set, the search for the theme is initiated. Here, one collates codes into potential themes, gathering all data relevant to each potential theme [MHS05].

Reviewing themes This phase begins when a set of candidate themes have devised and it involves the refinement of those themes. During this phase, it will become evident that some candidate themes are not really themes, while others might collapse into each other. Other themes might need to be broken down into separate themes. The outcome of this phase is, there will be a candidate *thematic map* of the data [MHS05].

Defining and naming themes This phase begins when there is a satisfactory thematic map of the data. Ongoing analysis to refine the specifics of each theme, and the overall story the analysis tells, generating clear definitions and names for each theme [MHS05].

Producing the report The final opportunity for analysis. Selection of vivid, compelling extract examples, final analysis of selected extracts, relating back of the analysis to the research question and literature, producing a scholarly report of the analysis [MHS05].

2.2 Languages and Grammars

Set of rules a DSL developer should follow while implementing a DSL. It lets us transform a program, into a syntax tree. Only programs that are syntactically valid can be transfromed in this way. The grammars we use to develop a DSL are *Context-free* [Aho+06]. There are two types of parser to parse *Context-free grammar*. *LL Parser*, it parses the input from Left to right, performing Leftmost derivation of the sentence, an *LR Parser* (Left-to-right, rightmost derivation in reverse) reads input text from left to right without backing up (this is true for most parsers), and produces a rightmost derivation in reverse [Aho+06].

There are three approaches to write a languange, depending on the kind of DSL you want to build: *textual languages*, *graphical languages*, and *projectional editors* [Tom22].

Textual languages They are the most classical languages. They are easier to support and can be used in all sort of context [Tom22]. We need special editor to use them efficiently. Xtext is most solid and user friendly solution to build textual languages. In our DSL, we are also using this approach.

Graphical languages seem approachable and frequently domain experts feel more at ease with them than with textual languages and their geeky syntaxes. Graphical languages require building specific editors to be used and they are less flexible than textual languages. Also, they are less frequently used than textual languages and the tools to build graphical languages tend to be less mature and more clunky [Tom22].

Projectional editors A projectional is an editor that show a projection of the content stored on file. The user interacts with such projection and the editor translates those interactions to changes to the persisted model [Tom22]. They are extremely powerful and exciting but they are unfamiliar to many users.

2.3 Developing Domain Specific Language (DSL)

Domain Specific Languages (DSLs) trade generality for expressiveness in a limited domain. By providing notations and constructs tailored toward a particular application domain, they offer substantial gains in expressiveness and ease of use compared with DSLs for the domain in question, with corresponding gains in productivity and reduced maintenance costs. Also, by reducing the amount of domain and programming expertise needed, DSLs open up their application domain to a larger group of software developers compared to DSLs [MHS05].

2.3.1 Why develop and use DSLs

Using DSLs can reap a multitude of benefits. The most obvious benefit of using DSLs is that, once you have got a language and a transformation engine, your work in the particular aspect of software development covered by the DSL becomes much more efficient, simply because you don't have to do the grunt work manually. If you are generating source code from your DSL program (as opposed to interpreting it) you can use nice, domain-specific abstractions without paying any runtime overhead, because the generator, just like a compiler, can remove the abstractions and generate efficient code.

Using DSLs and an execution engine makes the application logic expressed in the DSL code independent of the target platform. Using DSLs can increase the quality of the created product: fewer bugs, better architectural conformance, increased maintainability. This is the result of the removal of (unnecessary) degrees of freedom, the avoidance of duplication in code and the automation of repetitive work [Jet22].

2.3.2 Classifying DSLs

There are two types of languages in DSL world [Man20]. *Domain Specific Language (DSL)*, the language in which a DSL is written or presented. *host language*, in which a DSL is executed or processed.

A DSL written in a distinct language and processed by another host language is called an **external** DSL. If the DSL and the host language are the same, then the DSL type is **internal** [Man20]. There are another types of DSL called **embedded**, where the DSL is defined as a library for "host" language.

2.3.3 DSL Development

According to Mernik, Heering, and Sloane [MHS05] DSL development has four distinct phases: *decision*, *analysis*, *design*, *implementation*, and *deployment*.

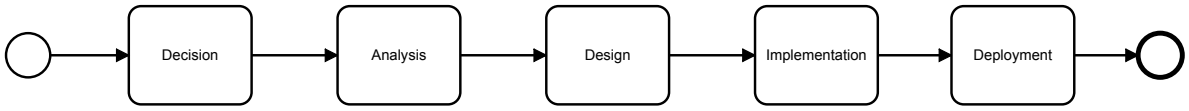

Figure 2.1. Phases of Domain Specific Language (DSL) development.

Decision Deciding in favor of a new DSL is usally not easy. It is important to think about the future use of the developed DSL and development and maintenance cost. In practice, short-term considerations and lack of expertise may easily cause indefinite postponement of the decision. Obviously, adopting an existing DSL is much less expensive and requires much less expertise than developing a new one [MHS05].

Analysis In this phase the problem domain is identified and domain knowledge is gathered. Inputs can be collected from various sources of explicit or implicit domain knowledge, such as expert interviews, existing GPL or DSL code, customer surveys, related technical papers. Most of the time domain analysis is done informally, but sometimes domain methodologies are used. The output of the domain analysis is a *domain model* consisting of

- ▻ A domain definition defining the scope of the domain [MHS05].
- ▻ Domain terminology [MHS05].
- ▻ Description of domain concepts [MHS05].
- ▻ Feature models [MHS05].

As an example in BGC-DSL we are using expert interviews and thematic analysis to gain insight in the domain, as well as, scientific and tecnical papers.

Design The easiest way to design a DSL is to base it on an existing language, Mernik, Heering, and Sloane [MHS05] identified three patterns of design based on an existing language. First, we can *piggyback* domain specific features to provide a *specialization* targeted at the problem domain. Second approchach is to take an existing language and extend it with new features that address domain concepts. Finally, design a DSL whose design bears no relationship to any existing language. In practice, development of this kind of DSL can be extremely difficult and is hard to characterize. They also distinguished between *informal* and *formal* designs. In an informal design the specification is usually in some form of natural language, probably including a set of illustrative DSL programs. A formal design consists of a specification written using one of the available semantic definition methods [Ken09].

Implementation Interpreter, Compiler/applicaiotn generator, Preprocessor, Embedding, Extensible compiler/interpreter, Commercial Off-The-Shelf (COTS) and Hybrid are the implementation patterns for executable DSLs [MHS05]. We are not yet considering this phase for our DSL, this can be a future task.

Deployment Hand over the DSL to the users. Its also an opportunity to get feedback from the users for future works.

In practice, DSL development in not a simple sequential process as shown in Figure 2.1. The decision process may be influenced by preliminary analysis which, in turn, may have to supply answers to unforeseen questions arising during design, and design is often influenced by implementation considerations [MHS05].

2.4 Tools and Frameworks

While the design is independent of a specific technology, for the implementation, we rely on certain tools and frameworks. In this section, we describe the frameworks and tools related with DSL development. *Eclipse* and *Xtext* we are using directly but EMF is used by Xtext internally.

2.4.1 Eclipse

Eclipse is an integrated development environment (IDE) used in computer programming. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Rust, Scala, and Scheme. It can also be used to develop documents with LaTeX (via a TeXlipse plug-in) and packages for the software Mathematica. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others [Ecl22].

2.4.2 Eclipse Modeling Framework (EMF)

The EMF is a open source framework that transforms models into efficient, correct, and easily customizable Java code. It focuses on class diagram subset of UML modeling and also provides the infastructure to use models effectively in our code. EMF is originally based on Meta Object Facility (MOF) from Object Management Group (OMG). To avoid confusion, the MOF-like meta model in EMF is called **Ecore** instead of MOF.

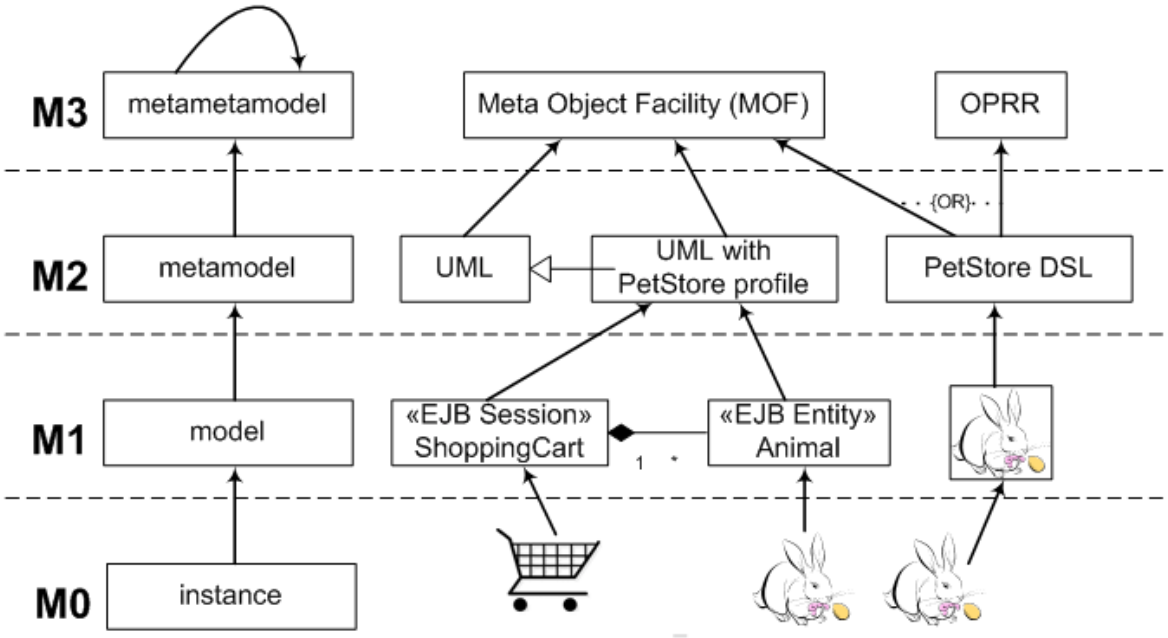

Figure 2.2. The four-layer metamodel as proposed by the Object Management Group (OMG) shows the relation between different levels [SØ22]

Figure 2.2 shows MOF has three levels. In M3 level metametamodels are used to define metamodels (M2 level) such as UML, UML profile and regular acrshortpldsl. The M1 level conaints model elements such as concrete UML classes of DSL (the boxed rabbit). Finaylly, the M0 level contains instances of M1 level constructs [SØ22]. EMF converts our models to **Ecore**. Figure 2.3 shows the **Ecore** meta-model components hierarchy.

2.4.3 Xtext

Xtext is a framework for development of programming languages and domain-specific languages [Xte22]. It works as a powerfull editor to write the grammars for a language. Xtext leverages the powerful ANTLR parser which implements an LL(*) algorithm [**xtextLL**]. **Ecore** meta-model is also auto generated by Xtext from the grammars. Xtext use EMF to generate the Java code from the **Ecore** meta model.

It also supports all the features that you'd expect from other mature IDEs, such as on-the-fly validation and error indication, syntax coloring, content assist, and code navigation. You can find

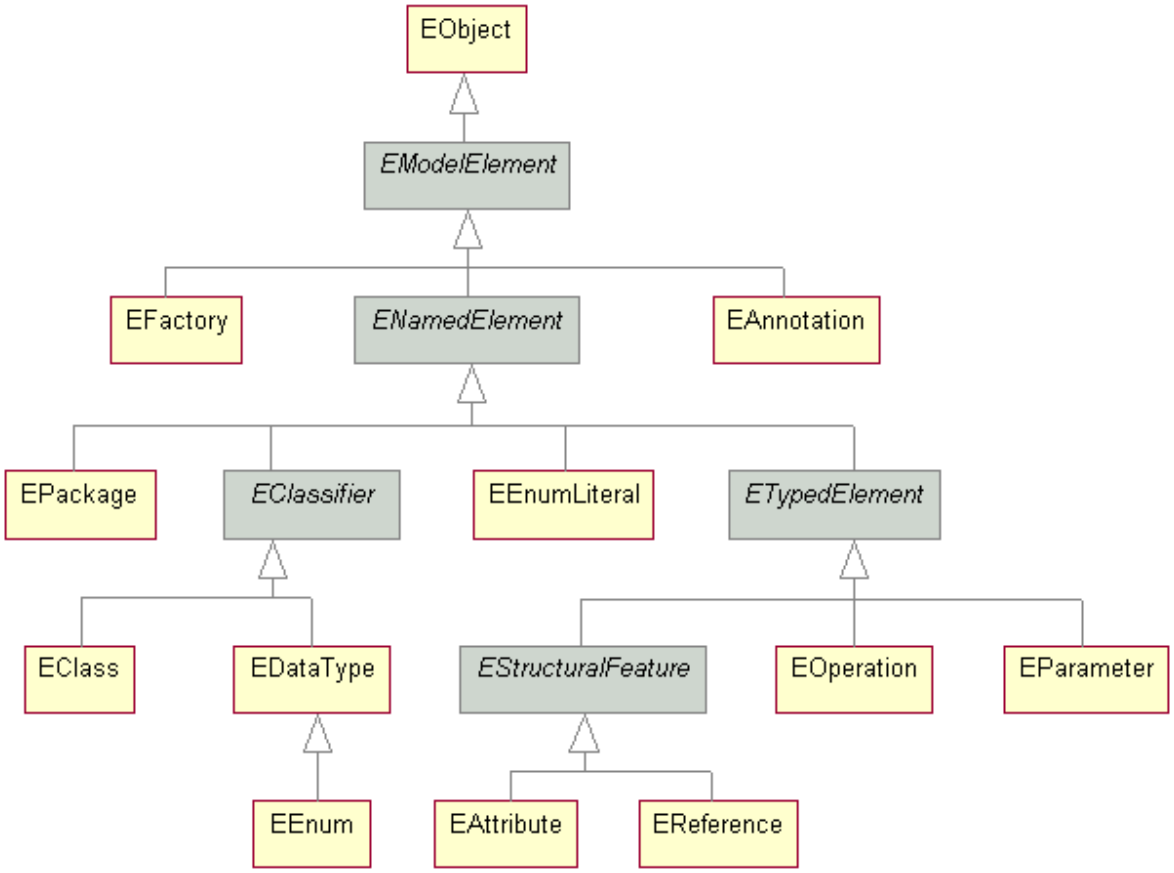

Figure 2.3. The Ecore meta-model components are related according to this hierarchy [Eco22].

references to your semantic elements, look them up in a global index, and share concepts across different languages [Xte22].

Chapter 3

Related Work

In almost every sector, scientists are using General Purpose Languages (GPLs) to implement any scientific phenomena from the beginning of the invention of programming languages. But in recent years DSLs became very popular in the modeling community to describe the structure and sometimes the behavior of a model.

In this chapter, we will discuss some DSLs which are being used practically in place of GPLs for scientific modeling. Since in Biogeochemical Domain Specific Language (BGC-DSL) we use mathematical notations, we will also review some tools developers use to implement mathematical equations.

3.1 DSLs in Scientific Modeling

Implementing models are the key aspects of any scientific discipline. A scientific model represents an empirical objects, phenomena, and physical processes in a logical way. According to John von Neumann:

> ... the sciences do not try to explain, they hardly even try to interpret, they mainly make models. By a model is meant a mathematical construct which, with the addition of certain verbal interpretations, describes observed phenomena. The justification of such a mathematical construct is solely and precisely that it is expected to work—that is, correctly to describe phenomena from a reasonably wide area [VT63].

In this section we will review some DSLs of specific domains. *PSyclone* eases weather and climate forecasting, *ExaStencil* is a code generation framework for stencil codes and *Dawn* is an optimizer and code generation library for *geophysical fluid dynamics models*.

3.1.1 PSyclone

PSyclone, a software framework that automatically generates the parts of the code necessary to run on supercomputers. PSyclone was developed for the UK Met Office and is now a part of the build system for Dynamo, the dynamical core currently in development for the Met Office's 'next generation' weather and climate model software.

PSyclone is an **embedded** DSL written in Python and host language is Fortran. PSyclone is also being extended to support an API being developed in the GOcean project for two finite difference ocean model benchmarks, one of which is based on the NEMO [Nem14] ocean model [PSy22].

3.1.2 ExaStencils and ExaSlang

ExaStencils is a code generation framework. It consists of two part, a source-to-source compiler written in Scala and a multi-layered DSL called **ExaSlang** (short for ExaStencils language) tailored for stencil codes in general and multigrid solvers in particular.

Present-day stencil codes are implemented in General Purpose Languages (GPLs), such as Fortran, C, Java, Python etc. Project ExaStencils pursued a domain-specific approach with a language, called ExaSlang [Tei+14].

ExaSlang is an **external** DSL stratified into four layers, each layer targets a certain user community and has a different degree of abstraction. This way, domain experts can formulate problems in a manner they are most familiar with, resulting in a separation of concerns and improved productivity.

With a rising layer number the DSL becomes more concrete, i.e. Layer 1 is the most abstract. In total, four layers exist:

ExaSlang 1 Continuous formulation of the problem. LaTeX-like syntax including specifications with Unicode symbols. Envisioned for users with only little interest in programming and numerical components.

ExaSlang 2 Discretization of the problem using Finite Differences (FD), Finite Volumes (FV) or Finite Elements (FE). Mostly used by domain experts for a certain application field, e.g. CFD.

ExaSlang 3 Definition of numerical solvers in a Matlab-like syntax. Different multigrid variants can be set up easily. Target users are (applied) mathematicians.

ExaSlang 4 Composition of whole program specifications. Data structures, parallelization schemes, data I/O and visualization are available for fine-tuning. Most frequently used by computer scientists.

In contrast, OceanDSLs consist three views [Kie22b]. Our BGC-DSL works in first view along with Transport-DSL.

3.1.3 Dawn and GTClang

Dawn is developed by MeteoSwiss, CSCS, ETHZ, and Vulcan allows the user to generate fast-performing code for several back-ends from the Stencil Intermediate Representation (SIR) for geophysical fluid dynamics models and GTClang is a DSL frontend using this toolchain. GTClang first translates the custom easy-to-understand language into a relatively simple Stencil Intermediate Representation (SIR). Dawn takes this Stencil Intermediate Representation (SIR), performs an array of optimizations, and subsequently generates code suitable for execution on different computing platforms [Swi22].

Developing Geophysical Fluid Dynamics (GFD) models in GPLs is time consuming and requires expert knowledge in high-performance computing. Using Dawn as a common compiler infrastructure can drastically reduce development and maintenance efforts, while increasing the performance of the generated code, for new and existing DSLs in the GFD model domain.

3.1.4 The Sprat Marine Ecosystem DSL

The Sprat Ecosystem DSL is an *external* DSL that allows to specify ecosystem simulations (with a focus on marine systems) in a declarative way. Specifically, complex 3-D ecosystem models of the HPC community based on PDE are targeted. A simulation description in the Sprat Ecosystem DSL consists of several top-level entities (Ecosystem, Output, Input, Species) that possess properties which describe the entity. Most of these properties have a constant numerical value given by an expression with a unit [JH17].

The Sprat Ecosystem DSL is similar to our DSL in the sense that the domain of this DSL focuses on the marine ecosystem, whereas our DSL domain is Bio-Geo-Chemical (BGC) models.

3.2 Mathematical Notations

Various programming languages are being used to implement mathametical formulas. Some popular languages are Wolfram Mathematica, MATLAB, Octave etc. They also have GUI mode and also very smart in generating graphics. Our DSL does not support graphical implementation becasue BGC models do not require graphics for their mathametical formulas. It can be a future task.

3.2.1 MATLAB

MATLAB combines a desktop environment tuned for iterative analysis and design processes with a programming language that expresses matrix and array mathematics directly. It includes the Live Editor for creating scripts that combine code, output, and formatted text in an executable notebook [Mat22].

3.2.2 Octave

GNU Octave is Scientific Programming Language. The Octave syntax is largely compatible with Matlab. The Octave interpreter can be run in GUI mode, as a console, or invoked as part of a shell script [Oct22].

3.2.3 Wolfram Mathematica

Worlfram Mathematica is a system for modern technical computing. For three decades, Mathematica has defined the state of the art in technical computing—and provided the principal computation environment for millions of innovators, educators, students and others around the world [Wol22].

3.2.4 R

R is a programming language for statistical computing and graphics supported by the R Core Team and the R Foundation for Statistical Computing. According to user surveys and studies of scholarly

literature databases, R is one of the most commonly used programming languages used in *data mining* [r-p22].

3.2.5 Fortran

Fortran is a general-purpose, compiled imperative programming language that is especially suited to numeric computation and scientific computing. Fortran was originally developed by IBM in the 1950s for scientific and engineering applications, and subsequently came to dominate scientific computing [for22].

Chapter 4

Analysis of Interviews

We need in-depth knowledge of how scientists work with and develop Bio-Geo-Chemical (BGC) models. Therefore, we use interviews, as they allow us to communicate with the potential users of the DSL. For the interviews, we chose a semi-structured approach to have guidance during the discussions and get answers to key questions. While simultaneously being able to react to new knowledge during the interview [Ada15]. Before the interviews, we prepared an interview guide (see Appendix A) and asked almost the same questions to all interviewees. While interviewing, we recorded those interviews and later transcribed them for analysis. This chapter discusses the findings from interviews and the captured themes from the transcribed data.

4.1 Interviews

We conducted two semi-sturctured [Ada15] interviews utilizing an interview guide (see Appendix A). The interviewees were specialists in BGC model development. The first interview was with a scientist from GEOMAR, Germany. From this interview, we got concepts like the steps of the development process for BGC models, how they establish the formulas or equations, why they prefer differential equations over chemicals, and how BGC model researchers work together in a group, etc. The second interview was with a group of scientists from the University of Kiel, Germany. This interview gave exciting information about the working environment, types of BGC models, tools, categories of researchers, programming languages, and operating environments (HPC or standard desktop). For further analysis, we recorded both interviews and later transcribed them. We use *thematic anylysis* (see Section 2.1 on page 3) to retrieve themes from the transcribed data by using the tool called QualCoder [Qua22].

In the following, we discuss the findings and themes from the conducted interviews in detail.

4.1.1 Types of BGC models

BGC model developers classify their models based on scale, time, feature and parameter.

Static and Dynamic models Based on time we have *static* and *dynamic* models, a *static* simulation model, represents a system at particular point in time. A *dynamic* simulation model represents systems as they change over time [Xia02].

Global or Local models Based on the scale of a model, there are *global* and *local* models; in *global* biogeochemical modeling, researchers use a global scale for any parameters, and in *local* modeling, the scale of any parameter depends on the surroundings of that model.

1D or 3D models Considering number of prominent parameters, we sub-divide the BGC models as *1D* and *3D*. *1D* models focus on the vertical water column, whereas *3D* models depend on the transport matrix approach. One of the great example of *3D* model is *Metos3D* [PS16] and *OPPLA*[1] uses *1D* approach.

We can also have *3D global or local* model, *1D or 3D static or dynamic* models based on combining different variables.

4.1.2 Categories of BGC model researchers

There are two categories of model researchers, where one group of researchers depends on a global scale other depends on the local scale. One of the scientist said in our interview:

> There are two categories of researchers who apply biogeochemical models.

Category one Involves researchers whose focus is on global biogeochemical modeling. They typically look at the worldwide scale and start developing the model based on existing model formulations or do some refinements on existing model formulations. These modelers are constrained more or less by the available and extensive code and model architecture they face.

Category two They focus on plankton dynamics and biogeochemistry, particularly resolving differences between carbon flux, nitrogen, and phosphorus. Here researchers try to determine the carbon-nitrogen-phosphorus flux on a global scale. The researchers of Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1) are an excellent example of this category.

4.1.3 Establishment of equations or formulas

BGC models are based on fundamental equations or formulas, but the question is how the researchers establish those equations for the model? According to the researchers of our interview, one way is by using data from experiments or observations in nature. The data is analyzed and based on the analysis a plausible or even exact formulation of the process is derived. Another way can be by refining existing ordinary differential equations from other models.

4.1.4 Ordinary differential equations or chemical formulas

Since we are focusing on BGC models, it looks self-evident that using chemical formulas would be a perfect approach to specify models. Still, this is insufficient in reality, as some substances and elements are more complex, e.g., Plankton. So using mathematical equations over chemicals while developing a BGC model is the best approach. One of the scientists said in our interview:

> We are dealing with biology, that would be perfect if we could write down everything just in terms of chemical formulas. That would be great. But unfortunately, we cannot do this.

[1] https://git.geomar.de/markus-pahlow/oppla

> We don't have first principles for an ecosystem or for biological growth. I mean, we do apply first principles, for instance, impose optimal allocation of carbon and nitrogen and phosphorus. So we can impose that, but in the end, we have no first principle other than mass conservation.

So it is important to focus on mathematical eqution over chemical formulas.

4.1.5 Approaches to decide initial equations to develop a BGC model

There is no unique way but initial phase is important for any BGC model development because in this phase developer needs to finalize model equations of formulas. To do this they use one of the approaches from below.

Experimental data To decide the initial equations, one group of developers analyzes *experimental data*. Based on the analysis, plausible or even exact equations are derived.

Refining existing differential equations In this approach, developers focus on refining differential equations from already developed BGC models and derive expected equations from there.

Trial and error Here developer tries any random approach and verify with the result. If the result does not match, try a different approach, and this process continues until they find the expected result. One researcher describes in our interview how *trial and error* works for the model *Particles sinking in the ocean*. At the beginning of the model development, they were unsure how to describe how particles sink. Do they sink together or alone? The sinking process also can vary from one place to another. Some models use different approaches. Some model analyzers say they sink with a constant speed; others say no, increasing their sinking speed with depth. Our interviewer used a different approach; he looked at the *density of the particle and calculated a sinking rate depending on the thickness*. Some people look at the size of the particles and how they stick together and then are torn apart again.

In summary, since equations are a vital part of a BGC model, it is wise to try multiple approaches to finalize the initial model equations.

4.1.6 How BGC model researchers work together in a group?

Like normal software development, BGC model development has steps. Common steps are *design*, *implementation*, *testing*, *documentation* etc. Technically skilled personal do the implementations; for instance, in R or Fortran, some do the testing, and others write their output and research results into paper.

4.1.7 Working environment

We have two environment to work *local* or *remote HPC*.

A local environment means personal computers or other low resourceful and less secure devices.

The remote environment has high-performing computing power and is highly secure.

Most of the researcher prefers *local environment* or dedicated machines to design and implement BGC models and high performance computing (HPC) to run model simulations. Sometimes researchers use the NEC HPC system (alias nesh), which provides a high computing power in the form of a combination of a scalar NEC HPC Linux Cluster including GPUs and an NEC SX-Aurora TSUBASA vector system [Kie22a].

4.1.8 Tools and languages

The interviewed scientists use *Aquamacs*, an Emacs port for Apple Macintosh computers, as the text editor. They use Emacs because it is nice for LaTeX and beamer presentations and editing the code. Some developers prefer *Vi* as the text editor, and some also use *Jupyter* notebook for data analysis. Some researcher also use *PyFerret* to load a *NetCDF* file with a global map of some data.

Scientists used *MATLAB* for equation implementations. However, nowadays, they prefer *R*, as *R* is considered more powerful and is less hassle with the licenses.

4.2 Themes

A *theme* captures something important about the data in relation to the research question, and represents some level of *patterned* response or meaning within the data set. There are a number of instances of the theme across the data set [BC06]. We looked at transcribed data for themes and use a *themetic analasis*, discussed in Chapter 2, to identify relevant themes.

In this section, we discuss themes identified from the interview data, namely: *Steps to develop a BGC model*, *Modeling with mathametical or chemical formula*, *Tools for BGC models development* and *Types of BGC models*.

4.2.1 Steps to develop a BGC model

This theme covers the development process of BGC models and explains the general development steps from modeling over implementation to testing and deployment. Figure 4.1 on page 20 depicts the process modelled in business process model and notation (BPMN). The development process consists of two phases.

The first phase starts with set of equations, implements those equations with *MATLAB*, and then tests and assesses those equations with experimental data. This step is repeated until the test is successful. The second and final phase begins with the implemented model from the previous phase, implement them in a programming language, such as *R* and *Fortran*, as a part of the model, and tests the implemented model with experimental data if success deploys the model or repeat this phase. In the following we discuss each step of the process separately:

Establishing an equation is the first and most crucial step in developing a BGC model. Equations can be

mathematical or chemical and can be extracted from the *refinement of existing differential equations* or *experimental data*.

Implementation The equations derived from the previous step are implemented in a programming language, like *MATLAB* or *R*, for prototyping.

Test those implemented equations and assess them against the experimental data. If the test and assessment pass, follow the next step; otherwise, implement them again from the learning of this step.

Reimplement Scientists established tested equations for the BGC model from previous steps. In this step, they re-implement them in *R* or *Fortran* to handle complex scenarios and implement the model's dependencies. After this step, a complete BGC model is developed without a test.

Test the model In this step, the scientists test the overall model behavior and assess the model with experimental data. If the model does not pass all tests and assessments, the scientists must modify their implementation and test until successful.

Deploy the model Deployment means handing over the model to its user or hosting it somewhere so that the user can use it. Deployment is not a part of BGC model development, but it is essential to make a model successful. Another important thing we must remember about the maintenance later on. BGC model deployment process can be done by following the steps below [Dep22]:

- ▻ Transfer code on machine (PC, HPC, mainframe, workstation), transfer is done by SCP, SFTP, clone, url download. Often it is a tar-archive, which is then extracted.
- ▻ The scientist configures the code, e.g., selects options and features.
- ▻ The scientists compiles the code.
- ▻ The scientist tests with a test set of parameters whether the program works (optional).
- ▻ The scientists applies the intended set of parameters, and downloads additional data, if necessary.
- ▻ The model is run.

4.2.2 Tools for BGC models development

From the interviews, we found several tools researchers are using to develop the models. Some tools are essential for modeling; some are text editors. In the following, we discuss those tools:

Emacs is a family of text editors that are characterized by their extensibility. The manual for the most widely used variant, GNU Emacs, describes it as "the extensible, customizable, self-documenting, real-time display editor" [Ema22]. BGC model developer as a LaTex, beamer presentations and also get away from all the Microsoft PowerPoint and Word.

Vi is a screen-oriented text editor originally created for the Unix operating system. The portable subset of the behavior of emphvi and programs based on it, and the ex editor language supported within these programs, is described by the Single Unix Specification and POSIX [vi22]. Nowadays BGC model developers not preferring *vi*, because the have to use emphvi editor over and over again to really get used to with it.

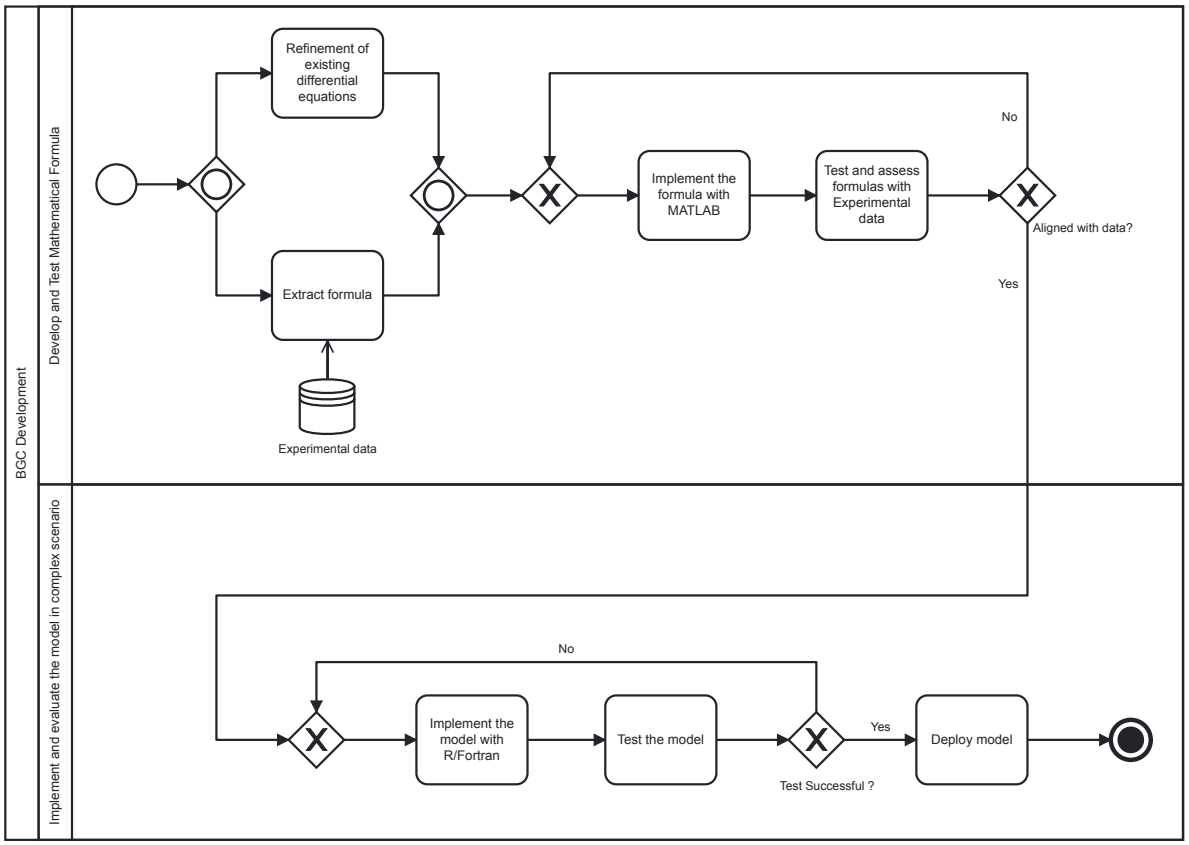

Figure 4.1. Process used to design and develop a BGC model

Metos3D is a modular software framework for the offline simulation of steady cycels of 3D marine ecosystem models based on the transport matrix approach. It is intended for parameter optimization and model assessment experiments. The simulation package has been tested with all six models. The Newton method converged for four models when using standard settings, and for two more complex models after alteration of a solver parameter or the initial guess. Both methods delivered the same steady states (within a reasonable precision) on convergence for all models employed, with the Newton iteration generally operating 6 times faster [PS16].

UVic used in the SPP 1689 is an Earth system model of intermediate complexity, developed at the University of Victoria in Canada. The model consists of the following components: (1) a three-dimensional ocean model, (2) a sea ice model, (3) a terrestrial model and (4) a simple two-dimensional atmosphere model.

OPPLA[2] Optimality-based plankton-ecosystem (OPPLA) is *1D* ecosystem model with inorganic nutrients, Dissolved Organic Matter (DOM), bacteria, phytoplankton, zooplankton, detritus. It is an offline model and requires temperature, salinity, vertical mixing coefficients and other forcing data from a physical circulation model.

Regional differences can be analysed with OPPLA by employing forcing for different locations

[2]https://git.geomar.de/markus-pahlow/oppla

in the ocean. The simulations can be calibrated and validated with time-series data, which are available for the Labrador Sea, the Bermuda Atlantic Time-Series (BATS) site and several other locations, so that we can contrast low- and high-latitude locations. OPPLA has a flexible ecosystem structure for simulations with various ecosystem configurations differing in the number and types of the functional groups. For example, simulations can be done with any number (including 0) of bacteria, phytoplankton, and zooplankton compartments [Pah22].

When OPPLA focuses on the plankton dynamics and the biogeochemistry in particular, resolving differences between carbon flux, nitrogen flux, and phosphorus flux this model assume a fixed ratio. So, it just resolves, for instance, phosphorus or just resolve nitrogen, and then it calculates the carbon flux with a fixed ratio.

OPEM Optimality-based Plankton Ecosystem Model (OPEM) is an updated version of OPPLA which uses same equations that are used in OPPLA and aslo applied to UVic. Unlike OPPLA it resolves the carbon-nitrogen-phosphorus flux on global scale.

UVic-updates-opem introduces optimality-based phytoplankton and zooplankton into the UVic-ESCM (version 2.9) with variable C:N:P (:Chl) stoichiometry for phytoplankton, diazotrophs, and detritus. Code development started with and has incorporated updates provided by David Keller, Karin Kvale, and Levin Nickelsen [Pah20].

Chapter 5

Analysis of BGC Modeling Papers

From the interviews discussed in Chapter 4, we got ideas like how people work while developing any BGC model, what tools they use, how they start, and what the technical environment is. However, we need to understand the core building blocks of BGC models. This could be addressed by extensive code analysis, documentation, developer interviews, and joint code reviews. However, the code obscures the mathematical model, as the code is on a lower level of abstraction, and different concerns, like parallelization and numerics, are mixed with the actual mathematical expression. Topic-related interviews were addressed in Chapter 4, and joint code reviews are time-consuming for the scientists. Therefore, the only option is to analyze documentation primarily described in BGC modeling papers.

In this chapter, we discuss the summarized result from analyzing the papers provided by the interviewed scientists regarding BGC model development. In interviews, we asked peers for publication, and they gave us related publications with some of their public code repositories. We looked at all the papers and analyzed the chapters where the authors discussed the building blocks of BGC models.

5.1 Analysis of the Papers

The scientists selected ten papers that might be helpful for our analysis. After a preliminary inspection, we decided four of these papers for a detailed analysis. We discarded the others, as they contained only superficial information on the composition and construction of the BGC models. The first paper is from Kreus et al. [Kre+15] where we found how we can conceptually illustrate the biogeochemical processes of a model and their links to major model compartments. The same paper introduces a BGC model called *C–N–P regulated ecosystem model (CNP-REcoM)*. This model gives us a general idea of the structure of BGC models, the architecture, and what kind of mathematical equations and configuration parameters are used in a BGC model.

The second paper is from Pahlow et al. [Pah+20], which explains the implementation and behavior of a BGC model called *Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)*. This model comprises several pools (like compartments) and uses mathematical equations to communicate among pools. In the third paper, Schartau et al. [Sch+07] describes a model called *Modeling carbon overconsumption and forming extracellular particulate organic carbon*. This paper explains the model architecture with graphic and mathematical equations to show the connection between two components. Forth paper we analyzed from Kreus and Schartau [KS15] is the second part of the first paper, where scientists describe a sensitivity analysis of biochemical key fluxes (Total production (TP), Remineralization (RM), Export (EX)). This paper uses the same model (*C–N–P regulated ecosystem model (CNP-REcoM)*) architecture from the first paper but uses different mathematical equations to simulate data and sensitivity analysis.

The research papers also use some abbreviations while explaining models and mathematical equations. The Table 5.1 shows those abbreviations with their elaboration and their respective process (or `compartment`) inside of a model.

Table 5.1. Explanations of the abbreviations used in Chapter 5

Abbreviations	Explanation	Part of process
DIM	Dissolved Inorganic Matter	Dissolved Inorganic Matter
DIC	Dissolved Inorganic Carbon	Nutrients
DIN	Dissolved Inorganic Nitrogen	Nutrients
DIP	Dissolved Inorganic Phosphorus	Nutrients
DOM	Dissolved Organic Matter	Dissolved Organic Matter
DOC	Dissolved Organic Carbon	DOM
DON	Dissolved Organic Nitrogen	DOM
DOP	Dissolved Organic Phosphorus	DOM
TEP	Transparent Exopolymer Particles	Detritus
Nif	Diazotrophs	DOM
LDOM	Labile Dissolved Organic Matter	DOM
Phy	Phytoplankton	Photoautotrophs
Het	Heterotrophs	Heterotrophs
Det	Detritus	Detritus
TA	Total Alkalinity	Nutrients
dCCHO	Dissolved Polysaccharides	Photoautotrophs

We found three Bio-Geo-Chemical (BGC) models with graphical notations and mathematical equations from four papers. In the following, we explain those graphics and equations.

5.2 Graphical Notations

Every graphic we analyze is a conceptual illustration of a BGC model. The main goal of analyzing graphical notations is to understand what different parts of a BGC model look like and how they connect each other. We study two graphics from Kreus et al. [Kre+15] paper and one from Schartau et al. [Sch+07]. In the following, we explain the details of those graphical notations.

5.2.1 Conception illustration of biogeochemical process

Kreus et al. [Kre+15] devised a Bio-Geo-Chemical (BGC) dynamic model to explain the phenomena of continuously decreasing surplus Dissolved Inorganic Phosphorus (DIP) concentration in the Baltic Sea. This model depends on the variations of the elemental Carbon-to-Nitrogen-to-Phosphorus (C:N:P) ratio during distinct periods of organic matter production and remineralization. The model was set up for the Helsinki Commission (HELCOM) monitoring station BY15. Figure 5.1 depicts the significant processes and links between biological and chemical pools relevant to this site. *Nutrients*, *Photoautotrophs*, *Dissolved Organic Matter (DOM)*, *Heterotrophs*, and *Detritus* are the five processes of this

setup or *ecosystem*. To explain this setup, scientists use a BGC model called *C–N–P regulated ecosystem model (CNP-REcoM)*; Figure 5.2 shows the graphical depiction of this model.

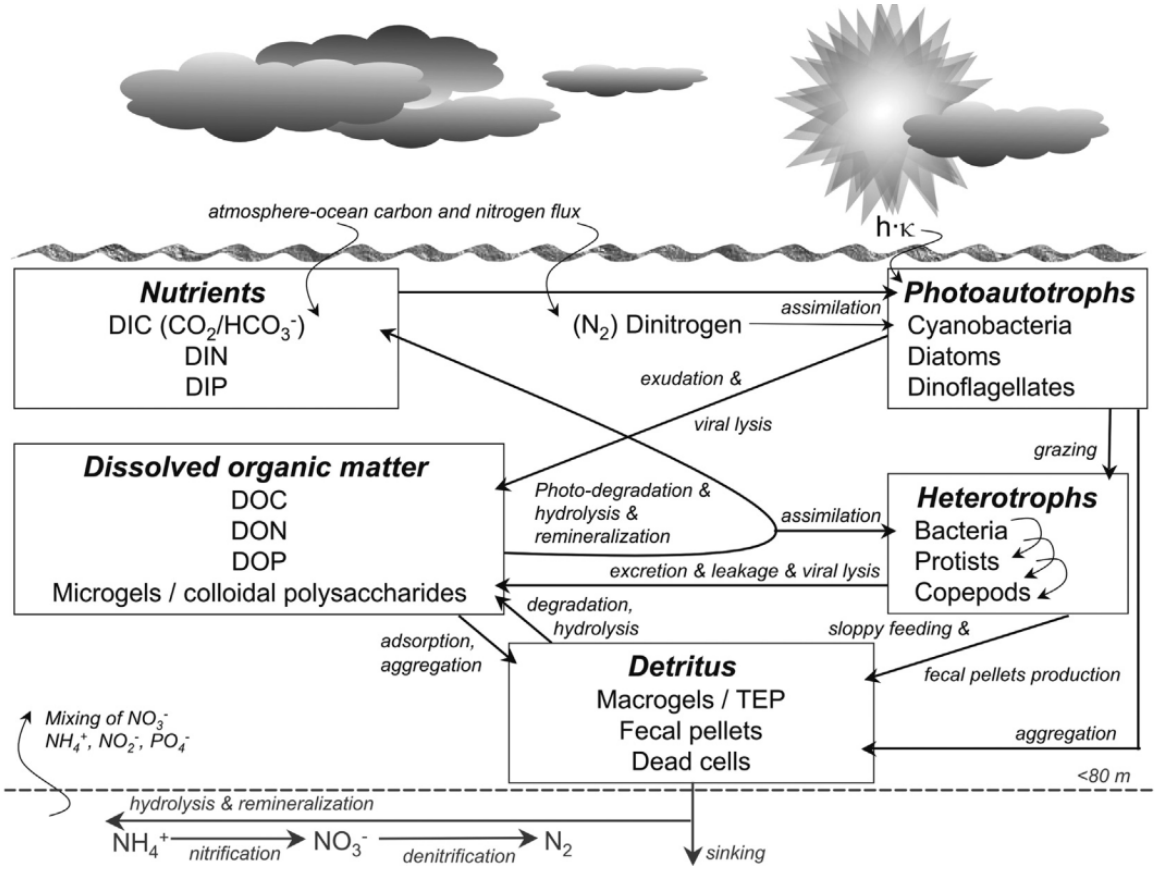

Figure 5.1. Conceptual illustration of biogeochemical processes and their links to major model compartments [Kre+15]. Abbreviations used in this figure are explained in the Table 5.1.

The model (Figure 5.2) uses six major functional groups to cover the ecosystem. The groups are expressed in C, N, and P currencies. The six functions groups are Nutrients (DIC, DIC, DIP), Phytoplankton (Phy), Diazotrophs (Nif), Heterotrophs (Het), Detritus (Det), and Labile Dissolved Organic Matter (LDOM). In this model, the `compartment` is introduced in place of components. Most model compartments are explicitly given in C, N, and P units of mass and resolve variations in C:N:P stoichiometry. Altogether with chlorophyll-a concentrations and components of the seawater carbonate system, the model has 23 biogeochemical state variables [Kre+15]. This model also uses arrows (with labels) to represent *interdependencies* between two compartments. For example, in Figure 5.2, there is an outgoing arrow with label *absorption* from compartment LDOM to MG, which represents that compartment MG is absorbing something from the compartment LDOM. Each of the arrows represents mathematical equations while implementing the model. In Section 5.3, we discuss some dependencies with respective mathematical notation.

Nutrient uptake The square shape compartment in Figure 5.2 nutrient uptake. This compartment depends on Phy and Nif by the link *respiration* and LDOM and MG by link *remineralization*, and

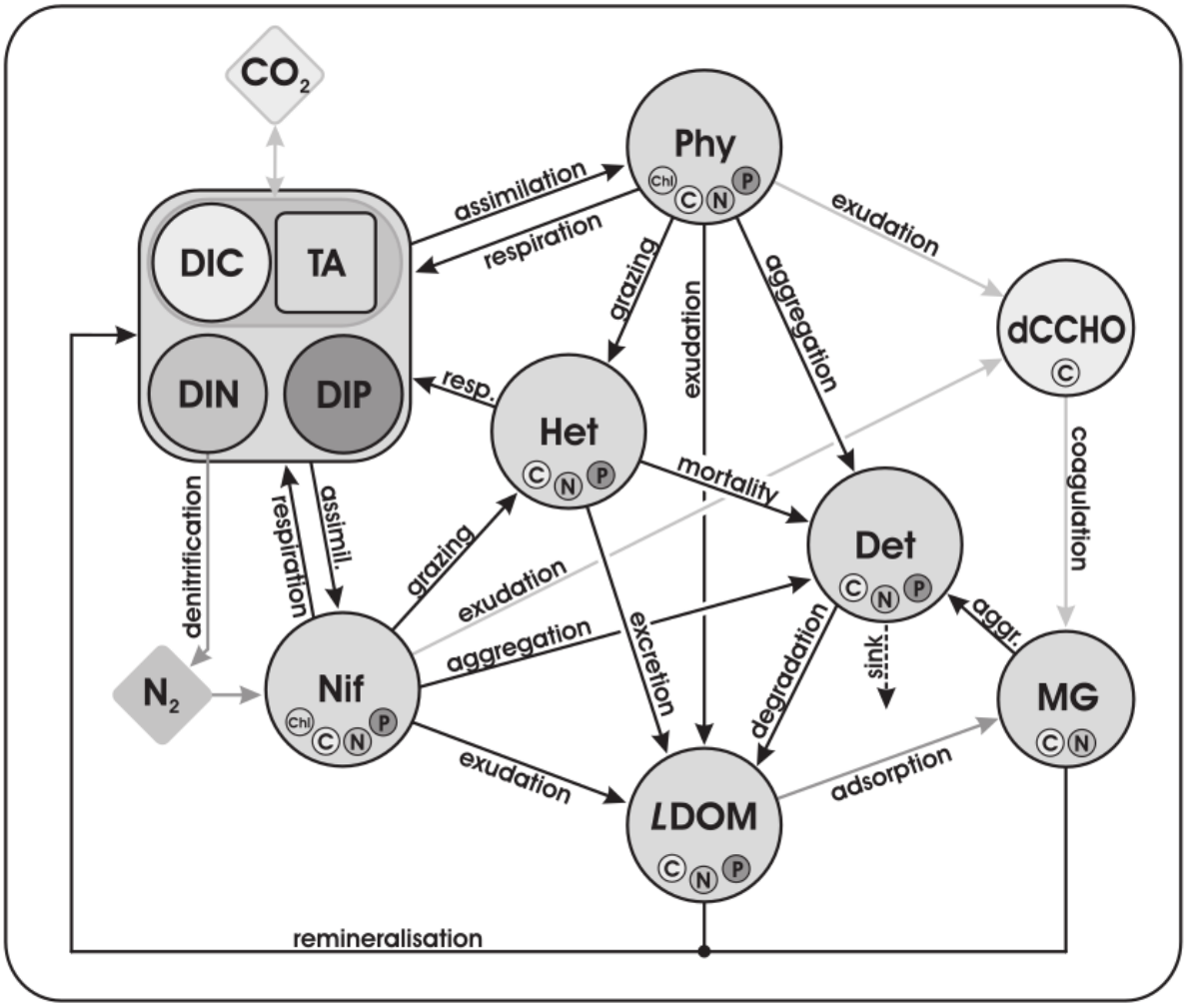

Figure 5.2. Structural diagram of the C–N–P regulated ecosystem model (CNP-REcoM). The ecosystem covers six major functional groups expressed in currencies of C, N and P, respectively, nutrients (DIC, DIN, DIP), Phytoplankton (Phy), Diazotrophs (Nif), Heterotrophs (Het), Detritus (Det) and Labile Dissolved Organic Matter (LDOM). The full set of prognostic state variables is completed by total Total Alkalinity (TA), Dissolved Polysaccharides (dCCHO), Macrogels (MG) as well as Chlorophyll-a in Phytoplankton and Diazotrophs (Dia). To budget any variation of state variables, all related processes are linked to fluxes, which are traced separately and maintaining mass balance as well [Kre+15].

this compartment is also responsible for *denitrification*.

Photoautotrophic growth The compartments Phy, dCCHO. and Nif are responsible for this process. By *assimilation* Nif and Phy receive nutrients. The link called *exudation* used by compartment dCCHO to receive chemical components from Phy and Nif.

Dissolved organic matter and macrogels This part is represented by the compartments called LDOM and MG. LDOM depends on Nif, Het, Phy, and Det. MG depends on LDOM and dCCHO. To receive chemical components from other compartments, LDOM uses the links called *exudation*, *excretion*, *degradation*, and compartment MG uses *absorption* and *coagulation*.

Model closures: detritus and heterotrophs The model compartments Het and Det depend on Phy and Nif, like the process of the setup. Het uses *grazing* to connect with Nif and Phy. Det uses *aggregation* to connnect Nif, Phy and MG.

In another paper from Kreus and Schartau [KS15], scientists describe a *sensitivity analysis* of annual mas flux estimates to that *1D ecosystem* model (Figure 5.1). Key fluxes of interest are annual (a) Total production (TP), (b) Remineralization (RM) above the halocline, and (c) Export (EX) at 50 m at the Baltic Sea monitoring site BY15 is located in the Gotland Deep basin [KS15]. This paper uses the same model (*C–N–P regulated ecosystem model (CNP-REcoM)*) to analyze the mass fluxes. Figure 5.3 shows the model compartments with arrows depicting fluxes between them.

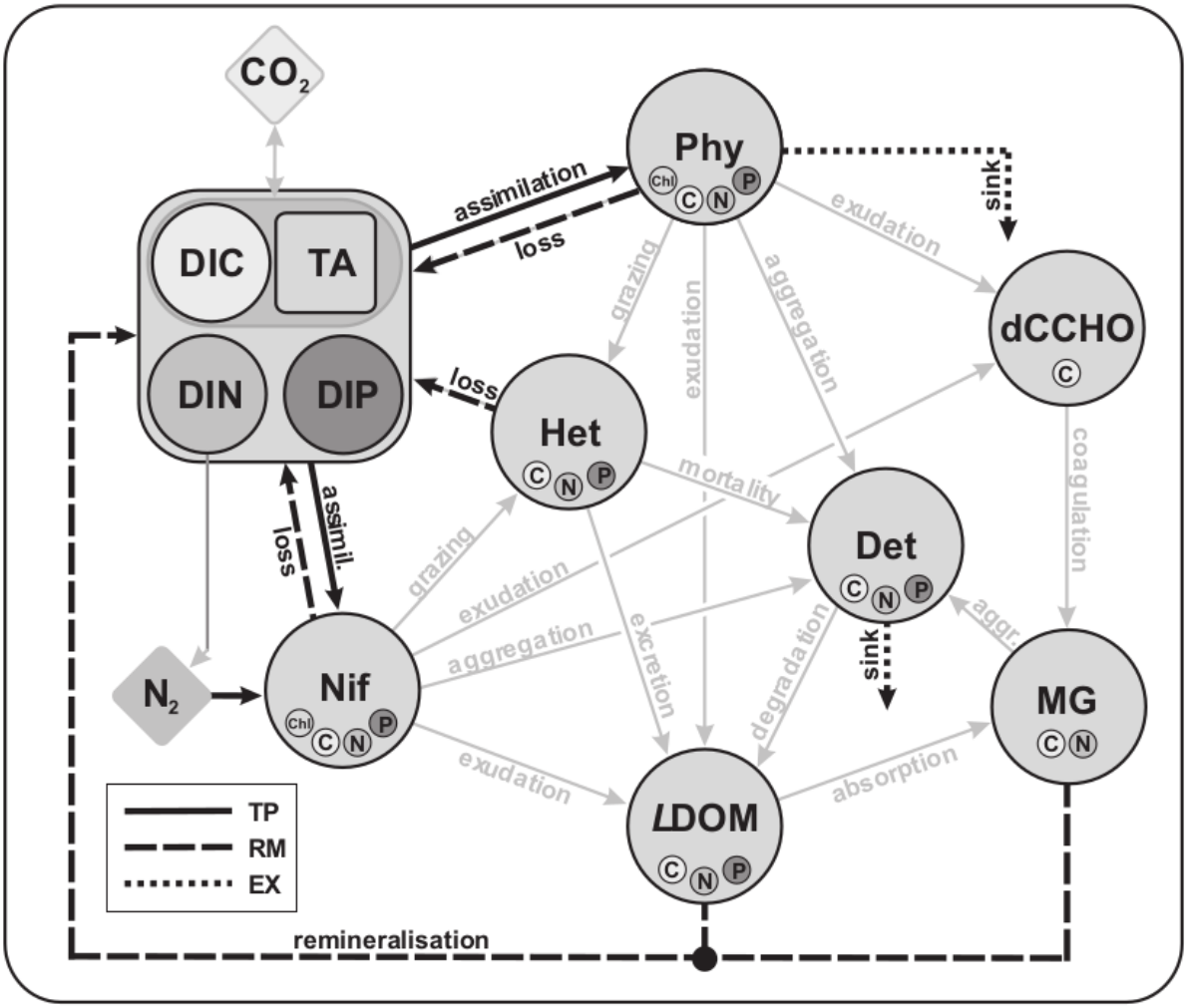

Figure 5.3. Sketch of model compartments together with arrows that depict fluxes between them. Those fluxes that are considered for the calculation of key fluxes are marked separately: Total production (TP) solid black lines; Remineralization (RM) dashed black lines; and vertical Export (EX) dotted black lines [KS15].

5.2.2 Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)

This BGC model is from the paper of Pahlow et al. [Pah+20], and Figure 5.4 is the conceptual illustration of this model. In this figure, panel (a) is the central part of this model, which depicts that ordinary Phytoplankton (Phy), Diazotrophs (Dia), Detritus (Det), and Zooplankton are the four pools of this model. This model also uses some links to connect between pools.

Phytoplankton and diazotrophs This compartment depends on DIP and DIN and is driven by optimal allocation of cellular resources.

Zooplankton This compartment directly depends on all other compartments and Zooplankton foraging depends on total activity (A_t) between foraging activity (A_f) and assimilation activity ($A_t - A_f$).

Detritus This compartment depends on all other compartments, mortality terms, and zooplankton's egestion of fecal particles to produce detritus. Which is itself subject to grazing and temperature-dependent remineralization [Pah+20].

Dissolved pools DIP and DIN are part of dissolved pools, and all the compartment depends on these pools.

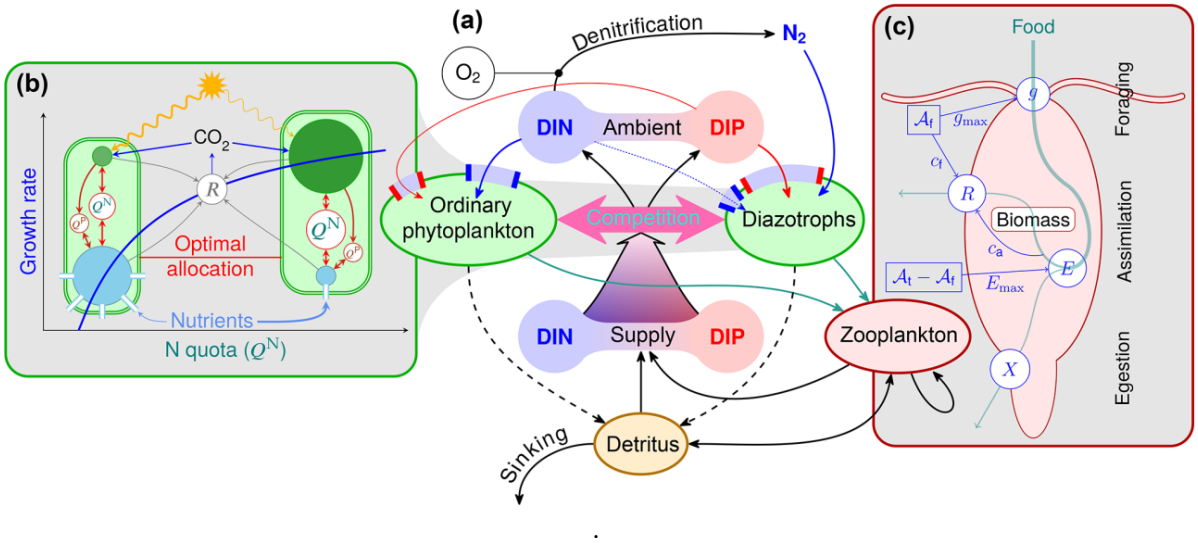

Figure 5.4. Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1) (panel **a**). Ordinary phytoplankton, diazotrophs, and zooplankton are represented by optimality-based physiological regulatory formulations. Ordinary phytoplankton and diazotrophs are driven by optimal allocation of cellular resources (b), balancing the benefits of nutrient assimilation and light harvesting against allocation and energetic costs (respiration, R) of these processes. The optimal allocation trades off, e.g. cellular N as defined by Q^N , between the requirements for photosynthesis (green) and nutrient acquisition (blue), with an additional compartment for N^2 fixation in diazotrophs (not shown). The phosphorus quota (Q^P) controls N assimilation, but only Q^N affects the growth rate directly (see Appendix C1.1). Zooplankton foraging (c) is optimised by balancing costs and benefits of allocating total activity (A_t) between foraging activity (A_f) and assimilation activity ($A_t - A_f$). Both foraging and assimilation incur energy costs (c_f and c_a , respectively) fuelled by respiration (R). Increasing ingestion (g) reduces assimilation efficiency ($E \leqslant E_{max}$), causing more particulate egestion (X) [Pah+20]

5.2.3 Modeling carbon overconsumption and the formation of extracellular particulate organic carbon

The model (*Modeling carbon overconsumption and the formation of extracellular particulate organic carbon*) setup was chosen to reproduce conditions of a *mesocosm experiment*. The mesocosm experiment was performed in a tank of 0.9 m height that was continuously stirred for 20 days [Sch+07]. This model is from the paper of Schartau et al. [Sch+07], which analyzes carbon overconsumption, combining phytoplankton growth with the Carbon content of TEP (TEPC) formation. The model describes two modes of carbon overconsumption. The first mode is associated with Dissolved Organic Carbon (DOC) exudation during phytoplankton biomass accumulation. The second mode is decoupled from algal growth but leads to a continuous rise in Particulate Organic Carbon (POC), while Particulate Organic Nitrogen (PON) remains constant [Sch+07].

Like other models discussed in previous sections, this model comprises components (or compartments) and connections. One compartment for Phytoplankton (Phy) growth, DOM, and TEPC, one compartment for DIC, DIN, and Total Alkalinity (TA), and other two compartments for Detritus and Heterotrophs. Figure 5.2 depicts the compartments with their connections to this model.

Phytoplankton growth depends on Carbon (C), Nitrogen (N) and chlorophyll *a* concentration (Chl*a*). *PhyN* and *PhyC* indicate the amount of nitrogen and carbon inside the compartment, Phy. *Assimilation* occurs from DIC, TA and DIN to this compartment.

DOM and TEPC In this part of this model, scientists account for carbon and nitrogen decoupling due to the Carbon content of TEP (TEPC) formation. The Dissolved Organic Matter (DOM) pool consists of freshly exuded, labile compounds. The labile DOM pool is split up into Polysaccharides (PCHO), Residual Dissolved Organic Carbon (resDOC). The complex process of polysaccharide *aggregation* is parameterized in terms of a two-size-class model, which describes the interaction between PCHO and TEPC [Sch+07].

DIC, DIN and TA This compartment is constrained by two *state variables*, namely TA and DIC. TA in the model varies with the DIN and phosphorus acquisition by phytoplankton and with *remineralisation* [Sch+07]. This pool connects with *Phytoplankton* and *Heterotrophs* with the link called *respiration*.

Detritus (Det) Nitrogen (N) and Carbon (C) losses are associated with cell lysis due to bacterial and viral activity and *grazing* by zooplankton. As a result of cell death, fragments of cellular material are described as a detrital compartment in the model [Sch+07]. In this model, the Detritus compartment depends on Phytoplankton compartment via the link *aggregation*.

Heterotrophs (Het) A heterotrophic `compartment` is included as a closure for modeling the Nitrogen (N) and Carbon (C) fluxes of the mesocosm experiment Schartau et al. [Sch+07]. This compartment depends on the Phytoplankton compartment by the connection called *GrazingLysis*.

5.3 Mathematical Notations

The paper analysis shows that each model has its model equations. *C–N–P regulated ecosystem model (CNP-REcoM)* (from the paper of [Kre+15]) and model *CNP-REcoM* (from paper of Pahlow et al.

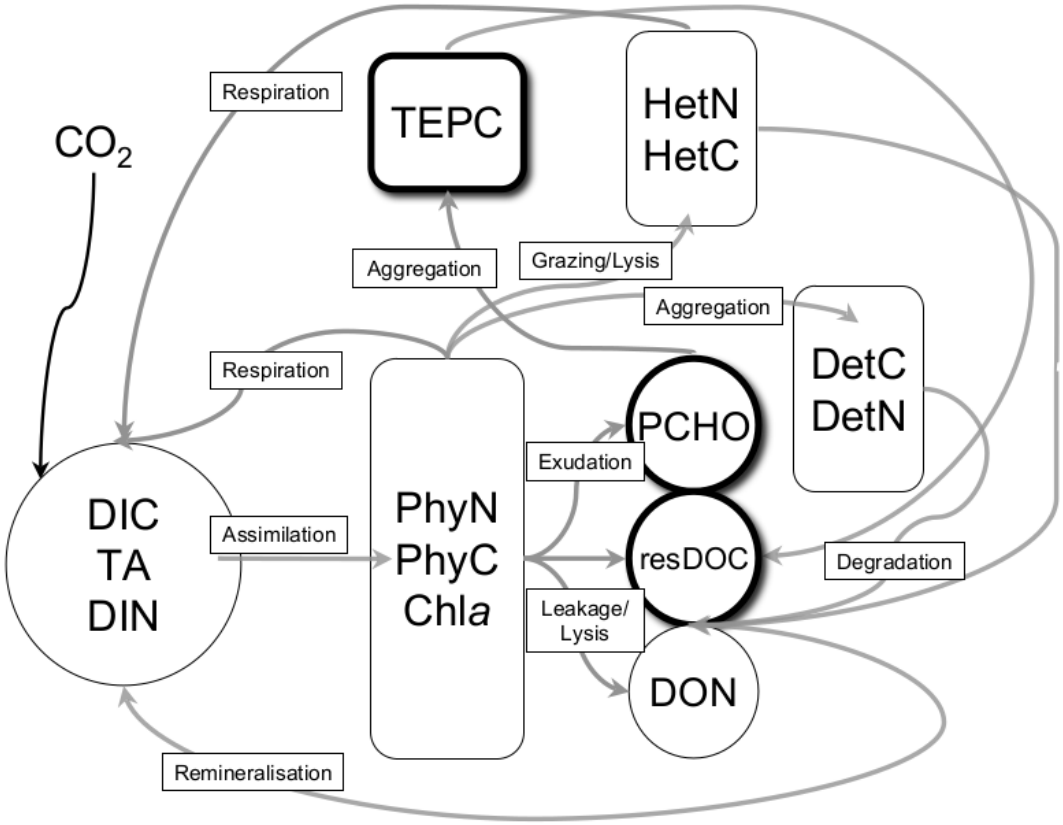

Figure 5.5. Structure of the model for simulations of nitrogen- and carbon fluxes as observed during a mesocom experiment [Sch+07]

[Pah+20]) use mathematical notations to calculate *Source-Minus-Sinks* terms and their dependencies. The model *Modeling carbon overconsumption and the formation of extracellular particulate organic carbon* from the paper of Schartau et al. [Sch+07] also uses equations to define *cost function* and to *estimate parameter values and errors*. In the following, we explain some mathematical notions from those models.

5.3.1 C–N–P regulated ecosystem model (CNP-REcoM)

As we already know from Section 5.2 that most model compartments are explicitly given in C, N, and P mass units and resolve variations in C:N:P stoichiometry. This model uses an equation called *Source Minus Sink (SMS)* to update all of the mass of every compartment. The model must resolve some dependent equations to call the SMS notation. For example, Equation 5.3.1 computes the mass C inside the `compartment` Phytoplankton (Phy). Where $resp^{\mathrm{Phy}}$, $\gamma^{\mathrm{Phy_C}}$, $aggr^{\mathrm{Phy}}$ and $graz^{\mathrm{Phy_N}}$ are dependent equations, the model needs to pre-calculate before using Equation 5.3.1.

$$\begin{aligned} \mathrm{SMS(C)} = {} & phot^{\mathrm{Phy}}_{\mathrm{DIC}} \cdot \mathrm{Phy_C} - (resp^{\mathrm{Phy}} + \gamma^{\mathrm{Phy_C}} + aggr^{\mathrm{Phy}}) \cdot \mathrm{Phy_C} \\ & - graz^{\mathrm{Phy_N}} \cdot q^{\mathrm{Phy}}_{\mathrm{C:N}} \end{aligned} \tag{5.3.1}$$

Where:
SMS (C) = Source Minus Sink (SMS)
$phot^{\mathrm{Phy}}_{\mathrm{DIC}}$ = Phytoplankton (Phy) Carbon assimilation rate (d^{-1}), it is also a calculated result from another equation
$\mathrm{Phy_C}$ = Phytoplankton (Phy) carbon, it is a model state variable
$resp^{\mathrm{Phy}}$ = Amount of Phytoplankton (Phy) loses due to respiration (resp)
$\gamma^{\mathrm{Phy_C}}$ = DOC exudation & leakage rate of Phytoplankton (Phy)
$aggr^{\mathrm{Phy}}$ = Phytoplankton (Phy) decay rate (d^{-1}) due to aggregation
$graz^{\mathrm{Phy_N}}$ = Heterotroph's grazing rate ($mmol\ N\ d^{-1}$) related with Phytoplankton
$q^{\mathrm{Phy}}_{\mathrm{C:N}}$ = Phytoplankton molar cellular C:N ratio ($mmol$ C $(mmol$ N$)^{-1}$)

5.3.2 Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)

The model Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1) (from the paper of Pahlow et al. [Pah+20]) also use the same equation to update the mass inside each compartment. In this model term, S is used in place of SMS. For example, Equation 5.3.2 calculate the mass C inside the compartments Phytoplankton (Phy) and Diazotrophs (Dia).

$$S(\mathrm{C}_p) = (\mu_p - \lambda_p - M_p) \cdot \mathrm{C}_p - G^{\mathrm{C}}_p,\ p \in \{phy, dia\} \tag{5.3.2}$$

Where:
SMS (C_p) = Source Minus Sink (SMS) of C_p.
μ_p = Net relative (C-Specific) growth rate.
λ_p = Leakage.
M_p = Mortality.
G^{C}_p = Grazing by zooplankton.

5.3.3 Modelling carbon overconsumption and the formation of extracellular particulate organic carbon

This model from the paper of Schartau et al. [Sch+07] uses mathematical notations for *data assimilation and maximum likelihood estimation of parameter values*. Equation 5.3.3 is an example of estimating parameter values and, Equation 5.3.4 is the cost function definition.

$$L(\mathbf{d}|\mathbf{p}, H, I) = \prod_{i=1}^{M} \prod_{j=1}^{N} \frac{1}{\epsilon_i \sqrt{2\pi}} \exp\left[- \frac{(m_{ij} - d_{ij})^2}{2\epsilon_i^2} \right] \tag{5.3.3}$$

Where: m_{ij} = Gaussian distribution
d_{ij} = Observations
ϵ_i^2 = Variance

$$J = \sum_{i=1}^{M} \sum_{j=1}^{N} \int_{t=0}^{T} \frac{E(t, \tau_j)}{2\sigma_i^2} ((m_i(t) - o_{ij})^2 \, dt \tag{5.3.4}$$

Where:
$\epsilon_{ij}(t)^2 = \frac{\sigma_i^2}{E(t,\tau_j)}$
Gaussian distribution, $E(t, \tau_j) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left[- \frac{(t-\tau_j)^2}{2\sigma_i^2} \right]$

5.4 Summary

The papers show that BGC models have multiple parts. Some use the term **`compartment`**, some use *pool*, and others use the *component* to separate those parts. Each **`compartment`** contains some **`state`** variables, constants, and configuration parameters. Each **`state`** variable has its initial values, and some mathematical equations are used to update those variables. Each model uses named connections (directional links) to represent the transfer of chemical elements from one compartment to another. Again these connections are represented with some mathematical notations.

Chapter 6

Design and Implementation

In Chapter 4 and Chapter 5, we have analyzed how Bio-Geo-Chemical (BGC) models are researched and developed. Our goal is to provide a Domain Specific Language (DSL) to support the development of BGC models. The DSL includes an editor, grammar, and semantic definitions for the grammar rules. A code generator for the DSL is outside the scope of this thesis.

In this chapter, we explain the basic ideas of the DSL in Section 6.1, the grammar of our DSL using an example in Section 6.2, together with semantic information on the interpretation of elements of the rules.

6.1 Concepts of the DSL

BGC-DSL comprises two parts; one is and optional `global` section and at least one `compartment`. It is not mandatory to use the `global` section because DSL may not contain any common variable used by multiple compartments. A *Compartment* contains states, constants, connections, and updates. To become a `compartment`, it is mandatory to have at least one `state` variable and at least one *update* statement to update that `state` variable inside that `compartment` (see Appendix B and C).

The terms `compartment` and `state` we found directly from the analysis of the BGC models papers (Chapter 5). The term `compartment` represents a sub-part of the BGC-DSL and the keyword `state` means a state variable. The terms `connection`, `constant`, and `update` are not terms used in the papers but describe the purpose of these elements. The term `connection` is a suitable expression for what it represents in the papers, i.e., a connection or transfer of matter from one compartment to the other. Term `constant` is fixed values used to set parameters in the model and and the keyword `update` to compute the updated amount of chemical element of a `state` variable. In the following, we discuss these terms in detail:

6.1.1 Compartment

To separate the different parts of Bio-Geo-Chemical (BGC) models, the scientists used the term *compartment* in their papers [Kre+15]. Therefore, we use the `compartment` as a keyword in DSL. For example, Figure 5.1 is an illustration of a BGC model which comprises five compartments called Nutrients, Dissolved Organic Matter (DOM), Photoautotrophs, Heterotrophs (Het), and Detritus (Det). Each compartment has its state variables. For example, the model *C–N–P regulated ecosystem model (CNP-REcoM)* depicted in Figure 5.2 has three state variables called Carbon (C), Nitrogen (N), and Phosphorus (P) inside the compartment *Phy*.

6.1.2 State

From the paper analysis, we found that each *Compartment* contains some particular chemical elements; scientists use the term as a model state variable in their papers. For example, the model C–N–P regulated ecosystem model (CNP-REcoM) depicted in Figure 5.2 on page 26 has 23 biogeochemical state variables. Each Compartment needs some initial amount of its state variables to start the model, and at the end of the model run, the states got updated using some mathematical equations. We are using the term **`state`** to represent this particular type of variable inside each Compartment, and it is mandatory to have at least one **`state`** variable inside a `Compartment`. We prefer the very beginning position of a compartment to initialize or declare the **`state`** variables because *Connections* and *Constants* may use these variables to compute some values.

6.1.3 Constant

Besides **state** variables each, each **`compartment`** uses some configuration parameters. The state of these parameters never changes during the model run. That is why we use the keyword **`constant`** to define these parameters. Like **`state`** variables, we need to initialize these configurations for each **`compartment`** at the beginning. Each **`compartment`** may have *local* and *global* **`constant`**. We initialize the global constants inside the compartments after the **`state`** variables and global constants inside the global section of our DSL. We use a special keyword **`global`** to focus this section.

To provide configuration parameters, BGC-DSL could entirely rely on the Configuration and Parametrization DSL (CP DSL) [Oce22]. However, in the context of this thesis, we do not directly use it, as it is still in development, and features may change, harming the functionality of the BGC-DSL.

6.1.4 Connection

During the model run, each **`compartment`** transfers and receives a calculated amount of **`state`** elements to or from other compartments. The calculated means, the model uses mathematical equations to measure the amount. Scientists use some special literal terms to represent this type of equation. For example, the model C–N–P regulated ecosystem model (CNP-REcoM) depicted in Figure 5.2 on page 26 has 12 types of directional equations, namely assimilation, respiration, grazing, etc. Arrow direction out means a transfer, and in means receive. Since these directional equations somehow connect two compartments, we use the term **`connection`** as a keyword in our DSL to represent this kind of equation.

6.1.5 Update

Scientists use some particular mathematical equations to update the state of the **`state`** variables. These equations depend on constants (*global* or *local*), **`state`** variables, and connections to update the particular **`state`** variable. We are using the keyword **`update`** to express these equations. The presence of at least one **`update`** statement is required to form a **`compartment`**. Since **`update`** depends on other terms, the bottom position of the compartment is fixed for the updates.

6.2 Grammar

A language developer defines the syntax of a DSL in context-free grammar (CFG) **[cfg]**. The rule language generator can automatically derive grammar from this definition [RW14]. A parser translates the DSLs' concrete syntaxes into internal representations, such as Abstract Syntax Tree (AST). We use Xtext as an editor to write the grammar and a parser that creates in-memory object graphs while consuming the grammar. Such object graphs are instances of Eclipse Modeling Framework (EMF) *Ecore* models. An Ecore model consists of an EPackage containing EClasses, EDataTypes, and EEnums and describes the structure of the instantiated objects [EMF22]. See more details about EMF in Chapter 2 on page 8.

Our grammar has a *start rule* and some production rules to express the syntaxes for *Constants*, *Compartments*, *States*, *UpdateState*, *Connections*, and *ArithmeticExpression*. In the following, we discuss the syntaxes of these rules with examples. The grammar rules are from our grammar [Ahm22], and illustrations are from the implementation of two BGC models called *C–N–P regulated ecosystem model* (see Appendix B) and *Optimality-based non-Redfield plankton–ecosystem model* (see Appendix C).

6.2.1 Start rule (BgcModel)

Every grammar requires a start rule; the parser starts parsing from this rule. It is important to begin writing the grammar with this rule. Our start rule name is *BgcModel* (Listing 6.1), which contains an unavoidable `model` name, a voluntary *global* section, and at least one compulsory **`compartment`**. The DSL needs to start with a `model` name and every model might have some global settings or configurations. We usually initialize the variables common for all compartments in the **`global`** section. The **`global`** section is optional because some models may not have common constants for all compartments, and one model can have just one **`global`** section. Moreover, the presence of at least one **`compartment`** is essential to forming a BGC model.

Listing 6.2 is the sample DSL implementation for the model CNP-REcoM with our grammar. Where, *model* name is *CNP_REcoM* and contains one **`global`** section and a sample **`compartment`** called *Phy*.

```
6  BgcModel:
7    {BgcModel}
8    'model' name=ID
9    ('global' (constant+=Constant)*)?
10   (compartment+=Compartment)+
11   ;
```

Listing 6.1. Start rule of the grammer

6.2.2 Constant

The *Constant* represents immutable variables whose value can not be changed. To initialize a constant variable, we directly assign a value or use a mathematical equation to calculate the value and then

```
model CNP_REcoM

global
  [...]

compartment Phy {
  [...]
}
```

Listing 6.2. Start rule for the DSL CNP-REcoM

set this value into that variable. This rule (Listing 6.3) starts with a keyword **constant**, then use a *PrimitiveTypes* and a name, and then uses an assignment operator ("="), and then to compute a value from the mathematical equation, we use an *ArithmeticExpression*. A **constant** can be *local* or *global*, Listing 6.4 shows two constants in the **global** section and, Listing 6.6 shows six *local* constants inside the compartment *Phy*.

```
Constant:
  'constant' type=PrimitiveTypes name=ID "=" expression=ArithmeticExpression
;

PrimitiveTypes:
  "int" | "float"
;
```

Listing 6.3. The rule *Constant* use the **constant** at the begining

```
global
  constant float T_in_K = 120 // Temperature in K , not given
  [...]
  constant float Tf = exp(-1 * Ar * (inverse(T_in_K) - inverse(T_ref)))
  [...]
```

Listing 6.4. Global variables for the DSL

6.2.3 Compartment

The *start rule* shows that a DSL requires at least one compartment. The *Compartment* rule is used to implement a single compartment of that DSL. The rule (Listing 6.5) uses the keyword **compartment** and needs a unique name. Inside two starting and ending curly braces it has three sub-rules, namely *States*, *Constant* or *Connection*, and *UpdateState*. The rule also shows that it needs at least one *State* and one *UpdateState* rules to form a *Compartment*. A sample **compartment** in Listing 6.6 shows the uses of the rules *States*, *Constant*, *Connection*, and *UpdateState* inside the *compartment Phy*.

```
Compartment:
  'compartment' name=ID '{'
    (states+=States)+
    (constants+=Constant | connections+=Connection)*
    (updateStates+=UpdateState)+
  '}'
;
```

Listing 6.5. The *Compartment* rule has three sub-rules

```
compartment Phy {
  state C = 14.16
  [...]
  constant float q_P_N_ratio = 1
  constant float W = -0.2
  constant int sampleArray = [3, 3, 4, 2] * [2, 3, 4*2, 2+2]
  [...]
  constant float Tf_Phy = exp(-1 * Ar_Phy * (inverse(T_in_K) - inverse(T_ref)))
  constant float Psi = max(0, Psi_max * (1 - (q_N_C_ratio/Q2_N_C_ratio)))
  [...]
  connection grazing_N = g_max * Tf * (N * N)/(Het.K_N * Het.K_N + (N * N + Nif.N * Nif.P)) *
       Het.N
  [...]
  update C = phot_DIC * C - (resp + gamma_C + aggr) * C - grazing_N * q_C_N_ratio
  [...]
}
```

Listing 6.6. Example of a *Compartment*

6.2.4 States

Each **compartment** of a DSL has at least one state variable, the *States* rule difine the stucture of those variables. The Listing 6.7 is the syntax of this rule, which is simply the extension of the sub-rule *State* with the keyword **state** at the beginning. Moreover, the sub-rule *State* needs a unique name and an *ArithmeticExpression* is assigned to that name. The example in Listing 6.8 uses this rule to set *14.16* (*ArithmeticExpression*) to the **state** variable *C*.

```
States:
  'state' states+=State
;

State:
  name=ID "=" expression=ArithmeticExpression;
```

Listing 6.7. The rule *States*, just an extension of the rule *State*

```
45 state C = 14.16
```

Listing 6.8. A `state` Carbon (C) with its initial value 14.16

6.2.5 UpdateState

In each `compartment`, there is a mathematical equation to compute the updated amount of each state variable at the bottom of the `compartment`, and the *UpdateState* rule is used to implement those mathematical expressions. This rule (Listing 6.9) starts with the keyword `update` and then uses the reference of an already initialized `state` variable. An *ArithmeticExpression* expression is being used to compute the current value of that `state`. In Listing 6.10, the compartment *Phy* uses this rule to evaluate the expression of the left side of the equal ("=") sign to update the value of *Carbon (C)*.

```
36 UpdateState:
37    'update' state=[State|ID] "=" expression=ArithmeticExpression
38    ;
```

Listing 6.9. Rule for *UpdateState*

```
102 update C = phot_DIC * C - (resp + gamma_C + aggr) * C - grazing_N * q_C_N_rat
```

Listing 6.10. An example for the rule UpdateState

6.2.6 Connection

The rule *Connection* is used to implement the concept described in Section 6.1.4. Each compartment can have multiple connections, but it is also optional. Listing 6.11 shows the syntax for this rule, it says we need to use the keyword `connection` before the name, and by using an *ArithmeticExpression*, we can assign a value to that connection. In the example (Listing 6.12), the compartment *Phy* uses this rule to compute the value for the connection *grazing_N*.

```
40 Connection:
41   'connection' name=ID "=" expression=ArithmeticExpression
42 ;
```

Listing 6.11. *Connection* rules, helpful to compute the transfer amount of different `state` variables between compartments

```
92 connection grazing_N = g_max * Tf * (N * N)/(Het.K_N * Het.K_N + (N * N + Nif.N * Nif.P)) *
       Het.N
```

Listing 6.12. Computing the value for the `connection` *grazing_N* inside the `compartment` Phy

6.2.7 ArithmeticExpression

The discussion above shows that almost every grammar rule uses the *ArithmeticExpression* as a sub-rule to evaluate mathematical equations. This rule's syntax (Listing 6.13) shows that it uses the recursive property to form this rule. *ArithmeticExpression* **returns** *MultiplicationExpression*, *MultiplicationExpression* returns *PowerExpression*, and *PowerExpression* returns *ValueExpression*. Since this rule works recursively, all return types for *ArithmeticExpression*, *MultiplicationExpression*, and *PowerExpression* have to be the same. In this case, the return type is *Expression*, and the *ValueExpression* works as a termination rule. This rule also uses *EAdditionOperator* **enum** to conjugate two *Expressions*. In the following, we discuss each sub-rules of *ArithmeticExpression* separately.

```
48 ArithmeticExpression returns Expression:
49   MultiplicationExpression ->({ArithmeticExpression.left=current} operator=EAdditionOperator
         right=ArithmeticExpression)?
50 ;
51
52 enum EAdditionOperator returns EAdditionOperator:
53   ADDITION = "+" |
54   SUBTRACTION = "-"
55 ;
56
57 MultiplicationExpression returns Expression:
58   PowerExpression ->({MultiplicationExpression.left=current} operator=
       EMultiplicationOperator right=MultiplicationExpression)?
59 ;
60
61 enum EMultiplicationOperator returns EMultiplicationOperator:
62   MULTIPLICATION = "*" |
63   DIVISION = "/" |
64   MODULO = "%"
65 ;
66
67 PowerExpression returns Expression:
68   ValueExpression ->({PowerExpression.left=current} "^" right=ValueExpression)?
69 ;
```

Listing 6.13. *ArithmeticExpression* with **enum** and sub-rules

MultiplicationExpression

This rule uses the **enum** *EMultiplicationOperator*, and we use the symbols "*," "/," and "%" as *EMultiplicationOperator*. This rule evaluates *PowerExpression* before *MultiplicationExpression* and finally returns an *Expression*. Listing 6.14 shows the use of this rule where we use the operator "*" as multiplication and "/" as division.

```
148 constant float phot_max = meu_max * Tf_Nif * (q_N_C_ratio - Q1_Nif_N_C_ratio)/(
        Q2_Nif_N_C_ratio - Q1_Nif_N_C_ratio)
```

Listing 6.14. Computing the value of a `constant` called *phot_max* inside the `compartment` Phy

PowerExpression

Some models may need to power expressions while implementing a mathematical equation in their DSL; the *PowerExpression* rule helps us to do so. This rule (see Listing 6.13) evaluates the *ValueExpression* first, then *PowerExpression*, and finally returns an *Expression*. In this rule, we use the symbol "^" between two expressions. In the example (Listing 6.15), `compartment` *Het* uses this rule to compute the value of the `connection` *resp*.

```
195 connection resp = Tf * r0 + tao * (max(0, 1 - (Qr_C_N_ratio/q_C_N_ratio), 1 - (Qr_C_N_ratio *
        Qr_N_P_ratio)/q_C_P_ratio)^2)
```

Listing 6.15. Use of *PowerExpression* inside the `compartment` Het

ValueExpression

ValueExpression exclusively depends on five sub-rules (see Listing 6.16) called *ArrayExpression*, *FunctionCallingExpression*, *LiteralExpression*, *ParenthesisExpression*, and classTermReference to implement *array*, *functions*, *literals*, *parenthesis*, and term reference, respectively. In the following, we discuss the syntaxes of each of the rules with examples.

```
72 ValueExpression:
73   ArrayExpression |
74   FunctionCallingExpression |
75   LiteralExpression |
76   ParenthesisExpression |
77   TermReference
78 ;
```

Listing 6.16. Value expressions and their data type rules

ArrayExpression An array is the collection of the same type of data. By this rule, we can quickly implement any *ArithmeticExpression* containing arrays. Listing 6.17 shows that we are using square brackets to represernt the collection of *ArithmeticExpression*, and the collection requires at least one *ArithmeticExpression*. This rule also supports muti-dimensional array expressions. Listing 6.18 shows the use of this rule to represent a list of *ArrayExpression* inside the `compartment` *Phy*.

FunctionCallingExpression Sometimes it is essential to call a function or method with multiple *ArithmeticExpression*s to evaluate a formula or equation. Listing 6.19 depicts the rule of *FunctionCallingExpression*, which uses *EFunction* **enum** to specify the type of function. This rule can implement three types of functions called **exp** (express exponential function), **inverse**, and **max**. Listing 6.20 shows the example of these three functions for the `compartment` *Phy*.

```
ArrayExpression:
  '[' expressions+=ArithmeticExpression (',' expressions+=ArithmeticExpression)* ']'
;
```

Listing 6.17. Rule *ArrayExpression*, useful to implements Arrays.

```
constant int sampleArray = [3, 3, 4, 2] * [2, 3, 4*2, 2+2]
```

Listing 6.18. Use of *ArrayExpression* inside the **compartment** Phy

```
FunctionCallingExpression: type=EFunction '(' expressions+= ArithmeticExpression(','
    expressions+= ArithmeticExpression)* ')';
enum EFunction:
  exp = 'exp' | inverse = 'inverse' | max = 'max'
;
```

Listing 6.19. *FunctionCallingExpression* rule contains **exp**, **inverse**, and **max** functions

```
  constant float R_N = inverse( 1 + exp(-1 * sigma_P_N * (Q2_P_N_ratio - q_P_N_ratio)))
  constant float Psi = max(0, Psi_max * (1 - (q_N_C_ratio/Q2_N_C_ratio)))
```

Listing 6.20. The **compartment** *Phy* using *FunctionCallingExpression* rule to evalute **inverse**, **exp**, and **max**

LiteralExpression The paper analysis observed that the BGC models use only the number of literals in their configuration parameters or settings. So, we need the grammar rule to implement only the numbers. Listing 6.21 shows the syntax to the *LiteralExpression*, an extension of the sub-rule *Literal*. We use a **terminal** rule called **NUMBER** to build the structure of the rule *Literal*. Listing 6.22 is the example of this rule used by the **compartment** *Phy*, where the number literal "1" is being assigned in the variable called *q_P_N_ratio*.

```
LiteralExpression:
  value=Literal
;

Literal:
  {NumberLiteral} value=NUMBER
;

terminal NUMBER returns ecore::EBigDecimal:
  ("-")?('0'..'9')* ('.' ('0'..'9')+)?;
```

Listing 6.21. Rule *LiteralExpression* is the extension of the rule *Literal*

ParenthesisExpression BGC models use parenthesis expressions to specify the mathematical equation's hierarchy of operations. In our grammar, we use the rule *ParenthesisExpression* to support this functionality while implementing the DSL. Listing 6.6 shows the syntax of this rule, using the start-

```
51 constant float q_P_N_ratio = 1
```

Listing 6.22. Use of the rule *LiteralExpression* inside the **compartment** *Phy*

ing and ending parenthesis between an *ArithmeticExpression*. Listing 6.24 used by the **compartment** *Phy*, telling that the part *(alpha * Theta_C * I)* has to be executed before applying the division (/) operation.

```
101 ParenthesisExpression:
102   '(' expression=ArithmeticExpression ')'
103 ;
```

Listing 6.23. The structure of the rule *ParenthesisExpression*

```
98 connection synth_Chl = assim_DIN * Theta2_N * phot_DIC / (alpha * Theta_C * I)
```

Listing 6.24. The **compartment** *Phy* uses the rule *ParenthesisExpression* to specify the hirerchy while computing the value for the **connection** *synth_Chl*

TermReference The BGC models use values or settings from other compartments, and they use the reference of the calling compartment. We use the rule TermReference to implement this property in our DSL. Listing 6.25 depicts the syntax of this rule, where we use the references of any implementations by following the rules *Term* or *QualifiedName*. The example in Listing 6.26 used by the **compartment** *Phy* to compute the value for the **connection** *grazing_N* uses the rule *TermReference*. The term references are *Het.K_N*, *Nif.N*, *Nif.P*, and *Het.N*, where *Het* and *Nif* are the references of different compartments.

```
105 TermReference:
106   ref=[Term | QualifiedName]
107 ;
108
109 Term:
110   name=ID
111 ;
112
113 Subterm returns Term:
114   UpdateState | Connection | State | Constant
115 ;
116
117 QualifiedName:
118   ID ('.' ID)*
119 ;
```

Listing 6.25. Syntax of the rule *TermReference*

```
92 connection grazing_N = g_max * Tf * (N * N)/(Het.K_N * Het.K_N + (N * N + Nif.N * Nif.P)) *
       Het.N
```

Listing 6.26. The `compartment` *Phy* uses the rule *TermReference* to access the value from the `compartment` *Het*

Chapter 7

Evaluation

The previous chapter discussed our implemented Domain Specific Language (DSL), and this chapter discusses the evaluation of our DSL by utilizing two Bio-Geo-Chemical (BGC) models as two case studies. *Case study 1* is for the model from the paper of Kreus et al. [Kre+15] called *C–N–P regulated ecosystem model (CNP-REcoM)*, and *Case study 2* is for the model called *Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)* from the paper of Pahlow et al. [Pah+20].

Our goal is to get the answer to the question: *Is it possible by our DSL to define models on the level of abstraction used in the BGC papers*? To do this, first, we explain the model properties from the papers and then show the respective implementation in our DSL. Remember that we are not using actual values in our evaluation and ignoring all quantities' units.

7.1 Case Study 1: C–N–P regulated ecosystem model (CNP-REcoM)

This model is from the paper of Kreus et al. [Kre+15], and as an example, we take the *Phytoplankton (Phy)* model compartment (see Appendix B). This compartment contains *global constants*, *local constants*, *state variables*, *mathematical functions*, and equations represent *connections* and update state variables. In the following, we take each part from the model paper and then show the DSL implementation.

7.1.1 Global and local constants

In Equation 7.1.1, (a) is a global constant and (b) is a local constant. As the name suggests, global constants are accessible by all compartments, and local constants are initialized inside a compartment. They can be only accessible just by this compartment and different compartments by using the reference of this compartment. Listing 7.1 is an example of the implementation of constants inside a DSL. Where constant and float are two keywords, T_{ref} is a variable, and 283.15 is the given temperature. We use the exact implementation to express *local* and *global* constants everywhere.

$$\begin{aligned} &\text{a)}\ T_{ref} = 283.15K \\ &\text{b)}\ q^{Phy}_{\text{C:N}} = 1\ \text{mmol C}\ (\text{mmol N})^{-1} \end{aligned} \tag{7.1.1}$$

Where, T_{ref} is reference temparature in Kalvin (K) and $q^{Phy}_{\text{C:N}}$ is Phytoplankton (Phy) molar cellular C:N ratio.

```
10 constant float T_ref = 283.15
```

Listing 7.1. DSL implementation of the Equation 7.1.1

7.1.2 State variables

Phy_C in Equation 7.1.2 is a state variable inside the compartment Phytoplankton (Phy). The initial amount of C inside this compartment is 14.16 mmol C m^{-3}. The first part of the variable refers compartment name, and the second part is the state variable C inside this compartment. Listing 7.2 shows the sample implementation of a **state** variable in a DSL. We use just C in place of Phy_C, because it is a state variable of this compartment.

$$Phy_C = 14.16 \text{ mmol C } m^{-3} \tag{7.1.2}$$

Where:
Phy_C = Initial amount of Carbon (C) inside **compartment** Phytoplankton (Phy).
mmol C m^{-3} = Is the unit of Carbon (C).

```
45 state C = 14.16
```

Listing 7.2. Implementation of the Equation 7.1.2

7.1.3 Mathametical functions

C–N–P regulated ecosystem model uses three different mathematical functions: The first is **inverse**, the second is to find the maximum (**max**), and the third is an exponential function (**exp**). In the example (Listing 7.3), the **max** is finding the maximum from the list of values. In Listing 7.4, **exp** is used to find the exponent value from the part of the equation and **inverse** doing the inversion.

Table 7.1 and 7.2 show the variable mappings from model to DSL. As an example, in Table 7.2, the model uses the term R_C^{Phy}, but in DSL, we use R_C to represent the same notation with the type **constant float**.

$$f_{N_2} = max\left[0, 1 - \left(\frac{\text{DIN}}{\text{DIN} + K_{\text{DIN}}} \cdot \frac{\text{DIP} + K_{\text{DIP}}}{\text{DIP}}\right)\right] \tag{7.1.3}$$

Where:
f_{N_2} = Is a function which calculate the amount of Nitrogen fixation.
max = Is a normal **max** function, which calculate maximum amount from a list of data.
DIN = Amount of Dissolved Inorganic Nitrogen (DIN) (mmol C m^{-3}), it is an elemenent of the **compartment** Dissolved Inorganic Matter (DIM).
DIP = Amount of Dissolved Inorganic Phosphorus (DIP) (mmol C m^{-3}), it is also an elemenent of the **compartment** Dissolved Inorganic Matter (DIM).
K_{DIN} = Half-saturation constant for DIN uptake (mmol N m^{-3}).

```
150  constant float f_N2 = max(0, 1 - ((DIM.DIN/DIM.DIN + DIM.K_DIN)*((DIM.DIP + DIM.K_DIP)/DIM.
         DIP)))
```

Listing 7.3. DSL implementation of the Equation 7.1.3

Table 7.1. Mapping of model to DSL for the Equation 7.1.3

Model Variable	BGC-DSL variables	BGC-DSL types
f_{N_2}	f_N2	`constant float`
max	max	`max` function
DIN	DIM.DIN	`state`(from `compartment` DIM)
DIP	DIM.DIP	`state` (from `compartment` DIM)
K_{DIN}	DIM.K_DIN	`constant float`

$$R_{\mathrm{C}}^{\mathrm{Phy}} = (1 + \exp[-\sigma_{\mathrm{C}}^{\mathrm{N}} \cdot (\mathrm{Q}2_{\mathrm{N:C}}^{\mathrm{Phy}} - q_{\mathrm{N:C}}^{\mathrm{Phy}})])^{-1} \tag{7.1.4}$$

Where:
$R_{\mathrm{C}}^{\mathrm{Phy}}$ = Ratio of Carbon (C) inside the `compartment` Phy.
exp = Represents exponential function.
σ_{C} = Slope parameter for DIN uptake regulation ((mmol N)$^{-2}$ (mmol C)$^{-2}$), It is a **global** parameter.
$\mathrm{Q}2_{\mathrm{N:C}}^{\mathrm{Phy}}$ = Maximum cellular N:C quota of phytoplankto (mmol N (mmol C)$^{-1}$).
$q_{\mathrm{N:C}}^{\mathrm{Phy}}$ = Phytoplankton molar cellular $N : C$ ratio.
$()^{-1}$ = Inverse operation of an expression.

```
85  constant float R_C = inverse((1 + exp(-1 * sigma_N_C * (Q2_N_C_ratio - q_N_C_ratio))))
```

Listing 7.4. Implementation of the Equation 7.1.4 in our DSL

Table 7.2. Mapping of model to DSL for the Equation 7.1.4

Model Variable	BGC-DSL variables	BGC-DSL types
$R_{\mathrm{C}}^{\mathrm{Phy}}$	R_C	`constant float`
exp	exp	Function expression
σ_{C}	sigma_N_C	`constant float`
$\mathrm{Q}2_{\mathrm{N:C}}^{\mathrm{Phy}}$	Q2_N_C_ratio	`constant float`
$q_{\mathrm{N:C}}^{\mathrm{Phy}}$	q_N_C_ratio	`constant float`
$(...)^{-1}$	`inverse`	Function expression

7.1.4 Equations to represent connections

The BGC model uses mathematical equations to compute the chemical elements transferring from one compartment to another. In Equation 7.1.5, $assim_{\mathrm{DIN}}^{\mathrm{Phy}}$ computes the DIN assimilation rate inside

the compartment Phytoplankton (Phy). To implement this equation for a DSL, we use the keyword **connection** at the front, and the equation remains the same, but we change the variable names of the actual equation. Listing 7.5 shows an example for Equation 7.1.5 and Table 7.3 depicts the related mapping from this model variable names to DSL. For instance, the model is using the notation $assim_{\mathrm{DIN}}^{\mathrm{Phy}}$, but in the DSL, we use assim_DIN to represent the same notation inside the **compartment**, *Phy*. Moreover, we do not include the compartment name, as its context inside the **compartment** defines this.

$$assim_{\mathrm{DIN}}^{\mathrm{Phy}} = \mathrm{Q}_{u\,\mathrm{N:C}}^{\mathrm{Phy}} \cdot \mu_{max}^{\mathrm{Phy}} \cdot T_f^{\mathrm{Phy}} \cdot R_{\mathrm{P}}^{\mathrm{Phy}} \cdot R_{\mathrm{C}}^{\mathrm{Phy}} \cdot \frac{\mathrm{DIN}}{K_{\mathrm{DIN}} + \mathrm{DIN}} \tag{7.1.5}$$

Where:
$assim_{\mathrm{DIN}}^{\mathrm{Phy}}$ = Phytoplankton (Phy) nitrogen assimilation rate (mmol N (mmol C)$^{-1}$ d^{-1}).
$\mathrm{Q}_{u\,\mathrm{N:C}}^{\mathrm{Phy}}$ = N:C uptake ratio for N_2 fixation of phytoplankton.
μ_{max}^{Phy} = Maximum potential photosynthesis rate of phytoplankton.
T_f^{Phy} = Derived temparature from another equation.
$R_{\mathrm{C}}^{\mathrm{Phy}}$ = Calculated result from the Equation 7.1.4.
DIN = Dissolved Inorganic Nitrogen (DIN) is a **state** variable of the **compartment** Dissolved Inorganic Matter (DIM).

```
93 connection assim_DIN = Qu_N_C_ratio * meu_max * Tf * R_P * R_C * (DIM.DIN/(DIM.K_DIN + DIM.
       DIN))
```

Listing 7.5. Implentation of Equation 7.1.5

Table 7.3. Mapping of model to DSL for the Equation 7.1.5

Model Variable	BGC-DSL variables	BGC-DSL types
$assim_{\mathrm{DIN}}^{\mathrm{Phy}}$	assim_DIN	**connection**.
$Q_{u\,\mathrm{N:C}}^{\mathrm{Phy}}$	Qu_N_C_ratio	**constant** float.
μ_{max}^{Phy}	meu_max	**constant** float.
T_f^{Phy}	Tf	**constant** float.
$R_{\mathrm{C}}^{\mathrm{Phy}}$	R_C	**constant** float.
$R_{\mathrm{P}}^{\mathrm{Phy}}$	R_P	**constant** float.
DIN	DIM.DIN	**state** (from **compartment** DIM).

7.1.5 Equations to update the state variables

To update the state variables, this model also uses mathematical equations, and scientists use the term *Source Minus Sink (SMS)* to represent those equations. Inside the compartment *Phy* the model uses Equation 7.1.6 to update the **state** variable *Carbon (C)*. To implement this equation, we are using **update** statements and keeping the equation unchanged but renaming the variable names in our

implementation. Listing 7.6 shows the implementation for this equation, where we use the term **update** C in place of SMS(C). Table 7.4 shows the list of all variable name changes.

$$\begin{aligned} \text{SMS(C)} = {} & phot^{\text{Phy}}_{\text{DIC}} \cdot \text{Phy}_{\text{C}} - (resp^{\text{Phy}} + \gamma^{\text{Phy}_{\text{C}}} + aggr^{\text{Phy}}) \cdot \text{Phy}_{\text{C}} \\ & - graz^{\text{Phy}_{\text{N}}} \cdot q^{\text{Phy}}_{\text{C:N}} \end{aligned} \tag{7.1.6}$$

Where:
SMS (C) = Source Minus Sink (SMS).
$phot^{\text{Phy}}_{\text{DIC}}$ = Phytoplankton (Phy) Carbon assimilation rate (d^{-1}), it is also a calculated result from another equation.
Phy_{C} = Phytoplankton (Phy) carbon, it is a model state variable.
$resp^{\text{Phy}}$ = Amount of Phytoplankton (Phy) loses due to respiration (resp).
$\gamma^{\text{Phy}_{\text{C}}}$ = DOC exudation & leakage rate of Phytoplankton (Phy).
$aggr^{\text{Phy}}$ = Phytoplankton (Phy) decay rate (d^{-1}) due to aggregation.
$graz^{\text{Phy}_{\text{N}}}$ = Heterotroph's grazing rate ($mmol\ N\ d^{-1}$) related with Phytoplankton.
$q^{\text{Phy}}_{\text{C:N}}$ = Phytoplankton molar cellular C:N ratio ($mmol$ C $(mmol$ N$)^{-1}$)

```
102 update C = phot_DIC * C - (resp + gamma_C + aggr) * C - graz_N * q_C_N_ratio
```

Listing 7.6. Implementation of the Equation 7.1.6

Table 7.4. Mapping of model to DSL for the Equation 7.1.6

Model Variable	BGC-DSL variables	BGC-DSL types
SMS (C)	C	**update**
$phot^{\text{Phy}}_{\text{DIC}}$	phot_DIC	**constant** float
Phy_{C}	C	**state**
$resp^{\text{Phy}}$	resp	**connection**
$\gamma^{\text{Phy}_{\text{C}}}$	gamma_C	**constant** float
$aggr^{\text{Phy}}$	aggr	**connection**
$graz^{\text{Phy}_{\text{N}}}$	graz_N	**connection**
$q^{\text{Phy}}_{\text{C:N}}$	q_C_N_ratio	**constant** float

7.2 Case Study 2: Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)

The model *Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)* from the paper of Pahlow et al. [Pah+20] also contains *global* and *local* constants, *state variables*, and equations to update *state variables* like the model of Section 7.1 but uses special *mathematical functions* and does not use any equations to represent the *connections*. Since only *mathematical functions* and equations to update, *state variables* are different from the previous case study. In the following, we discuss the evaluation for

these two parts. For this case study, we use the compartment *PhyDia (Phytoplankton and Diazotrophs)* and one function from the compartment *DetDisPools (Detritus and dissolved pools)* (see Appendix C).

7.2.1 Mathematial functions

This model uses two mathematical functions; one is δ *tracers function* which represents the differences between actual and subsistence phytoplankton Particulate Organic Nitrogen (PON) and Particulate Organic Phosphorus (POP) concentrations [Pah+20], and another one is for measuring detritus *sinking speed* v_{sink}.

Equation 7.2.1 from the paper is the δ *tracers function*, a composite function. The values for C_p, N_p, P_p, and $Q^n_{0,p}$ has to provide as a *local* or *global* constant before executing this function. Our DSL does not directly support implementing a composite function like this, but we have indirect support for this kind of function. Listing 7.7 shows the sample implementation of this δ function, where we use four individual statements to represent this function. We are also assigning the sample values to all *Nphy, Ndia, Pphy, Pdia, Q_0_phy_N, Q_0_phy_P, Q_0_dia_N, Q_0_dia_P* in the `global` section and assigning sample values to *Cphy, Cdia* inside this `compartment` before the implementation. The type of δn_p is `state` and for four equations we get four (*delta_Nphy*, *delta_Pphy*, *delta_Ndia*, *delta_Pdia*) results for the same state variable. Table 7.5 shows the complete set of model and DSL variables with their respective DSL types.

$$\begin{aligned} \delta n_p &= n_p - C_p \cdot Q^n_{0,p}, \\ & n \in \{N, P\},\ p \in \{phy, dia\} \end{aligned} \tag{7.2.1}$$

Where:
C_p, N_p and P_p are POC, PON and POC respectively.
$Q^n_{0,p}$ = Minimum (subsistence) concentrations, lower limit is 0.
N, P, phy, and dia represent Nitrogen, Phosphorus, Phytoplankton and Diazotrophs respectively.

```
27  state delta_Nphy = Nphy - Cphy * Q_0_phy_N
28  state delta_Pphy = Pphy - Cphy * Q_0_phy_P
29
30  state delta_Ndia = Ndia - Cdia * Q_0_dia_N
31  state delta_Pdia = Pdia - Cdia * Q_0_dia_P
```

Listing 7.7. Implementation of Equation 7.2.1

Table 7.5. Mapping of model to DSL for the Equation 7.2.1

Model Variable	BGC-DSL variables	BGC-DSL types
δn_p	delta_Nphy, delta_Pphy, delta_Ndia, delta_Pdia	`state`
n_p	Nphy, Pphy, Ndia, Pdia	`constant float`
C_p	Cphy, Cdia	`state`
$Q^n_{0,p}$	Q_0_phy_N, Q_0_phy_P, Q_0_dia_N, Q_0_dia_P	`constant float`

The compartment *DetDisPools (Detritus and dissolved pools)* uses Equation 7.2.2 to measure the detritus sinking speed. In the DSL, we use Listing 7.8 shows the definition of this equation. Here, we are assigning the values into *z*, *v0*, *a_v* inside the compartment as *local* constant (see Appendix C) and using `constant float` as the data type for each variable (see Table 7.6).

$$v_{sink} = v_0 - a_v \cdot z \tag{7.2.2}$$

Where, $v_0 = 6md^{-1}$ is the sinking velocity at the surface, z is depth and a $a_v = 0.06d^{-1}$ the rate of increase in v_{sink} with depth [Pah+20].

```
85  constant float v_sink = v0 + a_v * z
```

Listing 7.8. DSL implementation of Equation 7.2.2

Table 7.6. Mapping of model to DSL for the Equation 7.2.2

Model Variable	BGC-DSL variables	BGC-DSL types
v_{sink}	v_sink	`constant float`
v_0	v0	`constant float`
a_v	a_v	`constant float`
z	z	`constant float`

7.2.2 Equations to update state variables

This BGC model also uses the *Source Minus Sink (SMS)* equation to compute the `state` variable value. Equation 7.2.3 is used by this model to compute the updated state of C_p. The model user needs to provide the initial amount of C_p, μ_p, λ_p, M_p, and G_p^C as *global* or *local* constant. Since it is a composite function, we need multiple statements in our DSL to implement this equation. Listing 7.9 shows the DSL implementation for this equation, where we use two separate statements to represent this equation. We store the computed result in two `update` variables called *Cphy* and *Cdia*. Table 7.7 shows the set of all variable mapping with their types from the model to DSL.

$$S(C_p) = (\mu_p - \lambda_p - M_p) \cdot C_p - G_p^C,\ p \in \{phy, dia\} \tag{7.2.3}$$

Where, C_p is POC, μ_p is net relative (C-specific) growth rate (C fixation minus the sum of respiration and release of dissolved organic carbon by phytoplankton, immediately respired to DIC here), λ_p leakage, M_p mortality, G_n^C grazing by zooplankton(C-specific) [Pah+20].

```
61  update Cphy = (meuPhy - lamdaPhy - Mphy) * Cphy - Gphy_C
62  update Cdia = (meuDia - lamdaDia - Mdia) * Cdia - Gdia_C
```

Listing 7.9. Implementation of Equation 7.2.3

Table 7.7. Mapping of model terms to DSL for the Equation 7.2.3

Model Variable	BGC-DSL variables	BGC-DSL types
SMS (C_p)	Cphy, Cdia	`update`
μ_p	meuPhy, meuDia	`constant float`
λ_p	lamdaPhy, lamdaDia	`constant float`
M_p	Mphy, Mdia	`constant float`
C_p	Cphy, Cdia	`state`
G_p^C	Gphy_C, Gdia_C	`constant float`

7.3 Summary

The case studies show that each model from the papers has some common properties, like they have global or local constants, and they use mathematical functions and equations. The DSL is capable (directly or indirectly) of defining all of the properties of those BGC models.

Chapter 8

Conclusion and Outlook

8.1 Conclusion

BGC model scientists sometimes need to change the model specifications multiple times to build a model successfully. Each time they change their specification, they need to communicate with model developers to change the line of codes into GPL and make a runtime for them to re-test the model. This process is sometimes error-prone and time-consuming. Here, DSLs make it by converting domain specification into runtime and reducing the dependencies on the software developers. To develop DSLs for BGC models, we studied the related works, conducted interviews with scientists and analyzed these interviews, examined the BGC model papers, implemented the model, and finally evaluated the implemented DSL utilizing two BGC models.

From the interviews, we got pieces of information, like *types of BGC models*, *working environment*, and *equations they use* (see Section 4.1 on page 15) and also found some interesting themes like the *general development process for BGC models* (Section 4.2.1 on page 18) and *the tooling used during development* (Section 4.2.2 on page 19).

In the interviews, scientists also suggested some BGC model papers. To understand the core building block of BGC models, we analyzed four helpful papers from them. Those papers focus on three BGC models. The first model is called *C–N–P regulated ecosystem model (CNP-REcoM)*, which gives a general idea of what the biogeochemical processes of a model look like and also shows the links to significant model compartments (see Section 5.2.1 on page 24). The second model, called *Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)* explains some mathematical equations the model used to communicate among pools (compartments), and the third model, namely *Modelling carbon overconsumption and forming extracellular particulate organic carbon*, shows the graphical illustration of a BGC model and the mathematical equations used by each compartment to communicate with each other (see Section 5.2.3 on page 29).

Based on the information gathered from interviews and BGC model papers, we start the implementation of Biogeochemical Domain Specific Language (BGC-DSL). It has two parts, the first part is the `global` section, and the other is the compartments. Every model may have multiple compartments, which use the constants initialized inside the *global* section. Each `compartment` has its `state` variables, *local* constants, and mathematical equations to compute the connections and update the `state` variables. The order of using state variables, constants, connections, and updates inside a `compartment` is vital. We initialize the state variables first, then constants, then connections, and finally updates. After implementation, we evaluated the BGC-DSL utilizing two BGC models.

In summary, the evaluation result shows that BGC-DSL can successfully define all the model behavior of the first model (*C–N–P regulated ecosystem model*) as the level of abstraction used in the

paper (see Section 7.1 on page 45). It also can perfectly define the second model (*Optimality-based non-Redfield plankton–ecosystem model*) but does not follow the same level of abstraction used in the paper (see Section 7.2 on page 49).

8.2 Outlook

Potential changes can bring possible improvement to any system. For example, in our case, while defining (see Listing 7.7 on page 50) Equation 7.2.1, we need four equations instead of one, which is not optimal. Moreover, in BGC-DSL , there is no support for the expressions like matrix manipulation, integration, and differentiation. So, further improvement to this DSL allows efficient implementation of those expressions. Other improvement point is the `global` setting section, which is easily replaceable by a dedicated DSL to reuse the configuration parameter declarations in different models. Here, we can use *Configuration and Parametrization DSL (CP DSL)* in place of the global section. Since CP DSL is still in development, our DSL did not include that either. A *code generator* is also missing for this DSL since it is not a part of our thesis. Another important thing we can do next is the visualization of the BGC models and automatically produces graphics, like in the papers. KIELER Lightweight Diagrams (KLighD) from the KIELER research and software project group is a great visualization tool [KIE22].

Appendix A

Interview Guide

Our aim is to provide specific programming languages, called domain-specific languages, to help to separate scientific code, technical code, and automate steps that are tedious and error prone. Here in particular we want to address Bio-Geo-Chemical (BGC) models. To be able to do so, we need to understand how you work and with what you work.

A.1 Introductory questions

- ▻ Affiliation (Which Research Group)
- ▻ Scientific background
- ▻ What is your research topic and current project?
- ▻ When you work with BGC-models are they embedded or coupled with ocean models?
- ▻ Which ocean models do you use?

A.2 General questions to BGC models

We had heared that some scientists start with chemical formula and others develop partial differential equations based on observation and literature.

- ▻ What kind of BGC modeling approaches exist in general?
- ▻ Which approaches do you use and why?

A.3 Process related questions

- ▻ What kind of BGC models do you use and or develop?
- ▻ How is your process to develop them, i.e. you start with a research question, a hypothesis, an observation and then arrive at a implemented model. What is necessery in between to get there?
- ▻ What notations, e.g. math, chemistry formulas, etc. do you use in your research?
- ▻ (Differential equitations and Chemical formula)

A.4 Work environment

▻ Are you modeling on a local machine or are you working remote?

▻ What tooling do you use? (Matlab, Octave, Fortran, editors)

Appendix B

C–N–P regulated ecosystem model (CNP-REcoM)

```
model CNP_REcoM

global

  // global environmental variables. (have to provide)
  constant float T_in_K = 120 // Temperature in K , not given
  constant float I = 1 // (W m ^ -2) Irradiance (PAR)

  // Model parameters (category-A).
  constant float T_ref = 283.15
  constant int Ar = 4500


  constant int sigma_N_C = 100
  constant int sigma_P_N = 50

  constant float Zeta_N2 = 4.2

  constant float A_E = 0.75
  constant float epsilon = 0.78
  constant float Omega = 0.2
  constant float rdenit = 0.05
  constant float k_denit = 4.0
  constant float rsedi_C = 2.0
  constant int dd = 3
  constant float rsedi_N = 2.0
  constant float rsedi_P = 0.0
  constant float g_max = 0.3
  constant float A_E = 0.75
  constant float Rho_C = 0.06
  constant float Rho_N = 0.24
  constant float Rho_P = 0.36
  constant float omega_C = 0.02
  constant float omega_N = 0.06
  constant float omega_P = 0.12
  constant float Psi_max = 0.8
```

```
37    constant float Rho_MG = 0.06
38
39    constant float Tf = exp(-1 * Ar * (inverse(T_in_K) - inverse(T_ref)))
40
41  // Phytoplankton Phy:
42  compartment Phy {
43
44    // Initial amount of chemical elements i each compartment, The given values are not real
45    state C = 14.16
46    state N = 1.40
47    state P = 0.114
48    state Chl = 1
49
50    // environmental variables
51    constant float q_P_N_ratio = 1
52    constant float q_C_N_ratio = 1
53    constant float q_C_P_ratio = 1
54    constant float q_N_P_ratio = 1
55    constant float q_N_C_ratio = 1
56    constant float Theta_C = 1.2
57    constant float Theta_N = 1.2
58
59    // local constants
60    constant float Ar_Phy = 4000 // no unit
61    constant float phot_max = 2.6
62    constant float meu_max = 3.3
63    constant float W = -0.2
64    constant float r0 = 0.01
65    constant float gamm_Chl = 0.01
66    constant float Qu_N_C_ratio = 0.171
67
68    constant float alpha = 0.6
69    constant float gamma_max = 2.6
70    constant float gamma_C = 0.08
71    constant float gamma = 0.06
72
73    constant float Phi = 0.02
74
75    constant float Q1_N_C_ratio = 0.043
76    constant float Q2_N_C_ratio = 0.171
77    constant float Q1_P_N_ratio = 0.02
78    constant float Q2_P_N_ratio = 0.2
79    constant float Qu_P_N_ratio = 0.014
80    constant float Theta2_N = 2.4
81
82    constant int sampleArray = [3, 3, 4, 2] * [2, 3, 4*2, 2+2]
83
```

```
  constant float Tf_Phy = exp(-1 * Ar_Phy * (inverse(T_in_K) - inverse(T_ref)))
  constant float R_C = inverse((1 + exp(-1 * sigma_N_C * (Q2_N_C_ratio - q_N_C_ratio))))
  constant float R_P = 1 - (Q1_P_N_ratio / q_P_N_ratio)
  constant float R_N = inverse( 1 + exp(-1 * sigma_P_N * (Q2_P_N_ratio - q_P_N_ratio)))
  constant float Psi = max(0, Psi_max * (1 - (q_N_C_ratio/Q2_N_C_ratio)))
  constant float phot_max = meu_max * Tf * (q_N_C_ratio - Q1_N_C_ratio)/(Q2_N_C_ratio -
      Q1_N_C_ratio)
  constant float phot_DIC = 1.2 //phot_max ( 1- exp (( - alpha * Theta_C) * I) / phot_max))

  connection grazing_N = g_max * Tf * (N * N)/(Het.K_N * Het.K_N + (N * N + Nif.N * Nif.P)) *
       Het.N
  connection assim_DIN = Qu_N_C_ratio * meu_max * Tf * R_P * R_C * (DIM.DIN/(DIM.K_DIN + DIM.
      DIN))
  connection assim_DIP = Qu_P_N_ratio * meu_max * Tf * R_N * (DIM.DIP/(DIM.K_DIP + DIM.DIP))

  connection resp = r0 * Tf + DIM.Zeta_DIN * assim_DIN
  connection aggr = Det.Phi * N + Det.Phi * Det.N
  connection synth_Chl = assim_DIN * Theta2_N * phot_DIC / (alpha * Theta_C * I)
  connection degr_Chl = gamm_Chl

  // Final values
  update C = phot_DIC * C - (resp + gamma_C + aggr) * C - grazing_N * q_C_N_ratio
  update N = assim_DIN * C - ( r0 * Tf + gamma + aggr ) * N - grazing_N
  update P = assim_DIP * C - ( r0 * Tf + gamma + aggr ) * P - grazing_N * q_C_N_ratio
  update Chl = synth_Chl * C - ( degr_Chl + aggr ) * Chl - grazing_N * Theta_N
}

// Diazotrophs Nif
compartment Nif {

  // Initial amount of chemical elements i each compartment, The given values are not real
  state C = 14.16
  state N = 1.40
  state P = 0.114
  state Chl = 1

  // Environmental (have to provide)
  constant float q_N_C_ratio = 0.043 // not given
  constant float q_P_N_ratio = 0.043 // not given
  constant float q_C_N_ratio = 1 // not given
  constant float Theta_C = 1.2 // not given
  constant float Theta_N = 1.2 // not given

  // Local constants
  constant float Ar_Nif = 1500
  constant float W = 0.1
  constant float r0 = 0.01
```

```
128  constant float gamma_Chl = 0.01
129  constant float Qu_N_C_ratio = 0.171
130  constant float Qu2_N_C_ratio = 0.09
131
132  constant float alpha = 0.6
133  constant float meu_max = 1.0
134  constant float gamma_C = 0.08
135  constant float gamma = 0.06
136  constant float Phi = 0.02
137
138  // Q has global values also, different naming important
139  constant float Q1_Nif_N_C_ratio = 0.043
140  constant float Q2_Nif_N_C_ratio = 0.171
141  constant float Q1_Nif_P_N_ratio = 0.02
142  constant float Q2_Nif_P_N_ratio = 0.2
143  constant float Qu_Nif_P_N_ratio = 0.025
144  constant float Theta2_N = 1.2
145
146  // Tf has a global value, a local value should have differnt name
147  constant float Tf_Nif = exp(-1 * Ar_Nif * (inverse(T_in_K) - inverse(T_ref)))
148  constant float phot_max = meu_max * Tf_Nif * (q_N_C_ratio - Q1_Nif_N_C_ratio)/(
         Q2_Nif_N_C_ratio - Q1_Nif_N_C_ratio)
149  constant float phot_DIC = phot_max * ( 1 - exp((-1 * alpha * Theta_C * I)/phot_max))
150  constant float f_N2 = max(0, 1 - ((DIM.DIN/DIM.DIN + DIM.K_DIN)*((DIM.DIP + DIM.K_DIP)/DIM.
         DIP)))
151  constant float R_P = 1 - (Q1_Nif_P_N_ratio/q_P_N_ratio)
152  constant float R_C = inverse(1 + exp(-1 * sigma_N_C * (Q2_Nif_P_N_ratio - q_P_N_ratio)))
153  constant float R_N = inverse ( 1 + exp(-1 * sigma_P_N * ( Q2_Nif_P_N_ratio - q_P_N_ratio)))

154  constant float Psi = max(0, Psi_max * (1 - (q_N_C_ratio/Q2_Nif_N_C_ratio)))
155
156  connection assim_DIN = Qu_Nif_P_N_ratio * (1 - f_N2) * meu_max * Tf_Nif * R_P * R_C
157  connection assim_N2 = Qu_Nif_P_N_ratio * f_N2 * meu_max * Tf_Nif * R_P * R_C
158  connection assim_DIP = Qu_Nif_P_N_ratio * meu_max * Tf_Nif * R_N
159  connection synth_Nif_Chl = (assim_DIN + assim_N2) * Theta2_N * phot_DIC/(alpha * Theta_C *
          I)
160  connection resp = r0 * Tf_Nif + DIM.Zeta_DIN * assim_DIN + Zeta_N2 * assim_N2
161  connection aggr = Phi * N + Det.Phi * Det.N
162  connection degr_Chl = gamma_Chl // (for Theta_Nif_N > Theta2_Nif_N : degr_Nif_Chl = (r0 *
         Tf_private + gamma_Nif))
163  connection graz_N = g_max * Tf * (N * N)/(Het.K_N * Het.K_N + (N * N + Phy.N * Phy.N)) *
         Het.N
164
165  update C = phot_DIC * C - (resp + gamma_C + aggr) * C - graz_N * q_C_N_ratio
166  update N = (assim_DIN + assim_N2) * C - (r0 * Tf_Nif + gamma) * N - graz_N
167  update P = assim_DIN * N - (r0 * Tf_Nif + aggr) * P - graz_N * q_P_N_ratio
168  update Chl = synth_Nif_Chl * C - ( degr_Chl + aggr) * Chl - graz_N * Theta2_N
```

```
}


//Heterotrophs Het:
compartment Het {

  // Initial amount of chemical elements i each compartment, The given values are not real
  state C = 14.16
  state N = 1.40
  state P = 0.114

  // Compartment ratio calculation
  constant float q_C_N_ratio = 1 // not given
  constant float q_C_P_ratio = 1 // not given
  constant float q_N_P_ratio = 1 // not given
  constant float q_N_C_ratio = 1 // not given

  // Global
  constant float tao= 0.5
  constant float gamma = 0.30
  constant float K_N = 1.0
  constant float r0 = 0.01
  constant float Qr_C_N_ratio = 0.171
  constant float Qr_N_P_ratio = 0.171
  constant float Theta_MG = 0.06

  connection resp = Tf * r0 + tao * (max(0, 1 - (Qr_C_N_ratio/q_C_N_ratio), 1 - (
      Qr_C_N_ratio * Qr_N_P_ratio)/q_C_P_ratio)^2)
  connection excr_N = tao * (max(0, 1 - (q_C_N_ratio/Qr_C_N_ratio), 1 - (Qr_N_P_ratio)/
      q_N_P_ratio)^2)
  connection excr_P = tao * (max(0, 1 - (q_C_P_ratio/(Qr_C_N_ratio * Qr_N_P_ratio)), 1 - (
      q_N_P_ratio / Qr_N_P_ratio))^2)
  connection mort = gamma * N

  update C = A_E * (Phy.grazing_N * Phy.q_C_N_ratio + Nif.graz_N * Nif.q_C_N_ratio) - (resp +
       mort) * C
  update N = A_E * (Phy.grazing_N * Nif.graz_N) - (excr_P + mort) * P
  update P = A_E * (Phy.grazing_N * Phy.q_P_N_ratio + Nif.graz_N * Nif.q_P_N_ratio) - (
      excr_P + mort) * P
}


//Detritus Det
compartment Det {

  // Initial amount of chemical elements i each compartment, The given values are not real
  state C = 14.16
```

```
211   state N = 1.40
212   state P = 0.114
213
214    constant float Phi = 0.02
215   constant float sedimentation_C = rsedi_C * C
216   constant float sedimentation_N = rsedi_N * N
217   constant float sedimentation_P = rsedi_P * P
218
219   update C = Phy.aggr * Phy.C + Nif.aggr * Nif.C * dCCHO_MG.Phi * N * dCCHO_MG.MGC
220   + (1 - A_E) * (Phy.grazing_N + Phy.q_C_N_ratio + Nif.graz_N * Nif.q_C_N_ratio)
221   + Omega * Het.mort * Het.C - omega_C * C - sedimentation_C
222   update N = Phy.aggr * Phy.N + Nif.aggr * Nif.N + dCCHO_MG.Phi * N * dCCHO_MG.MGN
223   + (1 - A_E) * (Phy.grazing_N + Nif.graz_N)
224   + Omega * Het.mort * Het.N - omega_N * N - sedimentation_N
225   update P = Phy.aggr * Phy.P + Nif.aggr * Nif.P
226   + (1 - A_E) * (Nif.graz_N * Phy.q_P_N_ratio + Nif.graz_N * Nif.q_P_N_ratio)
227   + Omega * Het.mort * Het.P - omega_P * P - sedimentation_P
228 }
229
230
231 //Dissolved inorganic matter DIM:
232 compartment DIM {
233
234   // The given values are not real
235   state DIC = 1.0 // mmol C m^-3
236   state DIN = 1 // mmol N m^-3
237   state DIP = 1
238
239   constant float Zeta_DIN = 2.3
240   constant float K_DIN = 1.0
241   constant float K_DIP = 0.1
242   constant float denitrification = rdenit * ((DIN * DIN) / (DIN * DIN) * (k_denit * k_denit))
          * Det.N
243
244   update DIC = Tf * (Rho_C * DOM.LDOC + dCCHO_MG.MGC) + Phy.resp * Phy.C
245   + Nif.resp * Nif.C + (Het.resp + epsilon * (1- Omega) * Het.mort) * Het.C
246   - Phy.phot_DIC * Phy.C - Nif.phot_DIC * Nif.C
247   update DIN = Tf * (Rho_N * DOM.LDON + Rho_MG * dCCHO_MG.MGN) + Phy.r0 * Phy.Tf_Phy * Nif.N
          * epsilon * (Het.excr_N + (1 - Omega) * Het.mort) * Het.N
248   - Phy.assim_DIN * Phy.N - Nif.assim_DIN * Nif.N - denitrification
249   update DIP = Tf * Rho_P * DOM.LDOP * Phy.r0 * Phy.Tf_Phy * Phy.P * Nif.r0 * Nif.Tf_Nif *
        Nif.P
250   + epsilon * (Het.excr_P + (1 - Omega) * Het.mort) * Het.P
251   - Phy.assim_DIP * Phy.P - Nif.assim_DIP * Nif.P
252 }
253
254 //Dissolved organic matter DOM:
```

```
compartment DOM {

  // The given values are not real
  state LDOC = 1.0
  state LDON = 1.0
  state LDOP = 1.0


  update LDOC = omega_C * Det.C + (1 - Phy.Psi) * Phy.gamma_C * Phy.C
  + (1 - Nif.Psi) - Nif.gamma_C * Nif.C + ( 1 - epsilon) * ( 1 - Omega) * Het.mort * Het.C
  - Rho_C * Tf * LDOC
  update LDON = omega_N * Det.N + Phy.gamma * Phy.N + (1-epsilon) * Het.excr_N * Het.N
  + (1 - epsilon) * ( 1- Omega) * Het.mort * Het.N
  - ( Rho_N * Tf + dCCHO_MG.Xi * dCCHO_MG.MGC) * LDON
  update LDOP = omega_P * Det.P + Phy.gamma * Phy.P + Nif.gamma * Nif.P
  + (1- epsilon) * ( Het.excr_P+ (1- Omega)* Het.mort)*Het.P
  - Rho_P * Tf * LDOP
}


//Dissolved polysaccharides (dCCHO) and particulate macrogels MGC; MGN:

compartment dCCHO_MG {

  state dCCHO = 1
  state MGC = 1
  state MGN = 1
  state TA = 1

  // Global
  constant float Phi = 0.06
  constant float Phi_dCCHO = 0.0015
  constant float Phi_TEPC = 0.0128
  constant float Xi = 0.05 //(special symbol)

  constant float denitrification = rdenit * ((DIM.DIN * DIM.DIN) / (DIM.DIN * DIM.DIN) * (
      k_denit * k_denit)) * Det.N

  update dCCHO = Phy.Psi * Phy.gamma_C * Phy.C + Nif.Psi * Nif.gamma_C * Nif.C
  - (Phi_dCCHO * dCCHO + Phi_TEPC * MGC) * dCCHO
  update MGC = (Phi_TEPC * MGC + Phi_dCCHO * dCCHO) * dCCHO
  - (Phi * Tf + Phi * Det.N) * MGC
  update MGN = Xi * MGC * DOM.LDON
  - (Phi * Tf + Phi * Det.N) * MGN
  update TA = Phy.assim_DIN * Phy.C + Phy.assim_DIP * Phy.N
  + Nif.assim_DIN * Nif.C + Nif.assim_DIP * Nif.N
  + Phy.r0 * Phy.Tf_Phy * Phy.N - Nif.r0 + Nif.Tf_Nif * Nif.N
```

```
301   - Phy.r0 * Phy.Tf_Phy * Phy.P - Nif.r0 * Nif.Tf_Nif * Nif.P
302   - epsilon * (Het.excr_N + (1-Omega)*Het.mort)*Het.N
303   - epsilon * (Het.excr_P + (1 - Omega) * Het.mort) * Het.P
304   + Tf * (Rho_N * DOM.LDON - Rho_MG * MGN - Rho_P * DOM.LDOP)
305   + denitrification
306 }
```

Listing B.1. DSL implementation of the model C–N–P regulated ecosystem model (CNP-REcoM) with model name, global section, and a sample compartment

Appendix C

Optimality-based non-Redfield plankton–ecosystem model (OPEMv1.1)

```
model ObESM

global

  constant float A0 = 1.2


  constant float Nphy = 1
  constant float Ndia = 1

  constant float Pphy = 1
  constant float Pdia = 1

  constant float Q_0_phy_N = 1
  constant float Q_0_phy_P = 1
  constant float Q_0_dia_N = 1
  constant float Q_0_dia_P = 1



// Phytoplankton and diazotrophs
compartment PhyDia {
  // needs to provice, 1 is not original value


  state Cphy = 1
  state Cdia = 1

  state delta_Nphy = Nphy - Cphy * Q_0_phy_N
  state delta_Pphy = Pphy - Cphy * Q_0_phy_P

  state delta_Ndia = Ndia - Cdia * Q_0_dia_N
  state delta_Pdia = Pdia - Cdia * Q_0_dia_P

  constant float Mphy = 1.2
  constant float Mdia = 1.2
```

```
37
38    // local constant, needs to provide, These are not original values
39    constant float Mphy = 1.2
40    constant float Mdia = 1.2
41
42    constant float Vphy_N = 2.1 // potential-rate parameter
43     constant float Vphy_P = 2.1 // potential-rate parameter
44     constant float Vdia_N = 2.1 // potential-rate parameter
45     constant float Vdia_P = 2.1 // potential-rate parameter
46
47    constant float lamdaPhy = 3.2
48    constant float lamdaDia = 3.2
49    constant float meuPhy = 1.3
50    constant float meuDia = 1.3
51    constant float Mphy = 1.4
52
53    constant float Gphy_C = 1
54    constant float Gdia_C = 1
55
56    constant float Q_0_phy_N = 1.2
57    constant float Q_0_phy_P = 1.2
58
59    constant float Q_0_dia_N = 1.3
60    constant float Q_0_dia_P = 1.3
61
62
63    update Cphy = (meuPhy - lamdaPhy - Mphy) * Cphy - Gphy_C
64    update Cdia = (meuDia - lamdaDia - Mdia) * Cdia - Gdia_C
65
66     update delta_Ndia = Vdia_N * Cdia - (lamdaDia + Mdia) * Ndia - Gdia_C
67     update delta_Pdia = Vdia_P * Cdia - (lamdaDia + Mdia) * Pdia - Gdia_C
68
69     update delta_Nphy = Vphy_N * Cphy - (lamdaPhy + Mphy) * Nphy - Gphy_C
70     update delta_Pphy = Vphy_P * Cphy - (lamdaPhy + Mphy) * Pphy - Gphy_C
71 }
72
73
74 compartment Zooplankton {
75      // Initial config, these are not actual values
76    state N = 1.2
77    state C = 1.2
78    state P = 1.2
79
80    constant float meu = 1.2
81    constant float G_N = 1.1
82    constant float M = 1.2
83    constant float Q_C = 1.2
```

```
84    constant float Q_N = 1.2
85    constant float Q_P = 1.2
86
87    constant float X_C = 1.2
88    constant float X_N = 1.2
89    constant float X_P = 1.2
90
91    update N = meu * N - G_N - M * (N*N/Q_N)
92  }
93
94  // Detritus and dissolved pools
95  compartment DetDisPools {
96    // Initial config, these are not actual values
97    state C = 1.2
98    state N = 1.2
99    state P = 1.2
100
101   constant float G_C = 1.1
102   constant float G_N = 1.1
103   constant float G_P = 1.1
104   constant float z = 1 // depth
105   constant float v0 = 6.0 // m per day
106   constant float a_v = 0.06 // per day
107   constant float f_T = 1.2 // needs to evaluate the function
108   constant float v = v0 + a_v * z // evaluate the velocity
109
110   update C = PhyDia.Mphy * PhyDia.Cphy + PhyDia.Mdia * PhyDia.Cdia + Zooplankton.M * (
          Zooplankton.C*Zooplankton.C)/Zooplankton.Q_C
111         + Zooplankton.X_C - G_C - f_T * v * C
112
113   update N = PhyDia.Mphy * Nphy + PhyDia.Mdia * Ndia + Zooplankton.M * (Zooplankton.N*
          Zooplankton.N)/Zooplankton.Q_N
114    + Zooplankton.X_N - G_N - f_T * v * N
115
116    update P = PhyDia.Mphy * Nphy + PhyDia.Mdia * Ndia + Zooplankton.M * (Zooplankton.P*
           Zooplankton.P)/Zooplankton.Q_P
117    + Zooplankton.X_P - G_P - f_T * v * P
118
119 }
```

Listing C.1. Implementation of global section and two compartment PhyDia (Phytoplankton and diazotrophs) and DetDisPools (Detritus and dissolved pools) of the model OPEMv1.1.

Bibliography

[Ada15] William Adams. "Conducting semi-structured interviews". In: Aug. 2015. DOI: 10.1002/9781119171386.ch19.

[Ahm22] Faiz Ahmed. *Biogeochemical domain specific language*. 2022. URL: https://git.se.informatik.uni-kiel.de/thesis/faiz-ahmed-msc/-/blob/master/code/org.oceandsl.bgcdsl/src/org/oceandsl/BgcDsl.xtext.

[Aho+06] Alfred V. Aho et al. *Compilers: principles, techniques, and tools (2nd edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321486811.

[BC06] Virginia Braun and Victoria Clarke. "Using thematic analysis in psychology". In: *Qualitative Research in Psychology* 3.2 (2006), pp. 77–101. DOI: 10.1191/1478088706qp063oa. eprint: https://www.tandfonline.com/doi/pdf/10.1191/1478088706qp063oa. URL: https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa.

[Ber+20] Danielle Berardi et al. "21st-century biogeochemical modeling: challenges for century-based models and where do we go from here?" In: *GCB Bioenergy* 12.10 (2020), pp. 774–788. DOI: https://doi.org/10.1111/gcbb.12730. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/gcbb.12730. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/gcbb.12730.

[Boy98] Richard Boyatzis. "Transforming qualitative information: thematic analysis and code development". In: (Jan. 1998).

[Bur15] Vivien Burr. *Social constructionism (3rd ed.). routledge*. Routledge, 2015. ISBN: 9781315715421. URL: https://doi.org/10.4324/9781315715421.

[Dep22] Deployment. *Software development processes in ocean system modeling*. Aug. 2022. URL: https://arxiv.org/abs/2108.08589.

[Ecl22] Eclipse. *The community for open innovation and collaboration*. 2022. URL: https://www.eclipse.org/.

[Eco22] Ecore. *Provides an api for the ecore dialect of uml*. Aug. 2022. URL: https://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html.

[Ema22] GNU Emacs. *An extensible, customizable, free/libre text editor — and more.* Apr. 2022. URL: https://www.gnu.org/software/emacs/.

[EMF22] EMF. *Eclipse modeling framework (emf)*. 2022. URL: https://www.eclipse.org/modeling/emf/.

[for22] fortran-lang. *High-performance parallel programming language*. Mar. 2022. URL: https://fortran-lang.org/en/.

[Jet22] JetBrains. *Domain-specific languages*. https://www.jetbrains.com/mps/concepts/domain-specific-languages/. Accessed: 2022-09-06. 2022. URL: https://www.jetbrains.com/mps/concepts/domain-specific-languages/.

[JH17] Arne Johanson and Wilhelm Hasselbring. "Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment". In: *Empirical Software Engineering* 22.4 (Aug. 2017), pp. 2206–2236. ISSN: 1382-3256. DOI: 10.1007/s10664-016-9483-z.

[Ken09] Barry L. Kurtz Kenneth Slonneger. *Formal syntax and semantics of programming languages: a laboratory based approach*. Books4Cause Inc., 1994-09. ISBN: 0201656973.

[KIE22] KIELER. *Kieler lightweight diagams*. 2022. URL: https://github.com/kieler/KLighD.

[Kie22a] Christian-Albrechts-Universität zu Kiel. *The nec hpc system (alias nesh) of the university computing centre (rz) is a hybrid computing system which provides a high computing power in form of a combination of a scalar nec hpc linux cluster including gpus and a nec sx-aurora tsubasa vector system*. 2022. URL: https://www.rz.uni-kiel.de/en/our-portfolio/hiperf/nesh.

[Kie22b] University of Kiel. *Oceandsl aims to provide tools and processes to develop, evolve and use ocean biogeochemical models.* 2022. URL: https://oceandsl.uni-kiel.de/work-packages-2/.

[Kre+15] Markus Kreus et al. "Variations in the elemental ratio of organic matter in the central baltic sea: part i—linking primary production to remineralization". In: *Continental Shelf Research* 100 (2015), pp. 25–45. ISSN: 0278-4343. DOI: https://doi.org/10.1016/j.csr.2014.06.015. URL: https://www.sciencedirect.com/science/article/pii/S0278434314002313.

[KS15] Markus Kreus and Markus Schartau. "Variations in the elemental ratio of organic matter in the central Baltic Sea: Part II - Sensitivities of annual mass flux estimates to model parameter variations". In: *Continental Shelf Research* 100 (June 2015), pp. 46–63. DOI: 10.1016/j.csr.2015.02.004.

[Man20] Girish Managoli. *What developers need to know about domain-specific languages*. Feb. 2020. URL: https://opensource.com/article/20/2/domain-specific-languages.

[Mat22] Mathworks. *Designed for the way you think and the work you do.* Aug. 2022. URL: https://www.mathworks.com/products/matlab.html.

[MHS05] Marjan Mernik, Jan Heering, and Anthony Sloane. "When and how to develop domain-specific languages". In: *ACM Comput. Surv.* 37 (Dec. 2005), pp. 316–. DOI: 10.1145/1118890.1118892.

[Nem14] Nemo. *Nucleus for european modelling of the ocean*. 2014. URL: https://www.nemo-ocean.eu/.

[Oce22] Oceandsl. *Configuration and parametrization dsl for ochean models*. Aug. 2022. URL: https://git.se.informatik.uni-kiel.de/oceandsl/cp-dsl.git.

[Oct22] GNU Octave. *Scientific programming language*. Aug. 2022. URL: https://www.gnu.org/software/octave/index.

[Pah+20] Markus Pahlow et al. "Optimality-based non-redfield plankton–ecosystem model (opem v1.1) in uvic-escm 2.9 – part 1: implementation and model behaviour". In: (Oct. 2020). URL: https://noa.gwlb.de/rsc/thumbnail/cop_mods_00054178.png.

[Pah20] Markus Pahlow. *Uvic-updates-opem: optimality-based plankton ecosystem model (opem v1.0) for the uvic-escm*. This work is published under the GNU General Public License 3 (GPL-3.0). You may copy, distribute and modify the software as long as you track changes/dates of in source files and keep modifications under GPL. You can distribute your application using a GPL library commercially, but you must also provide the source code." Kiel, Germany, Jan. 2020. DOI: 10.3289/SW_1_2020. URL: https://portal.geomar.de/metadata/numericModel/show/354384.

[Pah22] Markus Pahlow. "Oppla optimality-based plankton ecosystem model in 1d". In: (Aug. 2022).

[PS16] J. Piwonski and T. Slawig. "Metos3d: the marine ecosystem toolkit for optimization and simulation in 3-d – part 1: simulation package v0.3.2". In: *Geoscientific Model Development* 9.10 (2016), pp. 3729–3750. DOI: 10.5194/gmd-9-3729-2016. URL: https://gmd.copernicus.org/articles/9/3729/2016/.

[PSy22] PSyclone. *Psyclone, the psy code generator, is being developed for use in finite element, finite volume and finite difference codes.* Aug. 2022. URL: https://psyclone.readthedocs.io/en/stable/.

[Qua22] QualCoder. *Qualcoder is a qualitative data analysis application written in python3*. Aug. 2022. URL: https://github.com/ccbogel/QualCoder.

[r-p22] r-project. *R is a language and environment for statistical computing and graphics. it is a gnu project which is similar to the s language and environment which was developed at bell laboratories (formerly at&t, now lucent technologies) by john chambers and colleagues.* Aug. 2022. URL: https://www.r-project.org/about.html.

[RW14] Bernhard Rumpe and Ingo Weisemöller. "A domain specific transformation language". In: *CoRR* abs/1409.2309 (2014). arXiv: 1409.2309. URL: http://arxiv.org/abs/1409.2309.

[Sán+18] Renáta Sándor et al. "The use of biogeochemical models to evaluate mitigation of greenhouse gas emissions from managed grasslands". In: *Science of The Total Environment* 642 (2018), pp. 292–306. ISSN: 0048-9697. DOI: https://doi.org/10.1016/j.scitotenv.2018.06.020. URL: https://www.sciencedirect.com/science/article/pii/S0048969718320837.

[Sch+07] M. Schartau et al. "Modelling carbon overconsumption and the formation of extracellular particulate organic carbon". In: *Biogeosciences* 4.4 (2007), pp. 433–454. DOI: 10.5194/bg-4-433-2007. URL: https://bg.copernicus.org/articles/4/433/2007/.

[SØ22] Peter Sestoft and Kasper Østerbye. "Domain-specific multimodeling". In: (June 2022).

[Swi22] Swissmeteo. *Compiler toolchain to enable generation of high-level dsls for geophysical fluid dynamics models*. Aug. 2022. URL: https://github.com/MeteoSwiss-APN/dawn.

[Tei+14] Jürgen Teich et al. "Exastencils: advanced stencil-code engineering". In: *inSIDE* 12 (Jan. 2014).

[Tom22] Federico Tomassetti. *The complete guide to (external) domain specific languages*. Aug. 2022. URL: https://tomassetti.me/domain-specific-languages/.

[vi22] vi. *Vi is a screen-oriented text editor originally created for the unix operating system.* Aug. 2022. URL: https://www.google.com/search?channel=fs&client=ubuntu&q=Vi+editor.

[VT63] J. Von Neumann and A.H. Taub. *Collected works*. Bd. 1. Pergamon Press, 1963. ISBN: 9780080095660. URL: https://books.google.de/books?id=32DeAQAACAAJ.

[Wik22] Wikipedia. *Biogeochemistry is the scientific discipline that involves the study of the chemical, physical, geological, and biological processes and reactions that govern the composition of the natural environment (including the biosphere, the cryosphere, the hydrosphere, the pedosphere, the atmosphere, and the lithosphere).* Aug. 2022. URL: https://en.wikipedia.org/wiki/Biogeochemistry.

[Wol22] Wolfram. *The world's definitive system for modern technical computing*. Aug. 2022. URL: https://www.wolfram.com/mathematica/.

[Xia02] Meng Xiannong. *Types of models*. Oct. 2002. URL: https://www.eg.bucknell.edu/~xmeng/Course/CS6337/Note/master/node8.html.

[Xte22] Xtext. *Language engineering for everyone!* https://www.eclipse.org/Xtext/. Accessed: 2021-12-06. 2022. URL: https://www.eclipse.org/Xtext/.