Model-Driven Software Engineering for Computational Science Applied to a Marine Ecosystem Model

Arne N. Johanson

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2015

Kiel Computer Science Series (KCSS) 2016/3 v1.0 dated 2016-03-14

ISSN 2193-6781 (print version)
ISSN 2194-6639 (electronic version)

Electronic version, updates, errata available via https://www.informatik.uni-kiel.de/kcss

Published by the Department of Computer Science, Kiel University

Software Engineering Group

Please cite as:

▻ Arne N. Johanson. *Model-Driven Software Engineering for Computational Science Applied to a Marine Ecosystem Model.* Number 2016/3 in Kiel Computer Science Series. Department of Computer Science, 2016. Dissertation, Faculty of Engineering, Kiel University.

```
@book{johanson2016,
  author    = {Arne N. Johanson},
  title     = {Model-Driven Software Engineering for Computational Science
               Applied to a Marine Ecosystem Model},
  publisher = {Department of Computer Science, Kiel University},
  year      = {2016},
  number    = {2016/3},
  series    = {Kiel Computer Science Series},
  note      = {Dissertation, Faculty of Engineering, Kiel University}
}
```

About this Series

The Kiel Computer Science Series (KCSS) covers dissertations, habilitation theses, lecture notes, textbooks, surveys, collections, handbooks, etc. written at the Department of Computer Science at Kiel University. It was initiated in 2011 to support authors in the dissemination of their work in electronic and printed form, without restricting their rights to their work. The series provides a unified appearance and aims at high-quality typography. The KCSS is an open access series; all series titles are electronically available free of charge at the department's website. In addition, authors are encouraged to make printed copies available at a reasonable price, typically with a print-on-demand service.

Please visit `http://www.informatik.uni-kiel.de/kcss` for more information, for instructions how to publish in the KCSS, and for access to all existing publications.

1. Gutachter:	Prof. Dr. Wilhelm Hasselbring Christian-Albrechts-Universität Kiel
2. Gutachter:	Prof. Dr. Andreas Oschlies GEOMAR – Helmholtz-Zentrum für Ozeanforschung Kiel

Datum der mündlichen Prüfung: 4. März 2016

Abstract

This interdisciplinary thesis contributes to both software engineering and ecological modeling. In the discipline of software engineering, we introduce *Sprat*, which is a model-driven software engineering approach for computational science. In the field of ecological modeling, we present the *Sprat Marine Ecosystem Model*—a spatially-explicit fish stock model for marine end-to-end modeling.

The ever-increasing complexity of *in silico* experiments in computational science is reflected in the growing complexity of the simulation software enabling these experiments. This development results in a need for collaboration between scientists from different disciplines in the development of such elaborate scientific software. However, in this process, state-of-the-art software engineering methods are rarely employed, which negatively affects the maintainability and performance of the software as well as the reliability of its results. To tackle this challenge, we introduce the Sprat Approach, which hierarchically integrates multiple domain-specific languages to facilitate the cooperation of scientists from different disciplines and to support them in creating well-engineered software without extensive software engineering training. In order to evaluate the Sprat Approach, we apply it to the implementation of the Sprat Marine Ecosystem Model in an exploratory case study.

The Sprat Marine Ecosystem Model is a fish stock model that is coupled with biogeochemical ocean models to simulate all trophic levels of a marine ecosystem. The model utilizes a novel modeling approach based on population balance equations that combines the advantages of existing end-to-end modeling frameworks while preventing their main drawbacks. For solving the partial differential equations that constitute the Sprat Model, we develop a flux-corrected transport finite element scheme that uses explicit multi-step methods to integrate in time. In order to evaluate the Sprat Model, we apply it to the eastern Scotian Shelf ecosystem with its intertwined direct and indirect fish stock interactions, which previously could not be modeled satisfactorily. Our simulation results provide new insights into the main drivers of regime shifts in marine ecosystems.

Zusammenfassung

Diese interdisziplinäre Arbeit leistet Beiträge sowohl in der Softwaretechnik als auch in der ökologischen Modellierung. In der Disziplin der Softwaretechnik führen wir mit *Sprat* einen modellgetriebenen Entwicklungsansatz speziell für das wissenschaftliche Rechnen ein. In dem Gebiet der ökologischen Modellierung präsentieren wir mit dem *Sprat Marine Ecosystem Model* ein räumlich explizites Modell für Fischpopulationen im Kontext der *end-to-end* Modellierung von marinen Ökosystemen.

Die steigende Komplexität von *in silico* Experimenten in den Naturwissenschaften spiegelt sich in einer beständig größer werdenden Komplexität der Simulationssoftware, die diese Experimente ermöglicht. Aufgrund dieser Entwicklung besteht zusehends die Notwendigkeit, dass NaturwissenschaftlerInnen verschiedener Disziplinen kollaborieren, um derart komplexe Simulationssoftware entwickeln zu können. Da die WissenschaftlerInnen in diesem Entwicklungsprozess jedoch nur selten moderne Methoden der Softwaretechnik verwenden, wird die Wartbarkeit und Performance der Software sowie die Zuverlässigkeit der mit ihrer Hilfe erzeugten Ergebnisse negativ beeinflusst. Dieser Herausforderung begegnen wir mit unserem Sprat-Ansatz, der verschiedene domänenspezifische Sprachen hierarchisch miteinander integriert. Unser Entwicklungsansatz vereinfacht die Kooperation von NaturwissenschaftlerInnen aus verschiedenen Disziplinen und unterstützt sie darin, qualitativ hochwertige Software zu erstellen, ohne eine umfassende Softwaretechnik-Ausbildung absolvieren zu müssen. Um den Sprat-Ansatz zu evaluieren, setzen wir ihn im Rahmen einer explorativen Fallstudie für die Implementierung des *Sprat Marine Ecosystem Model* ein.

Das *Sprat Marine Ecosystem Model* ist ein Bestandsmodell für Fische, welches mit biogeochemischen Ozeanmodellen gekoppelt wird, um alle trophischen Ebenen eines Ökosystems abbilden zu können. Das Modell verwendet einen neuartigen Modellierungsansatz, der auf Populationsbilanzgleichungen aufbaut. Dieser Ansatz vereinigt die Vorzüge von existierenden *end-to-end* Modellierungsansätzen und vermeidet ihre wesentlichen Nachteile. Um die partiellen Differentialgleichungen des Sprat-Modells zu lösen, entwickeln wir eine *flux-corrected transport* Finite-Element-Methode,

die explizite Mehrschrittverfahren für die Zeitintegration verwendet. In dieser Arbeit evaluieren wir das Sprat-Modell, indem wir es für das östliche Scotian Shelf Ökosystem mit seinen verwobenen direkten und indirekten Interaktionen parametrisieren, die vormals nicht zufriedenstellend modelliert werden konnten. Unsere Simulationsergebnisse bieten neue Erkenntnisse über Regimeumbrüche in marinen Ökosystemen.

Preface

by Prof. Dr. Wilhelm Hasselbring

In science and research, we observe an increasing use of software. Often scientific experiments take place in virtual research environments with computer-based simulations of real-world phenomena. To efficiently enable such scientific work, specific software support is required. Each scientific discipline has its own distinct requirements, and often develops very specific solutions.

Traditionally, scientific computing employs no or only few software engineering methods and techniques, while developing large simulation software systems. Particularly for community software with a long life span, it would be beneficial to have a well-structured, modular software architecture in accordance with basic software engineering principles.

To address the resulting challenges, Arne Johanson invents the so-called Sprat Approach for scientific computing based on hierarchically structured domain-specific languages. Its fundamental idea is to facilitate the interdisciplinary collaboration of scientists from different disciplines, without the need for these scientists to be trained in software engineering extensively.

Arne's thesis does not only support interdisciplinary research, the thesis itself is an interdisciplinary contribution to two scientific disciplines, namely software engineering and ecological modeling. Besides the conceptual work, this dissertation contains a significant experimental part and a multifaceted empirical evaluation, based on a high-quality implementation of the Sprat domain-specific languages.

This thesis is a good read and I recommend it to anyone interested in interdisciplinary research.

Wilhelm Hasselbring
Kiel, March 2016

Contents

List of Acronyms

ACM	Association for Computing Machinery
ADR	Advection-Diffusion-Reaction
AMR	Active Metabolic Rate
ANOVA	Analysis of Variance
API	Application Programming Interface
AST	Abstract Syntax Tree
BLAS	Basic Linear Algebra Subprograms
CAD	Computer-Aided Design
CG	Conjugate Gradient
CPU	Central Processing Unit
CRS	Compressed Row Storage
DEVS	Discrete Event System Specification
DSL	Domain-Specific Language
DSO	Domain-Specific Optimization
DoF	Degree of Freedom
EBNF	Extended Backus-Naur Form
EMF	Eclipse Modeling Framework
EMRQ	Ecological Modeling Research Question
EMWP	Ecological Modeling Work Package

List of Acronyms

FCT	Flux-Corrected Transport
FEM	Finite Element Method
FLOPS	Floating Point Operations Per Second
GPL	General-Purpose Language
GQM	Goal/Question/Metric
HPC	High Performance Computing
IBM	Individual-Based Model
ICES	International Council for the Exploration of the Sea
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
IPCC	Intergovernmental Panel on Climate Change
ISO	International Organization for Standardization
IT	Information Technology
JVM	Java Virtual Machine
LIL	List of Lists
LOC	Lines of Code
LOWESS	Locally Weighted Scatterplot Smoothing
MBE	Model-Based Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development

MDE Model-Driven Engineering

MDSE Model-Driven Software Engineering

MOF Meta-Object Facility

MPA Marine Protected Area

MPI Message Passing Interface

MPS Meta Programming System

NAFO Northwest Atlantic Fisheries Organization

NASA National Aeronautics and Space Administration

NATO North Atlantic Treaty Organization

NetCDF Network Common Data Form

NOAA National Oceanic and Atmospheric Administration

NPZ Nutrient, Phytoplankton, Zooplankton

OCL Object Constraint Language

PAR Photosynthetically Active Radiation

PDE Partial Differential Equation

ROMS Regional Ocean Modeling System

SERQ Software Engineering Research Question

SEWP Software Engineering Work Package

SI International System of Units

SMR Standard Metabolic Rate

SPE Software Performance Engineering

SPMD Single Program, Multiple Data

List of Acronyms

SSH	Secure Shell
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

This interdisciplinary thesis is situated at the boundary of software engineering and ecological modeling. To each of these fields of study, a distinct contribution is made:

1. In the discipline of software engineering, we introduce the *Sprat Approach*, which is a model-driven software engineering approach for computational science that facilitates the cooperation of scientists from different disciplines and supports them in creating well-engineered software.

2. To the field of ecological modeling, we contribute the *Sprat Marine Ecosystem Model*—a spatially-explicit fish stock model that is coupled with existing biogeochemical ocean models to simulate all trophic levels of a marine ecosystem.

While the two disciplines that our contributions belong to are quite different, the contributions are, nonetheless, reciprocally related to each other. The engineering of the implementation of the Sprat Marine Ecosystem Model is used for evaluating the Sprat Approach, from which valuable lessons for software engineering in computational science can be learned. Conversely, from the perspective of ecological modeling, the Sprat Approach lays the foundation for a well-engineered technical implementation of the Sprat Marine Ecosystem Model that is easily accessible also to scientists who have little programming experience.

In this introductory chapter, Section 1.1 motivates our research and points out the key challenges to be overcome. After summarizing our approach and our core contributions in Section 1.2, we outline the structure of the thesis in Section 1.3.

1.1 Motivation and Challenges

Computation has established itself as a *third paradigm* (Bell et al. 2009) for scientific discovery, next to theory and experimentation. Via computational models and simulation, it allows to investigate domains that have been largely inaccessible to traditional approaches, such as the evolution of the universe or the prediction of climate change. The discipline that studies the application of computation to problems from science and engineering is called *computational science* or, synonymously, *scientific computing*. The large number of application domains for computational science makes it a very diverse field that runs orthogonal to traditional discipline boundaries: it ranges from biologists implementing small-scale data analysis procedures in scripting languages to numerical mathematicians prototyping their algorithms to interdisciplinary groups of scientists developing large-scale climate simulations for high-end computing hardware.

About 10 years ago, Post and Votta (2005) found that, in spite of already yielding important results, the relatively new discipline of computational science was still "troublingly immature." In their much-noted article, they came to the conclusion that for fulfilling the potential of computational science, three major challenges had to be overcome:

1. *Performance challenge*—develop high-performance computers that can handle multi-scale models with extreme fidelity.
2. *Prediction challenge*—create truly predictive computational models.
3. *Programming challenge*—implement high-quality simulation software for complex computer systems.

Today, with the advent of heterogeneous computer architectures with many cores and low-latency, high-bandwidth interconnects, the performance challenge seems to be mostly met (Kogge and Shalf 2013). Additionally, cloud computing—although not yet fully mature for this purpose—promises to make vast compute resources available to the average researcher (Galante et al. 2014; Iosup et al. 2011).

Concerning the prediction challenge, there still remain open questions regarding how to integrate processes on different spatial and temporal scales, especially for the prediction of climate change. These questions are

being actively addressed, for example, by initiatives like the climate model intercomparison conducted continuously by the IPCC (Flato et al. 2013).

The third challenge of implementing computational models with high quality on complex computer systems has immediately found resonance in the software engineering community (Carver 2009; Kelly 2007; Wilson 2006b). It became clear that most scientists—who usually develop their simulation software by themselves and are often self-taught programmers—are not actually "up to the task" of implementing such complex simulations (Merali 2010). This is, of course, a big concern because it questions the credibility of simulation results, which—as with climate change—are sometimes even supposed to serve as a basis for global policy making.

Most attempts at a solution to this problem suggested by the software engineering community follow one of two lines:

1. Have software engineers build or re-engineer the simulation software for the scientists.
2. Educate scientists to enable them to use state-of-the-art software engineering methods for building the simulation software themselves.

Quite quickly, it became apparent that the first approach is not practicable because the requirements for the software to be developed are not known up front but emerge only during the course of development. After all, the software is deeply embedded into the scientific discovery process and is in itself something to be experimented with (Easterbrook and Johns 2009). Therefore, the scientists cannot instruct others to develop the simulation software for them but have to be able to implement and maintain it by themselves.

The second attempt at a solution—training scientists in software engineering—resulted in several workshop-based education programs mainly targeted at Ph.D. students. Examples for such programs are Greg Wilson's Software Carpentry (Wilson 2006a, 2014) and the Argonne Training Program on Extreme-Scale Computing (ATPESC) (Messina 2015). Even if the individual workshops are positively evaluated (Aranda 2012), their overall impact on the computational science community as a whole remains unclear. Additionally, the strategy of educating scientists in software engineering is complicated by the "accidental complexity" that is introduced along with software engineering methods. The fact that scientists often experience software engineering techniques as a burden rather than a relief, lets even

Wilson conclude: "education [...] won't be enough on its own" (Wilson 2006b).

As it seems undesirable and futile to try to turn scientists into software engineers—after all, they are busy enough keeping up with their quickly evolving scientific fields—, in this thesis, we propose to *adapt* existing software engineering approaches to meet the specific needs of the computational science community. To be fruitful, such adaptations must take into account the specific characteristics of the target community and prevent the introduction of unwanted, accidental complexities. We thereby provide a "third way" between the two aforementioned attempts at a solution to help to finally meet the programming challenge that is still largely unresolved (newer mentions of this deficiency include Brown et al. 2015; Carver and Epperly 2014; Joppa et al. 2013).

To evaluate our engineering approach—which is introduced in Section 1.2—, we focus on the implementation of a complex, newly designed marine ecosystem model. The need for such a model is motivated in the following.

1.1.1 End-To-End Marine Ecosystem Models

Living marine resources and their exploitation by fisheries play a key role in sustaining global nutrition but many of the world's fish stocks are in poor condition due to overharvesting (Myers and Worm 2003; Worm et al. 2009). This reduces the productivity of the stocks significantly and makes management of fisheries necessary in order to achieve a sustainable use of this natural resource.

Fishing, however, is not the only impact on the condition and productivity of fish stocks but long-term variability of environmental parameters due to climate change imposes additional pressures (Brander 2007). The effects of changes in the environment on fish can be direct (e. g., by altering individual growth rates) or indirect (by affecting the net primary productivity and, thus, the carrying capacity of the ecosystem) and often they interact with anthropogenic influences in complex, unforeseeable ways. For example, the expansion of oxygen minimum zones in the tropical northeast Atlantic Ocean due to climate change compresses the suitable habitat of pelagic predator fish to a narrow surface layer and, thus, increases their vulnerability to fishing gear (Stramma et al. 2012). The resulting high catch

rates in such areas can lead to overly optimistic estimates of species abundance and, therefore, to exaggerated fishing quotas that put the affected stocks in danger of overexploitation.

From such examples, it becomes apparent that fisheries management must address fishing and climate change jointly in order to acknowledge that these two pressures on the productivity of stocks are highly interconnected. However, current management practices, which rely mostly on a single-species approach, appear to be rather unsuitable for this task (Rose et al. 2010). A more holistic ecosystem-based approach is needed that focuses on marine ecosystems as a whole and takes into account the interdependence of their components (Cury et al. 2008). Such management strategies critically depend on predictions of how the interactions within an ecosystem affect its overall health and, in particular, the condition of the fisheries supported by that system.

Ecological models that can supply this kind of information are called *end-to-end models* because they incorporate all ecosystem components from the dynamics of the abiotic environment to primary producers to top predators (Travers et al. 2007). In such models, the different elements of the ecosystem are linked together mainly through trophic interactions—i. e., by feeding (Moloney et al. 2011). Ideally, all these links between components are modeled bidirectionally (e. g., an increase in fish biomass due to feeding on zooplankton is reflected in a decrease of the latter). Such a two-way coupling of model elements allows to explicitly resolve, at the same time, both bottom-up and top-down control mechanisms in an ecosystem. It is the combination of modeling these bidirectional links in the trophic structure and considering the dynamics of the environment that enables end-to-end models to provide long-term predictions for the development of fish stocks under the influence of environmental change. In the context of ecosystem-based fisheries management, these predictive capabilities can be used to evaluate different management scenarios—such as different marine protected areas and fishing quotas—with regard to their long-term effectiveness (Stock et al. 2011).

In practice, end-to-end models are typically constructed by using existing physical and biogeochemical ocean models (for the abiotic environment as well as for nutrient and plankton dynamics) and creating a fish model that can be coupled with these models (Shin et al. 2010). Implementing a whole end-to-end model from scratch is discouraged by the amount of effort that

is needed for developing sophisticated physical and biogeochemical models. But even with existing ocean models, the development and integration of a fish stock model poses serious challenges:

1. Space must be resolved explicitly by the fish model in order to take into account local changes in the environmental conditions (Cury et al. 2008). Being spatially-explicit requires the model to represent very large amounts of fish efficiently.
2. The fish model has to integrate well with the mathematical framework of existing ocean models. In particular, data exchange must not cause considerable overhead.
3. The number of free model parameters must be low enough to be able to reliably constrain these parameters with existing observations (Stock et al. 2011). Therefore, it is particularly important that as many parameters as possible have a clear biological meaning and can be measured by observing individual fish.

In this thesis, we introduce a new kind of fish stock model that is based on population balance equations (Ramkrishna 2000) and addresses all challenges named above at once. To evaluate our model, we apply it to simulate and mechanistically explain the complex interdependent trophic and climatic interactions between the different components of the eastern Scotian Shelf ecosystem with its commercially important but overexploited fish stocks.

1.2 Overview of Approach and Contributions

This thesis introduces a software development approach called *Sprat* that is based on Model-Driven Software Engineering (MDSE) techniques. The approach utilizes hierarchically integrated Domain-Specific Languages (DSLs) to facilitate the collaboration of scientists from different disciplines in the development of complex simulation software.[1] Each of the DSLs caters to the needs of domain experts from one of the scientific disciplines that are involved in the development project. Through the hierarchical integration of the different modeling languages, we achieve a clear separation of concerns

[1] For an introduction to MDSE and DSLs, see Chapter 2.

among the disciplines. This clear separation together with the intuitive and concise syntax enabled by the DSLs helps scientists to efficiently produce maintainable, reliable, portable, and performant software without the need for extensive software engineering training. By focusing on the scientists' productivity and on their ability to freely experiment with their software, Sprat prevents unnecessary complexities that are usually associated with software engineering approaches in computational science and ensures its acceptance in the target user community.

In the context of end-to-end ecosystem modeling, we present the Sprat Marine Ecosystem Model, which introduces fish into existing biogeochemical ocean models. In the Sprat Model, fish are described by density functions on a combined space-body size domain. The evolution of these densities is governed by population balance equations (Ramkrishna 2000), which are a special type of Partial Differential Equations (PDEs). Using a modeling approach based on PDEs, the Sprat Model integrates well with existing ocean models (which are also PDE-based). Furthermore, the great majority of the parameters needed for the application of the Sprat Model is observable in individual fish, which makes it possible to constrain these parameters with existing data readily available from databases such as FishBase (Froese and Pauly 2015).

Through its efficient parametrization and integration with biogeochemical ocean models, the Sprat Marine Ecosystem Model is able to evaluate fisheries management strategies on a long-term basis by taking into consideration the changing physical environment. In addition to that, the model can be used to study and mechanistically explain the impact of interconnected environmental and anthropogenic pressures on fish stocks.

In the following sections, we briefly summarize the core contributions of this thesis that go beyond the Sprat Approach and the Sprat Marine Ecosystem Model themselves.

1.2.1 Software Engineering in Computational Science

We conducted a literature review to assess the current state of software engineering practices in computational science. In this review, we identified 13 key characteristics of scientific software development that have to be taken into account when adapting software engineering techniques for this

community. The characteristics can be divided into three groups depending on whether they result from

1. the nature of scientific challenges,
2. the limitations of computers, or
3. the cultural environment of computational science.

Our detailed review allowed us to identify some shortcomings of existing suggested solutions to raise the productivity and reliability in scientific software development. Furthermore, we were able to show that, given the specific characteristics of the computational science community, MDSE approaches are a promising starting point for adapting software engineering approaches for this community.

1.2.2 Hierarchies of Domain-Specific Languages

In order to be able to make MDSE and DSLs fruitful for computational science, we introduced and formalized the concept of *hierarchies of DSLs*. The foundation for this concept is provided by an investigation of the architectural design of scientific simulation software in which we demonstrate that such software can typically be (and often is) implemented using a layered architecture. The boundaries of these layers run along the boundaries of the different scientific (sub-)disciplines involved in the development project, which enables a hierarchical compartmentalization of the whole software system according to these disciplines. This hierarchical segmentation of the software forms the basis for the Sprat Approach as it allows to assign a single DSL to every scientific discipline involved and to integrate these languages with each other in a straightforward hierarchical fashion.

1.2.3 DSLs for a Marine Ecosystem Model

We selected and developed DSLs for the implementation of PDE-based marine ecosystem models. Two of these languages, namely the Sprat PDE DSL and the Sprat Ecosystem DSL, were designed by us.

1. The Sprat PDE DSL concentrates on the implementation of mesh-based PDE solvers. It is implemented as an internal language embedded into

C++ with a focus on combining performance with a natural and concise syntax.

2. The external Sprat Ecosystem DSL allows to specify ecosystem simulations in a declarative way. The syntax of the language as well as the accompanying tooling is designed with users in mind who have little or no programming experience.

For each of the two DSLs, a publicly-available implementation and a detailed description of the meta-model are provided.

1.2.4 Population Balances for Marine End-To-End Modeling

We examined existing modeling frameworks that are currently used in marine end-to-end modeling to identify their advantages and disadvantages. Based on this analysis, we adapted population balance equations for the purpose of marine end-to-end modeling in a way that combines most of the advantages of the other modeling frameworks while avoiding their disadvantages. In particular, the most important benefits of our adaptation of population balance models are that

1. their parameters are observable in individual fish,
2. they offer a dynamic food web structure that is an emergent property of the model,
3. there is a well-established mathematical theory for approximating the solution of population balance models with a guaranteed accuracy, and
4. they integrate well with existing biogeochemical ocean models.

1.2.5 Designing a PDE-Based Fish Model

To put our population balance approach for marine end-to-end modeling into practice, we developed the Sprat Marine Ecosystem Model—a fish stock model based on population balance equations that can be coupled with existing biogeochemical ocean models. In addition to the Sprat Model, we constructed a simple Nutrient, Phytoplankton, Zooplankton (NPZ) model that can be integrated with the fish model to operate the latter without the

need for a fully-developed biogeochemical ocean model. A key contribution in developing the Sprat Model is the selection of relevant biological processes to be represented in a spatially-explicit, PDE-based fish model with a focus on the individual fish and the description of how these processes are to be modeled.

1.2.6 An Explicit Flux-Corrected Transport Solver

For approximating the solution of the Sprat Model, we developed a Flux-Corrected Transport (FCT) Finite Element Method (FEM) solver that uses explicit multi-step methods to integrate the solution in time. In contrast to other numerical methods for PDEs, such as finite differences, the modern FEM has the advantage of guaranteeing good approximation results even without high regularity of the solution and it allows to use unstructured meshes. However, the standard Galerkin FEM is unstable for the kind of equations that the Sprat Model is based upon and introduces spurious oscillations into the solution. We demonstrated that our FCT FEM solver overcomes these problems and produces ripple-free solutions with high accuracy.

1.2.7 Evaluation

We evaluated the Sprat Approach by applying it to the engineering of the Sprat Marine Ecosystem Model implementation. To judge the quality of the DSLs that we developed in this process, we conducted an online survey with embedded controlled experiments, benchmarking studies, and expert interviews.

In order to evaluate the Sprat Marine Ecosystem Model, we parametrized the model for the eastern Scotian Shelf region and performed several simulation experiments, including a sensitivity analysis of the model. With our simulations, we identified the main drivers of the fish stock dynamics on the eastern Scotian Shelf and evaluated possible management scenarios to prevent the collapse of the commercially valuable benthic predator stocks in that area. Furthermore, we coupled the Sprat Model with an existing biogeochemical ocean model to study the spatial distribution of fish in the Scotian Shelf region.

1.3 Document Structure

The remainder of this thesis is structured as follows:

Part I is concerned with the foundations of this thesis.

- ***Chapter 2*** discusses the fundamentals of MDSE and DSLs.
- ***Chapter 3*** introduces traditional models for fish stock prediction.
- ***Chapter 4*** covers the basics of the Finite Element Method (FEM).

Part II is dedicated to the description of the Sprat Approach—our model-driven software engineering approach for computational science.

- ***Chapter 5*** summarizes our research design for the software engineering part of this thesis.
- ***Chapter 6*** identifies the specific characteristics of software engineering in computational science and gives reasons why model-driven approaches are promising starting points for adapting software engineering techniques for computational science.
- ***Chapter 7*** presents the Sprat Approach based on the concept of hierarchies of DSLs.
- ***Chapter 8*** details the application of the Sprat Approach to the implementation of the Sprat Marine Ecosystem Model, including a description of the DSLs employed for this purpose.

Part III covers the construction of the Sprat Marine Ecosystem Model—our population balance model for marine end-to-end modeling.

- ***Chapter 9*** summarizes our research design for the ecological modeling part of this thesis.
- ***Chapter 10*** introduces our novel modeling approach based on population balance equations and compares it with existing modeling frameworks.
- ***Chapter 11*** describes the Sprat Marine Ecosystem Model as well as a simple NPZ model that can be integrated with the Sprat Model.
- ***Chapter 12*** presents the FCT FEM solver that is used to numerically approximate the solution of the Sprat Model.

Part IV is concerned with the evaluation of the Sprat Approach and of the Sprat Model.

Chapter 13 presents results from expert interviews and benchmarking studies assessing the Sprat PDE DSL.

Chapter 14 discusses results from an online survey with embedded controlled experiments for the evaluation of the Sprat Ecosystem DSL.

Chapter 15 covers the application of the Sprat Model to mechanistically explain the fish stock dynamics on the eastern Scotian Shelf.

Chapter 16 describes results from coupling the Sprat Model with an existing biogeochemical ocean model for the eastern Scotian Shelf.

Chapter 17 discusses related work.

Part V presents overall conclusions and lessons learned (***Chapter 18***) as well as an outlook on future work (***Chapter 19***).

Part I

Foundations

Chapter 2

Model-Driven Software Engineering

This chapter covers the fundamental concepts of Model-Driven Software Engineering (MDSE) in Section 2.1 and those of Domain-Specific Languages (DSLs) in Section 2.2. Additionally, Section 2.3 introduces language workbenches, which are integrated tools that facilitate the development of DSLs and their infrastructure. Specifically, we discuss Xtext and JetBrains MPS, which are language workbenches used in the context of this thesis.

The brief overview given in this chapter is mainly based on the works of Brambilla et al. (2012), Fowler (2010), and Stahl and Völter (2006). For a more detailed discussion of MDSE and DSLs, we refer to these textbooks.

2.1 Model-Driven Methods

Models have always played a critical role in computer science in general and software engineering in particular because "the reality of the applications of information technology systems is, at every point in time, accessible only through models" (Mahr 2009, translation by the author). A well-known heuristic characterization of the general notion of a *model* is given by Stachowiak (1973), who names three defining features:

Mapping feature: a model is always a representation of a (natural or artificial) original individual.

Reduction feature: a model only captures a subset of the attributes of the represented original.

Pragmatic feature: a model represents an original for a specific purpose.

Note that throughout this thesis the term *model* is applied with different meanings depending on whether it is used in the context of software engineering or ecological modeling. Software engineers construct models of software to reduce the complexity of existing or planned software systems and thereby make them accessible to human reasoning in the first place. In contrast to this, ecological modelers consider models to be mathematically formulated abstractions of real-world ecosystems.

Software engineering methods that treat models as "first-class citizens" in the engineering process are called *model-driven*. By focusing on modeling in this way, these methods aim at, among other things, increasing productivity and quality. However, there are different interpretations of how this is to be accomplished and, therefore, different concrete branches of model-driven methods, which form subsets of one another:

Model-Based Engineering (MBE) methods are the most general ones. For them, models play an important role (e. g., in the analysis phase) but are in general not itself implementation artifacts (i. e., they are neither executed nor is code generated from them).

Model-Driven Engineering (MDE) uses models as the primary artifact in every stage of the software life cycle (meaning that models are also implementation artifacts). Therefore, MDE forms a subset of the MBE paradigm.

Model-Driven Development (MDD) is more specific than MDE as it employs models as the primary artifact only for the development process.

Model-Driven Architecture (MDA) is a concrete MDD approach with standardized modeling and transformation languages by the Object Management Group (2015b).

In this thesis, we use the term Model-Driven Software Engineering (MDSE) throughout to explicitly state that we are concerned only with the engineering of software and to convey that we focus on modeling for all aspects of the software life cycle.

2.1.1 Modeling Languages and Transformations

The main concepts of MDSE are *models*, which are expressed in *modeling languages*, and *transformations*. Transformations map a source model to a target

artifact which can either be a model again (*model-to-model* transformation) or an arbitrary textual artifact (*model-to-text* transformation).

Note that MDSE, unlike round-trip engineering, is a strict forward engineering approach (though it can also be used for reverse engineering purposes). There is no need for synchronizing models and other implementation artifacts because the models already *are* implementation artifacts. This implies that the modeling languages used in the context of MDSE have to be executable either by means of *interpretation* or through *generation* of source code from the model. If source code is generated from a model, changes are only applied to the model and the generated code is modified solely by means of re-generating it.

Some authors, like Brambilla et al. (2012), differentiate between two classes of modeling languages, namely *domain-specific* and *general-purpose modeling languages*. While the former are designed particularly for a certain application domain or context, the latter can be applied to modeling purposes in any domain. Others, like Stahl and Völter (2006), view the concept of MDSE as always being linked with the use of domain-specific modeling languages. They adopt this stance because they attribute the gains in productivity promised by MDSE to the domain-specificity of the applied modeling languages. Since for them, there is no need to contrast such languages with general-purpose *modeling* languages, they just speak of Domain-Specific Languages (DSLs). In this thesis, we adopt the perspective of Stahl and Völter (2006) and, accordingly, refer to such languages as DSLs.

2.2 Domain-Specific Languages

Like General-Purpose Languages (GPLs), such as C or Java, Domain-Specific Languages (DSLs) are programming languages. However, unlike GPLs, which are designed to be able to implement any program that can be computed with a Turing machine, DSLs limit their expressiveness to a particular application domain. By featuring high-level domain concepts that enable to model phenomena at the abstraction level of the domain and by providing a notation close to the target domain, DSLs can be very concise. A popular example of a DSL are regular expressions that target the domain of text pattern matching and allow to model search patterns independently of any concrete matching engine implementation.

DSLs can be *textual*, *graphical*, or use a syntax that combines both textual and graphical elements (which we call *semi-graphical*). In accordance with Völter (2013), we refer to DSL artifacts as *models*, *code*, and *programs* interchangeably, regardless of the concrete syntax of the language.

Another distinction among DSLs is the one between *external* and *internal* languages. Internal or *embedded* DSLs are implemented as libraries of existing GPLs that convey the impression of being independent languages by using programming techniques such as operator overloading (Czarnecki et al. 2004). An internal DSL can be discerned from a mere Application Programming Interface (API) by the fact that a language does not only provide "vocabulary" (i. e., functions, data types, etc.), as an API typically does, but also provides a grammar (i. e., it prescribes how to combine the language elements to form larger expressions). In contrast to internal DSLs, external ones are completely independent languages with their own syntax designed from scratch.

The increased abstraction level of DSLs and the reduced complexity of their source code is supposed to enhance productivity and quality during the implementation and maintenance of software in comparison to GPLs. While there are not many systematic studies of this claim, the few existing ones support this hypothesis (Kieburtz et al. 1996; Kosar et al. 2012).

2.2.1 Meta-Modeling

As with any other formal language, a DSL is defined by its *concrete* and *abstract syntax* as well as its *static* and *execution semantics*. While the concrete syntax defines the textual or graphical notation elements with which users of the DSL can express models, the abstract syntax of a DSL determines the entities of which concrete models can be comprised. These abstract model entities (abstract syntax) together with the constraints regarding their relationships (static semantics) can again be expressed as a model of all possible models of the DSL, which is therefore called the *meta-model* of the DSL. Since the DSL meta-model embodies the structure of the target domain, it plays a critical role in the field of *language engineering* (Kleppe 2008). Gašević et al. (2009) point out that meta-modeling bears great resemblance to the construction of ontologies and shares some of its inherent difficulties.

The meta-model of a DSL and its entities and relationships can, of course, again be described by a model, which is the *meta-meta-model* of the language

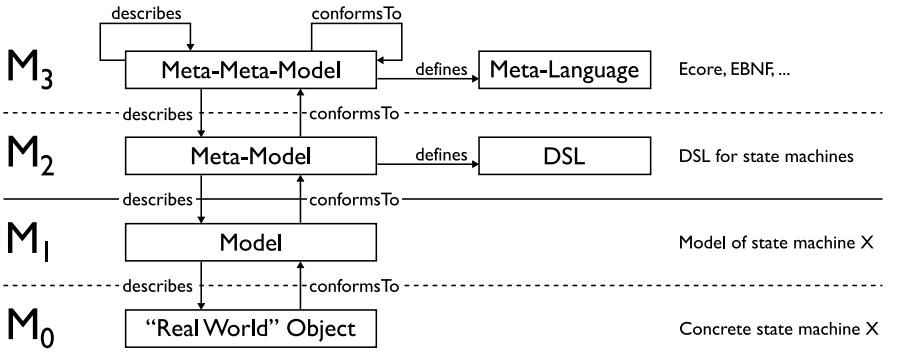


Figure 2.1. The four levels of meta-modeling. Figure adapted from Stahl and Völter (2006).

(see Figure 2.1). Beginning with the "real world" information at the zeroth *meta-level* M_0, each formal abstraction introduces another meta-level. We say that the model on level M_i *describes* or *represents* the model or the information on level M_{i-1} and *conforms to* the model on level M_{i+1}. It is clear that such an abstraction process results in an infinite regress. For practical purposes, this infinite regress is typically handled by choosing a modeling language for level M_3 that is able to describe itself. Examples of such self-describing meta-languages are EBNF (ISO 14977 1996), EMF Ecore (Steinberg et al. 2008), and MOF (Object Management Group 2015a).

The execution semantics of a DSL are defined by the model transformations that are applied to it. A transformation can be described as a mapping from the meta-model of the source DSL to a target. If the DSL is directly executed by means of an interpreter, this target is the set of available machine instructions, which directly constitutes the semantics. If source code is generated from the DSL, the target of the mapping is the meta-model of the target programming language (be it a DSL or a GPL). In this case, the semantics of the source DSL are defined in terms of the semantics of the target language.

2.2.2 Generation and Domain-Specific Platforms

In the context of this thesis, we only consider generation and not interpretation for making DSLs executable. This means that for each *target platform* on which a DSL is supposed to run, a *code generator* must be provided to transform DSL models into source code either in another DSL or a GPL (for internal DSLs, the generation step is implicit). This approach of generating code in existing languages is efficient because it allows to reuse the low-level optimizations of existing GPL compilers while introducing additional high-level *Domain-Specific Optimizations (DSOs)* that are independent of the target platform. For example, the generator of a DSL for matrix-vector expressions could optimize computations based on recurring block structures of sparse matrices. Such optimizations are only possible at the level of the DSL generator and not at the level of a GPL compiler because the latter (necessarily) lacks any domain-specific concepts such as "block structure." Following this approach, generators can be much simpler than GPL compilers but still create even more runtime-efficient applications.

To further reduce generator complexity, Stahl and Völter (2006) propose to not only provide a *technical* but also a *domain-specific platform* as the target for code generation (Figure 2.2). Since such a domain-specific platform already incorporates reusable "semantically-rich" domain components, transformations that target this platform are simplified.

2.3 Language Workbenches

As mentioned above, there is some evidence that DSLs with adequate tool support can increase productivity in the engineering of software. However, the design and implementation of a DSL and corresponding tools, such as Integrated Development Environment (IDE) support, require considerable effort that could potentially offset the productivity gains. *Language workbenches* are tools that intend to reduce the effort of defining, reusing, and composing languages and their IDEs to make DSL development efficient enough to be applied even in small-scale projects (Erdweg et al. 2013). For this purpose, a language workbench will typically assist the DSL developer with the definition of:

1. the meta-model and its representation

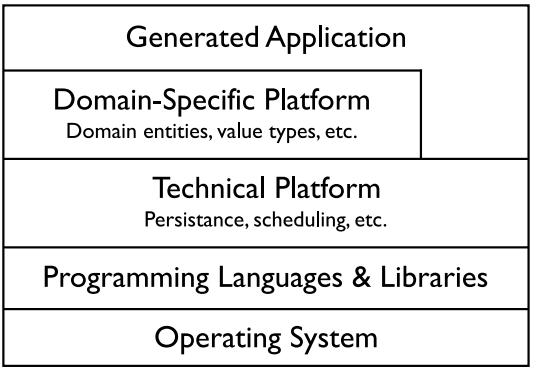


Figure 2.2. Technical and domain-specific target platforms. Figure adapted from Stahl and Völter (2006).

2. an editor that allows to manipulate model instances
3. generators/interpreters that embody the model transformations

Popular language workbenches which are applied in research as well as in industry include Xtext[1] and JetBrains MPS.[2] In the context of this thesis, we use Xtext to implement a DSL for marine ecosystem simulations (see Chapter 8).

2.3.1 Xtext

Xtext is a framework based on Eclipse[3] for developing textual DSLs and re-uses technologies from the Eclipse Modeling Framework (EMF) (Steinberg et al. 2008). It allows to define the meta-model of a DSL by using a grammar specification language similar to EBNF (ISO 14977 1996). From this grammar (which is automatically converted into an Ecore meta-model (Steinberg et al. 2008)), a parser, a code generator stub, and a customizable Eclipse editor with syntax highlighting as well as an autocomplete feature are generated.

[1] https://eclipse.org/Xtext
[2] https://www.jetbrains.com/mps
[3] https://eclipse.org

DSL generators are implemented using Xtend,[4] which is a programming language for the Java Virtual Machine (JVM). Xtend features template expressions with intelligent white space handling and multiple dispatch, which simplifies writing generator templates and traversing the Abstract Syntax Tree (AST) of parsed models. For an introductory text regarding Xtext and Xtend, see Bettini (2013).

2.3.2 JetBrains MPS

JetBrains MPS (Meta Programming System) is an environment for the development of semi-graphical DSLs with features similar to those of Xtext. In contrast to Xtext, however, MPS does not allow to create classical IDEs that are based on text files but uses *projectional editing*. With projectional editing, users do not edit text that is subsequently parsed but directly manipulate an AST representation of the model. The AST is made accessible to the user by rendering (projecting) it using a concrete syntax that consists of fields (i. e., boxes) of styled text, formulas, and graphs, etc. Since all of these boxes "know" which element of the abstract syntax they represent, it is easy to compose different DSLs and extend existing languages with MPS. A drawback of this approach, however, is the potentially cumbersome integration of such editors with tools outside the MPS ecosystem (e. g., revision control systems), which is due to the lack of a genuine textual representation of models.

For an introduction to DSL development with MPS, see Campagne (2014).

[4] https://eclipse.org/xtend

Chapter 3

Models for Fish Stock Prediction

This foundational chapter introduces established approaches to modeling the dynamics of fish stocks. It focuses on the fundamentals of deterministic differential equation models because they serve as a basis for our own stock model, which we develop in Chapter 11. For a more in-depth discussion of the topics covered in this chapter and for an overview on other common modeling approaches for fish stock dynamics, such as difference equations and stochastic models, we refer to the works of Quinn and Deriso (1999), Hilborn and Walters (1992), as well as Beverton and Holt (1957).

Based on the textbooks named above, Section 3.1 discusses the general mathematical framework that has traditionally been applied to describe fish stocks and their dynamics quantitatively. The subsequent sections show how different aspects of these dynamics can be modeled within the general framework, namely, mortality and fishing (Section 3.2), movement and migration (Section 3.3), as well as aging and reproduction (Section 3.4).

3.1 Modeling Fish Population Dynamics

Traditionally, models for fish populations describe stocks through a single time-dependent variable $N(t)$, which represents the number of individuals in that stock. Alternatively, the aggregated biomass of the stock $B(t)$ is considered, which is related to $N(t)$ via the *average fish weight* $\overline{W}$:

$$B(t) = \overline{W}N(t) \tag{3.1}$$

The dynamics of a stock are modeled by differential equations and *instantaneous* change rates (in contrast to interval change with difference

equations). For example, the *growth* of a population can be described by

$$\frac{dN(t)}{dt} = rN(t), \tag{3.2}$$

where r is the *intrinsic rate of increase*. Thus, for any given initial population size N_0 at time $t_0 = 0$, the stock size is given by

$$N(t) = N_0 \exp(rt). \tag{3.3}$$

Depending on the value of r, the stock either increases exponentially ($r > 0$), stays the same ($r = 0$), or decreases exponentially ($r < 0$).

In reality, however, stocks do not grow infinitely large because there are certain limitations for its growth (e. g., food limitation due to competition). This aspect is modeled by introducing the *carrying capacity* K of the ecosystem for the specific stock and by redefining the rate of change of $N(t)$ to

$$\frac{dN(t)}{dt} = rN(t)\left(1 - \frac{N(t)}{K}\right). \tag{3.4}$$

In this way, population growth is slowed down until it reaches zero for $N = K$ (and is negative for $N > K$). The model given by Equation 3.4 is called the *logistic law of population growth*.

3.1.1 Multiple Species

The ecological foundation of the logistic growth model is that individuals of a population compete with each other for limited resources, which restricts growth rates. However, since competition does not only occur within the same population but also between different species, we are interested in extending our model to include multiple species.

For this purpose, we consider a community of n species with population sizes $N_i(t)$, intrinsic rates of increase r_i, and carrying capacities of the ecosystem for each species K_i. Furthermore, we assume interaction parameters $\alpha_{ij} \geqslant 0$ that describe the negative effect which species j has on species i. We extend the logistic growth law to the following system of differential

equations:

$$\frac{dN_i(t)}{dt} = r_i N_i(t) \left(\frac{K_i - N_i(t) - \sum_{j \neq i}^{n} \alpha_{ij} N_j(t)}{K_i} \right) \tag{3.5}$$

Thereby, we effectively lower the carrying capacity for species that are affected negatively by competitors (for details see Quinn and Deriso 1999).

3.2 Mortality and Fishing

The mortality of individuals in a fish stock can be modeled as negative population growth via the *instantaneous total mortality* $Z \geqslant 0$ resulting in

$$\frac{dN(t)}{dt} = -ZN(t). \tag{3.6}$$

As with population growth, this leads to an exponential decay of the population over time:

$$N(t) = N_0 \exp(-Zt) \tag{3.7}$$

In the context of mortality, one is often interested in assessing how many fish are still alive after a certain amount of time has passed, which is represented by the *survival fraction*

$$S(t) = \frac{N(t)}{N_0} = \exp(-Zt). \tag{3.8}$$

The inverse measure is the *death fraction* given by

$$A(t) = 1 - S(t) = 1 - \exp(-Zt). \tag{3.9}$$

From Equation 3.8 it can be seen that the survival fraction after $t_1 > 0$ can be converted into instantaneous total mortality via the formula

$$Z = -\frac{\ln(S(t_1))}{t_1}. \tag{3.10}$$

3.2.1 Fishing

For harvested fish stocks, total mortality Z is typically composed of instantaneous *natural mortality* $M \geqslant 0$ and instantaneous *fishing mortality* $F \geqslant 0$:

$$Z = M + F \tag{3.11}$$

Therefore, the mortality in harvested stocks affects the population size in the following way:

$$\frac{dN(t)}{dt} = -MN(t) - FN(t) \tag{3.12}$$

Of course, any instantaneous rate, such as fishing mortality F, can be modeled to be dependent on time. This allows, for example, to represent intra- and inter-year variability of fishing pressure.

3.3 Space and Movement

If space is considered at all in traditional fish stock models, it is typically included only by regarding n abstract regions. For each of these areas, we let $N_i(t)$ represent the population size in that region, r_i its intrinsic rate of increase, and K_i the carrying capacity of the i-th area. We parametrize the instantaneous *rate of movement* from region i to region j as $\psi_{i\to j} \geqslant 0$. The total rate of movement out of the i-th region—called the instantaneous *exit rate*—is given by

$$\psi_i^* = \sum_{j\neq i}^{n} \psi_{i\to j}. \tag{3.13}$$

This allows us to extend the logistic law of population growth to multiple areas by considering the following system of equations:

$$\frac{dN_i(t)}{dt} = r_i N_i(t)\left(1 - \frac{N_i(t)}{K_i}\right) - \sum_{j\neq i}^{n} \psi_{i\to j} N_i(t) + \sum_{j\neq i}^{n} \psi_{j\to i} N_j(t) \tag{3.14}$$

$$= r_i N_i(t)\left(1 - \frac{N_i(t)}{K_i}\right) - \psi_i^* N_i(t) + \sum_{j\neq i}^{n} \psi_{j\to i} N_j(t) \tag{3.15}$$

The second and third term in Equation 3.15 extend the simple logistic model by adding emigration from region i and immigration into region i, respectively.

3.4 Age-Structured Models and Recruitment

The age structure of fish populations is commonly modeled by taking into account a finite set of fixed age classes with the transport of fish from one age class to the next being described for discrete time steps (see, for example, the well-known Leslie matrix population model; Leslie 1945). Since we are only interested in continuous models, we are not going to discuss these time-discrete models here. Instead, we focus on the less often employed continuous models that describe a fish stock through a density function $N(a,t)$ of the number of female fish of age a at time t. With this density function, the number of female fish that are present at time t_0 and that are larger than a_0 but smaller than a_1 is given by

$$\int_{a_0}^{a_1} N(\phi, t_0)\, d\phi. \tag{3.16}$$

The evolution of the stock with respect to aging and mortality is governed by the Partial Differential Equation (PDE)

$$\frac{\partial N(a,t)}{\partial t} + \frac{\partial N(a,t)}{\partial a} = -Z(a,t)N(a,t), \tag{3.17}$$

where $Z(a,t) \geqslant 0$ is the total mortality of females of size a at time t. The term $\frac{\partial N}{\partial a}$ ensures that the aging of fish is represented in the model: they are "transported" with unit velocity in the age dimension.

3.4.1 Recruitment

An age-structured model must not only take into account aging and mortality but also has to include the regeneration of the stock through reproduction. In the context of fish population dynamics, the renewal of the stock is referred to as *recruitment*—the process of acquiring new recruits. An

individual fish is called a *recruit* once it has grown big enough to be caught with fishing gear (and, thus, is accessible to fisheries assessment methods).

Many time-discrete stock models parametrize the recruitment process and do not explicitly model its dependence on earlier life stages (eggs, larvae, and juveniles). Well-known parametrizations of this kind include the Beverton-Holt (Beverton and Holt 1957) as well as the Ricker spawner-recruit model (Ricker 1975).

In contrast to this, continuous models are able to explicitly describe the full life cycle of fish. We incorporate reproduction into the model given by Equation 3.17 by prescribing a boundary condition on the age-time domain for $a = 0$:

$$N(0,t) = \mathcal{B}(t), \tag{3.18}$$

where $\mathcal{B}$ is the density function of all births in the population. Given the *net fecundity rate* $f(a,t)$ of females of size a at time t as well as the *minimum* and *maximum age of reproduction* α and ω, we can express $\mathcal{B}(t)$ as

$$\mathcal{B}(t) = \int_{\alpha}^{\omega} f(\phi,t)N(\phi,t)\,d\phi. \tag{3.19}$$

Chapter 4

The Finite Element Method

This chapter covers the foundations of the Finite Element Method (FEM), which we employ in this thesis to discretize the differential operators of the Partial Differential Equations (PDEs) of our fish stock model. For brevity, we have to leave some loose ends in our outline of the FEM. Specifically, we do not cover Sobolev spaces and only hint at the concept of weak derivatives. For a more comprehensive introduction of the FEM, we refer to the textbook by Brenner and Scott (2008).

To convey the key ideas of the FEM, Section 4.1 introduces a model problem and describes how to transform it into a (weak) variational formulation. Section 4.2 presents the Galerkin approach that enables us to approximate the solution of the variational formulation by solving a linear equation system. For this approximation, we need a suitable finite-dimensional function space. Section 4.3 illustrates how to construct such a space using the concept of finite elements.

4.1 Variational Formulation and Weak Derivatives

In order to introduce the FEM, we study a model boundary value problem: let $\Omega \subset \mathbb{R}^2$ be a polygon domain and f be a continuous, real-valued function on $\overline{\Omega}$ (the closure of Ω). We want to find u that fulfills

$$-\Delta u(x) = f(x) \quad \forall\, x \in \Omega \tag{4.1}$$

with the homogeneous Dirichlet boundary condition

$$u|_{\partial\Omega} \equiv 0. \tag{4.2}$$

Here, Δ is the Laplace operator given in two dimensions by

$$\Delta = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2}. \tag{4.3}$$

We relax the requirement of the point-wise identity in Equation 4.1 by multiplying our model problem with so-called *test functions* and integrating to obtain the *variational formulation*

$$-\int_\Omega v(x)\,\Delta u(x)\,dx = \int_\Omega v(x) f(x)\,dx \quad \forall\, v \in V. \tag{4.4}$$

It can be shown that if the test function space V is "large enough," Equations 4.4 and 4.1 are equivalent.

The variational formulation in Equation 4.4 requires u to be continuously differentiable twice. To relax this requirement, we partially integrate using *Green's formula*, which states that

$$\int_\Omega v(x)\,\frac{\partial u}{\partial x_i}(x)\,dx = -\int_\Omega \frac{\partial v}{\partial x_i}(x)\,u(x)\,dx + \int_{\partial\Omega} n_i(s)\,v(s)\,u(s)\,ds, \tag{4.5}$$

where n_i is the i-th component of the outward-pointing normal vector on the boundary of the domain $\partial\Omega$. By applying Green's formula, we see that for test functions v with $v|_{\partial\Omega} \equiv 0$, Equation 4.4 can be written as

$$\int_\Omega v(x) f(x)\,dx = \sum_{i=1}^{2} \int_\Omega \frac{\partial v}{\partial x_i}(x)\,\frac{\partial u}{\partial x_i}(x)\,dx \tag{4.6}$$

$$= \int_\Omega \langle \nabla v(x), \nabla u(x) \rangle\,dx \quad \forall\, v \in V. \tag{4.7}$$

For a solution u to fulfill Equations 4.6 and 4.7, it has to be continuously differentiable only once. It is desirable to relax the regularity requirements imposed on u even further by introducing so-called *weak derivatives* that are also defined for functions which cannot be evaluated point-wise. However, we cannot explore this concept here and refer the reader to the textbook by Brenner and Scott (2008) for further details.

It can be shown that the integral

$$a(v,u) := \int_\Omega \langle \nabla v(x), \nabla u(x) \rangle \, dx \tag{4.8}$$

from Equation 4.7 is a symmetrical, continuous, coercive bilinear form. With this bilinear form, we can state the *weak variational formulation* of our model problem as

$$a(v,u) = \int_\Omega v(x) f(x) \, dx \quad \forall\, v \in V. \tag{4.9}$$

Note that a solution u that satisfies Equation 4.1 also satisfies Equation 4.9, while the opposite is not necessarily true. However, it can be proven that a solution to the weak variational formulation, if it exists, is unique. From this it follows that if a solution to our original problem Equation 4.1 exists, we find it by solving the weak variational problem.

4.2 Galerkin Approach

In order to approximate the solution of the weak variational formulation, we employ the *Galerkin approach*. The key idea of this approach is to discretize the problem by solving Equation 4.9 only for a finite dimensional sub-space $V_h \subseteq V$. Thus, we search $u_h \in V_h$ that satisfies the *finite-dimensional variational formulation*

$$a(v_h,u_h) = \int_\Omega v_h(x) f(x) \, dx \quad \forall\, v_h \in V_h. \tag{4.10}$$

For any basis $(\varphi_j)_{j=1,\dots,n}$, $n = \dim(V_h)$ of V_h, this is equivalent to testing only with the basis functions:

$$a(\varphi_i,u_h) = \int_\Omega \varphi_i(x) f(x) \, dx \quad \forall\, i = 1,\dots,n \tag{4.11}$$

Furthermore, with such a basis, we can express u_h via a coefficient vector $\mathbf{u} = (u_j)_{j=1,\dots,n}$ with

$$u_h(x) = \sum_{j=1}^{n} u_j \, \varphi_j(x). \tag{4.12}$$

Therefore, Equation 4.11 is equivalent to

$$\sum_{j=1}^{n} a(\varphi_i, \varphi_j)\, u_j = \int_\Omega \varphi_i(x) f(x)\, dx \quad \forall\, i = 1, \ldots, n. \tag{4.13}$$

Equation 4.13 can be written in matrix form as

$$\mathbf{A}\mathbf{u} = \mathbf{f} \tag{4.14}$$

with

$$\mathbf{A}_{ij} = a(\varphi_i, \varphi_j) \quad \text{and} \quad \mathbf{f}_j = \int_\Omega \varphi_j(x) f(x)\, dx. \tag{4.15}$$

Thus, we can approximate the solution to our initial problem simply by solving a linear equation system. It turns out that the approximation error of the Galerkin approach is bound by the best possible approximation of u in the finite dimensional space V_h as described by Céa's Lemma.

Lemma 4.1 (Céa's Lemma)**.** *Assume a variational problem*

$$\text{Given } F \in V', \text{ find } u \in V \text{ such that } a(u, v) = F(v) \;\; \forall\, v \in V, \tag{4.16}$$

where V is a closed sub-space of a Hilbert space $(H, \langle \cdot, \cdot \rangle)$, V' is the dual space to V, and $a(\cdot, \cdot)$ is a continuous, coercive bilinear form on V. Then for the finite-dimensional variational problem

$$\text{Given a finite-dimensional sub-space } V_h \subseteq V, \text{ find } u_h \in V_h \tag{4.17}$$

$$\text{such that } a(u_h, v_h) = F(v_h) \;\; \forall\, v_h \in V_h, \tag{4.18}$$

it holds that

$$\|u - u_h\|_V \leqslant \frac{C}{\alpha} \min_{v_h \in V_h} \|u - v_h\|_V, \tag{4.19}$$

where C is the continuity constant of $a(\cdot, \cdot)$ on V and α is the coercivity constant of $a(\cdot, \cdot)$ on V.

Proof. See Brenner and Scott (2008). □

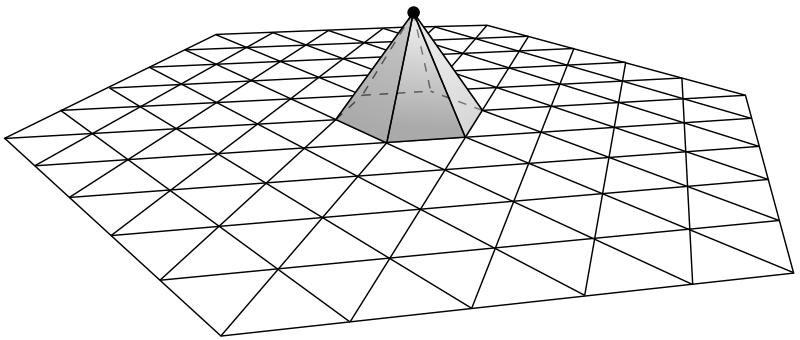


Figure 4.1. A triangulation of a two-dimensional domain with a hat function. Figure adapted from `http://brickisland.net/cs177fa12/?p=302`.

4.3 Finite Elements

In order to make use of the Galerkin approach, we need to construct a suitable finite dimensional space $V_h \subseteq V$. For this purpose, piecewise polynomial spaces are commonly employed. We describe such a piecewise polynomial space for our two-dimensional model problem via a triangulation. A triangulation $\mathcal{T}$ of Ω is a partition of the domain into triangles (or, more generally, simplices) that are either disjoint, share a common vertex, or share a common edge. With such a triangulation $\mathcal{T}$, we can define the corresponding space of two-dimensional piecewise linear polynomials as

$$V_h := \big\{ v \in L^2(\overline{\Omega}) \mid v \text{ is continuous} \wedge \tag{4.20}$$

$$v|_{\partial\Omega} = 0 \wedge \tag{4.21}$$

$$\forall\, \tau \in \mathcal{T} : v|_\tau \text{ is a linear polynomial} \big\}. \tag{4.22}$$

A mathematically convenient basis for this piecewise linear polynomial space V_h consists of the so-called *hat functions*. If we number the vertices of the triangulation $\mathcal{T}$ from 1 to n, then for each vertex $i \in \{1, \ldots, n\}$ with the spatial location $x_i \in \overline{\Omega}$, the corresponding hat function $\varphi_i \in V_h$ is (uniquely)

characterized by

$$\varphi_i(x_j) = \begin{cases} 1 & \text{if } j = i \\ 0 & \text{if } j \neq i. \end{cases} \tag{4.23}$$

An example of a triangulation and a hat function is depicted in Figure 4.1.

Note that the support of each hat function φ_i only includes the triangles that share the vertex i. Because of their small support, functions like the hat functions are called *finite elements*, which give the FEM its name.

Together, the hat functions $(\varphi_i)_{i=1,\dots,n}$ form a basis of V_h with the property that for each $v_h \in V_h$, which can be written as

$$v_h = \sum_{i=1}^{n} \lambda_i \varphi_i, \tag{4.24}$$

it holds that

$$v_h(x_i) = \lambda_i \quad \forall\, i \in \{1, \dots, n\}. \tag{4.25}$$

This means that with the basis $(\varphi_i)_i$, each function in V_h can easily be described by its values in the vertices of the triangulation $\mathcal{T}$. Since we seek to determine the λ_i that define our approximate solution $u_h \in V_h$, the λ_i are also called *Degrees of Freedom (DoF)*.

4.3.1 Finite Elements for Rectilinear Grids

For rectilinear grids, which divide the domain Ω into rectangular elements, other types of finite elements than the hat functions (which are tailored to simplices) are required. However, such finite elements can be constructed from one-dimensional hat functions via tensor products, as is described in the following.

Consider a rectangular two-dimensional domain $\Omega = [a, b] \times [c, d]$ with a rectilinear $\eta \times \nu$-grid that results from the partition points

$$a = x_0 < x_1 < \ldots < x_\eta = b \quad \text{and} \tag{4.26}$$

$$c = y_0 < y_1 < \ldots < y_\nu = d \tag{4.27}$$

for the respective dimensions. We denote the hat functions for the one-dimensional triangulation given by the $(x_i)_{i=1,\dots,\eta}$ with $(\varphi_i^{\mathrm{x}})_{i=1,\dots,\eta}$ and the

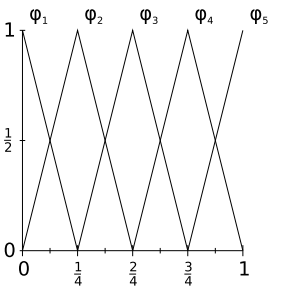


Figure 4.2. One-dimensional hat functions on the domain $[0,1]$ with partition points $\{0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\}$.

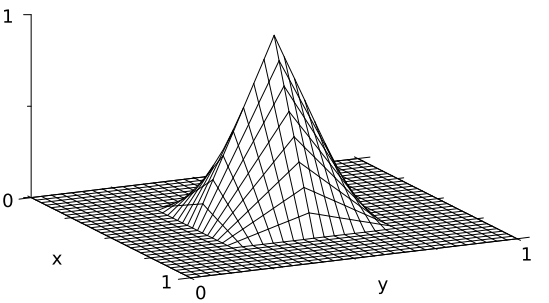


Figure 4.3. Quadrilateral finite element function with support $[1/4, 3/4]^2$ in the domain $[0,1]^2$.

hat functions for the triangulation in the y-dimension given by the $(y_i)_{i=1,\dots,\nu}$ with $(\varphi_i^{\mathrm{y}})_{i=1,\dots,\nu}$. For an example of such one-dimensional hat functions, see Figure 4.2.

For each vertex $(i,j) \in \{1,\dots,\eta\} \times \{1,\dots,\nu\}$ of the grid, we define the two-dimensional quadrilateral finite element

$$\varphi_{(i,j)}(x,y) := \varphi_i^{\mathrm{x}}(x) \cdot \varphi_j^{\mathrm{y}}(y). \tag{4.28}$$

An example of such a quadrilateral finite element function is depicted in Figure 4.3. Together, the $(\varphi_{(i,j)})_{i,j}$ form a basis of a finite-dimensional approximation space that is suitable for rectilinear grids.

Part II

The Sprat Approach

Chapter 5

Software Engineering Research Design

This chapter outlines the research design that we employ to tackle the complex challenges involved in enabling scientists to implement high-quality simulation software (cf. Chapter 1). For the research design regarding the challenge of end-to-end marine ecosystem modeling, see Chapter 9.

Section 5.1 formulates the main research goal for the software engineering part of this thesis and infers research questions to characterize this goal. In Section 5.2, we describe the research plan to accomplish the aforementioned goal and give an overview on the work packages of the plan as well as on the methods we employ to address the research questions.

5.1 Goal and Research Questions

From the stance we took on the programming challenge in computational science (Section 1.1), we can formulate our research goal for the software engineering part of this thesis as:

> Adapt existing software engineering methods to design a software engineering approach that improves the ability of scientists from different disciplines to collaboratively develop well-engineered software in an efficient and organizationally uncomplicated way without the need for extensive training of the scientists.

This goal is further explicated by the following set of Software Engineering Research Questions (SERQs).

- **SERQ1:** What is specific about software engineering in computational science?
  - **SERQ1.1:** Which software engineering methods are well-suited for being adapted for computational science?
- **SERQ2:** How can multiple DSLs be integrated for scientific software development and how can they interact with each other?
- **SERQ3:** Which DSLs are suitable for the implementation of the Sprat Marine Ecosystem Model with the Sprat Approach and how can they be integrated into a DSL hierarchy?
- **SERQ4:** Does the Sprat Approach improve the productivity of scientists in developing software and does it raise the quality of solutions implemented by them?
  - **SERQ4.1:** How does the runtime performance of solutions implemented with the Sprat PDE DSL compare to equivalent GPL solutions?
  - **SERQ4.2:** How do experts rate the functional and technical quality of the Sprat PDE DSL?
  - **SERQ4.3:** How much can program comprehension and efficiency for the implementation and maintenance of marine ecosystem simulations be increased by the Sprat Ecosystem DSL in comparison to a GPL-based solution?
  - **SERQ4.4:** How do experts rate the functional and technical quality of the Sprat Ecosystem DSL?

5.2 Research Plan

Based on our research goal and the corresponding research questions presented in Section 5.1, we structure our research in software engineering undertaken in this thesis into the following four Software Engineering Work Packages (SEWPs):

- **SEWP1:** Software Engineering in Computational Science
- **SEWP2:** Hierarchical Integration of Domain-Specific Languages
- **SEWP3:** DSLs for a Marine Ecosystem Model

- **SEWP4:** Evaluation

For each work package, we give a short description of the research conducted as part of this thesis. In particular, we highlight which research questions are answered using which methods.

5.2.1 SEWP1: Software Engineering in Computational Science

The first work package consists of identifying the specific requirements of scientists for a software engineering approach for computational science. For this purpose, in Chapter 6, we carry out a literature review to identify key characteristics of scientific software development and to explain why state-of-the-art software engineering techniques are poorly adopted in computational science. Based on this review, we are able to demonstrate that MDSE approaches are a promising starting point for adapting software engineering approaches for computational science.

Via the methods of literature review and argumentation, the work package **SEWP1** answers the research questions **SERQ1** (*What is specific about software engineering in computational science?*) with its sub-question **SERQ1.1** (*Which software engineering methods are well-suited for being adapted for computational science?*).

5.2.2 SEWP2: Hierarchical Integration of Domain-Specific Languages

Our second work package covers designing the Sprat Approach, which is an MDSE approach specifically tailored for scientists from different disciplines who collaborate to implement complex simulation software. By analyzing the software architecture of scientific simulation software (partly by studying literature, partly via argumentation), we find that such software can typically be or actually is constructed using a hierarchically layered architecture. To take advantage of the fact that the boundaries of the layers in such an architecture usually coincide with divisions between scientific disciplines, we integrate multiple DSLs into a DSL hierarchy, which forms the core concept of the Sprat Approach. We employ the method of formal specification—utilizing a combination of the Unified Modeling Language (UML)

(Object Management Group 2015c) with Object-Z (Smith 2000)—to define the notion of DSL hierarchies without any ambiguities. Furthermore, we introduce a notation for DSL hierarchies and model the process of applying Sprat with the UML.

This work package—the results of which can be found in Chapter 7—addresses the research question **SERQ2** (*How can multiple DSLs be integrated for scientific software development and how can they interact with each other?*).

5.2.3 SEWP3: DSLs for a Marine Ecosystem Model

The third work package is concerned with the process of selecting suitable DSLs for applying the Sprat Approach and the construction of a concrete DSL hierarchy. We present guidelines for choosing DSLs for scientific software development projects and list key requirements for such languages. To illustrate the DSL selection process, we describe its application to our case study example of implementing the Sprat Marine Ecosystem Model with the Sprat Approach (see **SEWP4**). For those domains for which we could not identify suitable DSLs that already exist, we designed new ones; namely, the Sprat PDE DSL for mesh-based PDE solvers and the Sprat Ecosystem DSL for ecosystem simulation specification. We specify the meta-models of both of these languages using, again, a combination of the UML and Object-Z.

The results of this work package, which can be found in Chapter 8, answer the research question **SERQ3** (*Which DSLs are suitable for the implementation of the Sprat Marine Ecosystem Model with the Sprat Approach and how can they be integrated into a DSL hierarchy?*).

5.2.4 SEWP4: Evaluation

Our fourth work package addresses the evaluation of the Sprat Approach. Koziolek (2008) distinguishes three different evaluation types, which exhibit increasing degrees of external validity:

- Type I (Feasibility): In this form of evaluation, the authors of a method apply it by themselves to an example of their choice. Thereby, this type of evaluation tests whether the method possesses certain desired properties under laboratory conditions.

- Type II (Practicability): This form of evaluation assesses the method to be evaluated when it is used by the targeted developers instead of its authors. In this way, Type II evaluations allow to test whether the method is actually suitable for the intended users.
- Type III (Cost-Benefit Analysis): A Type III evaluation consists in a full cost-benefit analysis of the method to be assessed. For example, applying the Sprat Approach potentially requires to design new DSLs, which leads to higher up front costs for a scientific software development project. We claim that these costs are more than compensated by the increased productivity of the scientific software developers and by the improved code quality achieved by employing DSLs. To test this claim, in a Type III evaluation, the same software development project is conducted at least twice with two different teams of computational scientists, one team using Sprat and the other not using it. Comparing the respective costs of both projects over their whole life cycle makes it possible to quantify the business value of the method to be evaluated. Since such studies are costly and difficult to conduct (e. g., one has to accurately control confounding variables such as developer expertise etc.), they are rarely put into practice.

To evaluate the Sprat Approach, we employ a mixed-method exploratory case study (Runeson et al. 2012) that combines Type I and Type II evaluation elements (for the reasons given above, a Type III evaluation is beyond the scope of this Ph.D. thesis). Our case study focuses on the engineering of the Sprat Marine Ecosystem Model and the DSLs designed in this process (see Chapter 9). The fact that, in the context of this thesis, we were able to use the Sprat Approach to implement a model as complex as the Sprat Model is, in itself, a demonstration of the feasibility of the Sprat Approach—i. e., a positive Type I evaluation of the approach as a whole. Throughout this thesis, however, we focus only on the evaluation of the DSLs we designed for implementing the Sprat Model, namely the Sprat PDE DSL and the Sprat Ecosystem DSL. By evaluating these languages, we also assess the Sprat Approach itself because the increase in productivity and code quality promised by the approach can mainly be attributed to the quality characteristics of the hierarchically integrated DSLs.

To evaluate the Sprat PDE DSL, we use micro- and macro-benchmarks for performance evaluation (Type I) as well as expert interviews with both

domain experts and professional DSL developers (Type II evaluation). For the assessment of the Sprat Ecosystem DSL, we conduct an online survey among domain experts (Type II evaluation) that contains embedded controlled experiments and feedback questionnaires. Furthermore, we qualitatively compare the Sprat Approach and the DSLs used to engineer the Sprat Model to related research via argumentation.

The results of our mixed-method approach, which can be found in Chapters 13, 14, and 17, answer the research question **SERQ4** (*Does the Sprat Approach improve the productivity of scientists in developing software and does it raise the quality of solutions implemented by them?*) with its sub-questions **SERQ4.1** to **4.4**. Note that one outcome of the research conducted in the context of work package **SEWP1** is that the term *quality of solutions* in **SERQ4** can be explicated in the following way: a solution is of high quality if it reconciles the conflicting quality requirements of performance, portability, and maintainability and if it produces scientifically reliable results. For further details on how the evaluation of the individual DSLs confirms the effectiveness and efficiency of the Sprat Approach as a whole, refer to Section 18.1.4 of Chapter 18.

Chapter 6

Software Engineering in Computational Science

In this chapter, we present a literature review on software engineering in computational science for which we surveyed about 50 publications. Note that, as explained in Chapter 1, the term *computational science* is used synonymously with *scientific computing* to refer to the field that applies computation to answer scientific questions. These two terms are, of course, not to be confused with *computer science*, from which *software engineering*—the application of engineering to software—is a sub-discipline. Note that software engineering may also refer to the *process* of engineering specific software systems—in this sense the term is used in the title of this chapter.

To provide a basis for our literature review, we outline the historical development of the relationship between the disciplines software engineering and computational science. This relationship is to a large extent characterized by an isolation between the two disciplines which resulted in a productivity and credibility crisis of computational science (Section 6.1). Based on this perspective, we review publications on case studies and surveys conducted among computational scientists to identify several key characteristics that are unique to scientific software development and that explain why state-of-the-art software engineering techniques are poorly adopted in computational science (Section 6.2). We conclude that software engineering methods should be adapted to the specific requirements of scientific software development in order to be accepted by the corresponding community. Furthermore, we demonstrate that Model-Driven Software Engineering (MDSE) approaches are a promising starting point for such an adaptation process (Section 6.3). The insights gained in this chapter lay the foundations for the design of the Sprat Approach as presented in Chapter 7.

6.1 Software Engineering and Computational Science

When software engineers started to examine the software development practice in computational science, they noticed a "wide chasm" (Hannay et al. 2009) between how these two disciplines view software development. Faulk et al. (2009) describe this chasm between the two subjects using an allegory which depicts computational science as an isolated island that has been colonized but then was left abandoned for decades:

> "Returning visitors (software engineers) find the inhabitants (scientific programmers) apparently speaking the same language, but communication—and thus collaboration—is nearly impossible; the technologies, culture, and language semantics themselves have evolved and adapted to circumstances unknown to the original colonizers."

The fact that these two cultures are "separated by a common language" created a communication gap that inhibits knowledge transfer between them. As a result, modern software engineering practices are rarely employed in computational science.

6.1.1 The Origins of the Chasm

The origins of the rift between computational science and software engineering can be traced back as far as the dawn of modern computing in the 1940s. At that time, "scientific computing" was a pleonasm: the electronic digital computer was invented solely to solve complex mathematical problems for the advancement of science and engineering (Ceruzzi 2003). As the discipline of computer science emerged in the late 1950s and early 60s, it struggled to distinguish itself from electrical engineering and applied mathematics—the disciplines traditionally engaged in the "study of computers" (Newell et al. 1967). To differentiate itself from these applied disciplines, computer science invented the "stigma of all things 'applied'" and aimed for generality in all its methods and techniques (Vessey 1997). This approach was supposed to ensure that "the core of computer science [...] will remain a field of its own, ahead of, and separate from the application domain specialists" (George E. Forsyth, President of ACM in 1965, as quoted by

Vessey 1997). This first estrangement of computer science from the field that would later on become computational science was adopted and even reinforced by software engineering.

The term software engineering—and at the same time the corresponding sub-discipline of computer science—was institutionalized by a conference with the aforementioned title organized by NATO in 1968 (Naur and Randell 1969). That NATO was the sponsor of this conference marks the relative distance of software engineering from computation in the academic context. The perception was that while errors in scientific data processing applications may be a "hassle," they are all in all tolerable. In contrast, failures in mission-critical military systems may cost lives and substantial amounts of money (Ceruzzi 2003, Chap. 3).

Based on this attitude, software engineering—like computer science as a whole—aimed for generality in its methods, techniques, and processes and focused almost exclusively on business and embedded software (Kelly 2007). Because of this ideal of generality, the question of how specifically computational scientists should develop their software in a well-engineered way, would probably have perplexed a software engineer and the answer might have been: "Well, just like any other application software."

For some time, the coexistence of computational science and software engineering in relative ignorance of one another continued without greater interruptions. One noteworthy exception from this is a paper from Hatton and Roberts (1994) in which they examine the accuracy of over 15 large commercial software libraries of numerical algorithms. Their findings disagree with the typical assumption of computational scientists that their software is accurate to the precision of the machine arithmetic. Hatton and Roberts discovered that in their sample the numerical discrepancy between expected and computed results increases about 1 % per 4000 Lines of Code (LOC). These results, however, did at first not find a larger echo in both communities.

6.1.2 The Productivity Crisis of Computational Science

In general, computational scientists did not see a reason to be concerned about the quality of their software. This attitude started to change roughly ten years ago, when more and more deficiencies regarding the productivity of scientific software development became apparent. These deficiencies,

which led some to speak of a "productivity gridlock" (Faulk et al. 2009) in computational science, were revealed mostly by two parallel developments:

1. The encounter of the clock speed limit for single-core processors and, with it, the introduction of multi-core and heterogeneous computing systems.
2. The integration of more and more effects into scientific simulations that govern the behavior of the system under study.

Scientific software has a quite long life span (from years to decades) as it often encapsulates the accumulated knowledge and effort of scientists or research groups. Thus, it typically outlives the computing hardware for which it was originally designed. For more than two decades this was not a problem as it could be taken for granted that each new generation of microprocessors would considerably increase the performance of the scientists' software without greater modifications of the source code. With the encounter of the clock speed limit for single-core processors around the year 2004, this development came to a halt (Fuller and Millett 2011).

In order to achieve more processing power, chip designers started to scale the number of processor compute cores. At first, this was only realized for the Central Processing Unit (CPU) of the computer (multi-core era). The most recent step was to assist such multi-core CPUs by providing external accelerator devices with even greater numbers (on the order of magnitudes!) of compute cores that are inspired by the design of modern graphics processors (heterogeneous systems era). To harness the power that multi-core and especially heterogeneous systems provide for more and more detailed simulations, computational scientists are now faced with the challenge of adapting their software to exploit parallelism on ever-finer levels of granularity. A problem revealed in the course of this process is a lack of knowledge about software engineering among the scientists, which resulted in a poor maintainability of their "codes." This lack of maintainability impedes the scientists' ability to successfully scale their simulations through adaptation to new hardware architectures (Buttari et al. 2007; Dongarra et al. 2007).

The second development that prevents computational scientists from ignoring modern software engineering techniques if they want to stay—or become again—productive is related to the complexity of their models. For computation to fulfill its role as the "third pillar of scientific inquiry"

(Benioff 2005), the scientists have to increase the predictive capabilities of their models by integrating more and more scientific effects into them. This results in the need for coupling or even integrating contributions from a multidisciplinary team of scientists into a single simulation application. As earlier scientific software was developed by small teams of scientists primarily for their own research, modularity, maintainability, and team coordination could often be neglected without a large impact. The shift towards larger, interdisciplinary teams makes these often ignored aspects of software development important for the scientists and, again, exposes a knowledge gap among them that can only be overcome by engaging in a dialog with software engineering (Post 2013).

6.1.3 The Credibility Crisis of Computational Science

The challenges regarding the quality of scientific software do not only lead to a decreased development performance but also interfere with the credibility of its results. This aspect becomes especially important as the societal impact of computer simulations has grown in recent times, which can be exemplified by the so-called "Climategate" scandal. The scandal erupted after hackers leaked the e-mail correspondence of scientists from the Climatic Research Unit at the University of East Anglia not long before the 2009 United Nations Climate Change Conference. While the accusations that data was forged for this conference turned out to be unfounded, the e-mails uncovered lacking programming skills among the researchers and exposed to a large public audience the widely applied practice in climate science to *not* release simulation code and data together with corresponding publications (Merali 2010). This in itself was, of course, enough to undermine the work of the scientists, as the predictive capabilities of simulations are only as good as their code quality (Hatton and Roberts 1994) and their code was not even available for peer review—not to mention public review.

Within the scientific community the "Climategate" scandal initiated a debate about the *reproducibility* of computational results that also attracted some attention of the software engineering community (for a discussion of this debate, see Section 6.3).

6.1.4 Bridging the Gap

Both the productivity and the credibility crisis, make it abundantly clear that the isolated coexistence of computational science and software engineering cannot continue. Problems with programming and the design of scientific software should not be dismissed as "just a hassle" anymore in order for computational science to advance and to keep its promise of fulfilling the role of a third paradigm for scientific discovery. But even though the events of the past decade have initiated a dialog between software engineering and computational science, progress is still slow. For example, Brown et al. (2015) in a recent publication give a quite sarcastic description of the current status of scientific software and the associated development practices: imagine scientific software as a web browser with no URL entry box—you enter the web address into a configuration file. The browser can use either http or https but not both at the same time, which is controlled by an `#ifdef` switch requiring you to recompile the browser with the second to last version of a Fortran77 compiler by a specific vendor. In principle, you could change all that as the software is open source but, unfortunately, development is private and you would have to apply to be granted access to the source code, which you will only gain if your intentions are in agreement with those of the main developers.

While such design choices seem absurd to us for modern-day applications, Brown et al. conclude that they "represent the status quo in many scientific software packages" and are often "vehemently defended." However, the mistrust of computational science towards modern software engineering techniques is not totally ungrounded: as software engineering aimed for generality in all its methods and processes, it ignored the unique demands of computational science (Kelly 2007). Therefore, scientists far too often experienced the methodological offerings of software engineering as being full of "accidental complexities" instead of being helpful (Wilson 2006b). Accordingly, distrust and prejudices are still regularly found on both sides of the "software chasm" that so far has not been closed up again.

In order to understand which approaches might be suitable to bridge the gap between the two disciplines, we have to closely examine the characteristics of scientific software development and must take the distinctive requirements of computational science seriously. Only if we—from the perspective of software engineering—abandon the "stigma of all things

'applied'" (Vessey 1997) and end the unconditional striving for generality that does not do computational science any justice, can we hope to improve the current situation.

6.2 Characteristics of Scientific Software Development

In this section, we survey literature from the software engineering community that examines the characteristics of scientific software development. As described in the introduction in Chapter 1, this topic emerged only after 2005 and was investigated mainly by conducting case studies and a few surveys. Reviewing and integrating the observations of these different studies allows us to reduce the major risk that is commonly associated with case studies: their lack of generalizability. Combining and contrasting the findings of multiple studies in different environments makes it possible to identify a set of characteristics that is likely to be inherent to scientific software development in general. Although the majority of the literature on software development practices in computational science dates back to the years 2006 to 2009, the observations made by them appear to still hold true today, as is indicated by related newer publications (e. g., Brown et al. 2015; Carver and Epperly 2014; Joppa et al. 2013).

The papers included into our literature review were identified by querying databases like the IEEE Computer Society Digital Library,[1] the ACM Digital Library,[2] and Google Scholar.[3] Additionally, we searched the articles of journals we expected to be of specific interest, such as *Computing in Science & Engineering* as well as the proceedings of conferences and workshops like the *International Conference on Software Engineering* and the *International Workshop on Software Engineering for Computational Science and Engineering* from 2005 on. Some papers were suggested by peers or identified by references from other articles. A limitation of this strategy is that we necessarily have to rely on a limited number of keywords in our database queries. We tried to mitigate this risk by varying the search phrases and, e. g., using

[1] http://www.computer.org/csdl

[2] http://dl.acm.org

[3] https://scholar.google.com

different synonyms (such as scientific computing for computational science etc.).

Since the variety of scientific software and its applications is large (Segal and Morris 2008), computational scientists do not form a homogeneous group. Scientists develop software ranging from scripts for small-scale data analysis to complex coupled multi-physics simulations executed on high-end hardware. In our literature survey, we focus mostly—though not exclusively—on the latter group which forms the High Performance Computing (HPC) community. The reason for directing our attention to this group is that it is most affected by the productivity and credibility crisis portrayed in the previous section.

As a result of our literature review, we identified 13 recurring key characteristics of scientific software development that software engineering has to take into consideration when adapting techniques for this community. These characteristics can be divided into three groups:

1. Characteristics resulting from the *nature of scientific challenges*:
   - 1.a) Requirements are not known up front
   - 1.b) Verification and validation is difficult and strictly scientific
   - 1.c) Overly formal software processes restrict research
2. Characteristics resulting from the *limitations of computers*:
   - 2.a) Development is driven and limited by hardware
   - 2.b) Use of "old" programming languages and technologies
   - 2.c) Intermingling of domain logic and implementation details
   - 2.d) Conflicting software quality requirements (performance, portability, and maintainability)
3. Characteristics resulting from the *cultural environment* of scientific software development:
   - 3.a) Few scientists are trained in software engineering
   - 3.b) Different terminology
   - 3.c) Scientific software in itself has no value but still it is long-lived
   - 3.d) Creating a shared understanding of a "code" is difficult
   - 3.e) Little code re-use
   - 3.f) Disregard of most modern software engineering methods

In the following subsections, we detail our findings with regard to these three groups of key characteristics and describe their impact on the adaptation of software engineering methods for computational science.

6.2.1 Characteristics Resulting From the Nature of Scientific Challenges

All characteristics of software development in computational science that are listed in this section result from the fact that scientific software is an integral part of a discovery process. When you develop software to explore previously unknown phenomena, it is hard to specify exactly up front what the software is required to do, how its output is supposed to look like, and how to proceed during its development.

a) Requirements Are not Known Up Front
In science, software is used to make novel discoveries and to further our understanding of the world. Since scientific software is deeply embedded into an exploratory process, you never know where its development might take you. Thus, it is hard to specify the requirements for this kind of software up front as demanded by traditional software processes. Accordingly, most of the requirements—except for the most obvious high-level ones—are discovered only during the course of development in a highly iterative process (Segal and Morris 2008). The reason for this is that while the underlying scientific theory is well-established in most scientific software projects, it is unclear in advance how this theory can be applied to the specific problem at hand (Carver et al. 2007). When the sole purpose of the project is to further domain understanding, the exact outcome of the project is—by definition—unknown.

The primary intention of software development in computational science is *not* to produce software but to obtain scientific results. For this reason, it is unsurprising that scientific programmers say about themselves that they are "programming experimentally" (Segal 2005). The scientific models as well as their implementations are treated as evolving theories to test specific hypotheses (Easterbrook and Johns 2009). Thus, it is the insights gained from one version of the software that determine what is needed for the next version in relatively short iterations (Hochstein et al. 2005). This iterative

nature of the scientific software development process does, therefore, not indicate a lack of programming skills among the scientists but mirrors the growing understanding of the requirements as the software evolves (Segal 2007).

That scientists rarely see design and requirements analysis as distinct steps in software development (Sanders and Kelly 2008), is in part due to the fact that many scientific applications start out as very small projects and begin to grow only on the basis of their scientific success (Basili et al. 2008). Thus, the requirements for the first version of the software often stem from a single scientist's experience and are usually not explicated by that person. If the software proves to be useful to a broader community, its members tend to make suggestions on features to incorporate into the software and, thereby, they add requirements. These requirements, however, are not explicated in a way that would be detailed enough to form the basis of a contractual document as it is required in established software engineering processes (Segal 2008). In the case that a sponsor organization demands the documentation of the design and requirements analysis process, the scientists typically do not write these documents before the software is almost complete (Sanders and Kelly 2008).

b) Verification and Validation Is Difficult and Strictly Scientific
In the context of scientific software, *verification* means to demonstrate that the implementation of algorithms and the equations embodied within them are correct. Thus, verification is purely concerned with theoretical constructs. In contrast, *validation* means to demonstrate that the software and the mathematical model represented by it succeed in capturing all relevant scientific effects correctly. Hence, validation has to ensure that the software output is in sufficient agreement with observations from the real world (Carver et al. 2007). Verification and validation pose serious challenges in all areas of software development but are especially difficult in computational science due to a lack of test oracles, because of complex distributed hardware environments with inadequate tool support, and due to the scientists' undervaluation of software in general (Kanewala and Bieman 2014).

Validation is particularly challenging as the scientists frequently lack observational data to compare their model results to—after all, they use simulations precisely because the subject at hand is "too complex, too large,

too small, too dangerous, or too expensive to explore in the real world" (Segal and Morris 2008). But even if observations are available, they can still be incomplete or incorrect and they never extend to the future, with which many simulations are concerned (Sanders and Kelly 2008). Lastly, if deviations from observations occur, it is hard to trace down their causes which can lie in three distinct dimensions or even a combination of them (Carver et al. 2007):

1. The mathematical model of reality can be insufficient, meaning that scientific aspects are wrong.
2. The algorithm used to discretize the mathematical problem can be inadequate (e. g., have stability problems).
3. The implementation of the algorithm can be wrong due to programming errors.

Therefore, extensive checks of the code and the scientific model have to take place during the development, which highlights the importance of proper verification (Shull et al. 2005).

For the purpose of verification, computational scientists can rely on established testing methods (e. g., unit tests and assertions). In addition to these traditional approaches, they employ checks to test whether theoretically guaranteed results hold true (propositions regarding approximation stability and quality, conservation of certain physical quantities, etc.). However, especially system testing is complicated by the fact that simulation software often runs on distributed hardware that is poorly supported by tools for debugging and profiling (Basili et al. 2008).

Because of the difficulties associated with testing and because of a general disregard for code quality (Section 6.2.3 c)), formal verification procedures are not common in computational science (Segal 2007). Prabhu et al. (2011) report that according to their survey, scientists spend more than half of their programming time on finding and fixing errors but only employ "primitive" debugging and testing methods. The testing which is performed is only of cursory nature and consists in manually checking for the answer to questions like "does the software do what I expect it to do with inputs of the type I would expect to use?" (Segal 2008). In this context, visualization of output data is the most common tool for verification and

validation purposes. However, visualization can provide no more than a "sanity check" indicating that the code is behaving "reasonably" (Carver et al. 2006).

A reason for this lack of disciplined testing can be seen in the scientists' regarding their software as imperfect evolving theories that allow them to test hypotheses. From this point of view, they judge model and algorithmic defects to be of far greater significance than coding defects (Basili et al. 2008; Easterbrook and Johns 2009). This also explains why almost all testing strategies employed by the scientists are strictly scientific (Faulk et al. 2009). Since they do not perceive the source code to be an entity in its own right and view it as a more or less direct representation of the underlying scientific theory (Section 6.2.3 c)), they only look at the output of the software and check whether it is in agreement with their current theory. The scientists treat the software like any other (physical) experimentation apparatus that is usually expected to function well. This assumption is only questioned if the data is in conflict with what the scientists would roughly expect (Segal 2008). For this reason, a software engineering approach for computational science should draw the programmers attention to the important role of the correctness of the source code (Hinsen 2015). This can be accomplished, for example, by providing easy-to-use methods to test assertions that are meaningful to the scientists on a scientific level.

c) Overly Formal Software Processes Restrict Research
Traditional software development processes that employ a "big design up front" approach—like the waterfall model (Royce 1970)—are "a poor fit" for computational science (Easterbrook and Johns 2009). The reason for this is that software development in science is deeply embedded into the scientific method, which makes the up front specification of requirements impossible (Section 6.2.1 a)) and introduces challenges with the verification and validation of the implementation (Section 6.2.1 b)). As scientific software is evolving continuously, no clear-cut requirements analysis, design, or maintenance phases could be discerned (Segal 2007) and the developers need the flexibility to quickly experiment with different solution approaches (Carver et al. 2007).

Instead of established software engineering processes, scientists apply an informal, non-standard process that is depicted in Figure 6.1. Their method is highly iterative and starts from a vague idea of which scientific problem

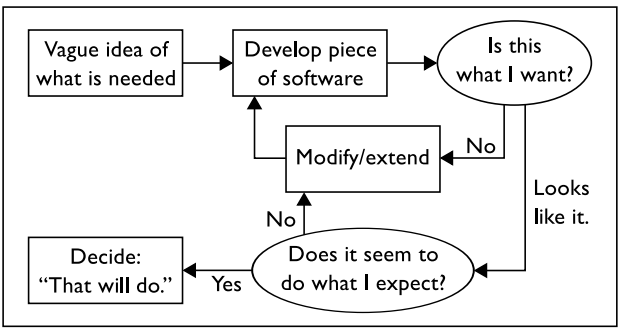


Figure 6.1. A model of scientific software development adapted from Segal and Morris (2008).

the software is supposed to solve and what the application, therefore, could be required to do. Based on this idea, a prototype is developed and is continuously improved guided by the questions "does it do what I want?" and "does it help solve the scientific problem at hand?" (Segal 2008). When the software reaches a state of maturity which enables it to answer the research question under study, it is subjected to cursory testing as described in Section 6.2.1 b). If the output of the software does not meet the expectations of the developers, modifications become necessary until "plausible" output is achieved. Note that these modifications almost always involve *both* the code and the underlying scientific theory (Sanders and Kelly 2008). Therefore, and because the code is often perceived as a mere representation of the theory and not as an entity in its own right (Section 6.2.3 c)), the development method of the scientists could, in a certain sense, be considered primarily a *theory* development method rather than a software development method. The scientists regard their informal software process as necessarily following from applying the scientific method to scientific reasoning with the help of computing (Kendall et al. 2008).

Several researchers point out that the development approach prevalent in computational science bears some similarity to "agile" software engineering methods,[4] such as Extreme Programming (Beck 2000). Many computational

[4] http://agilemanifesto.org

scientists have been operating with an "agile philosophy" long before the term was even introduced in software engineering (Carver et al. 2007). However, all established development processes—even agile ones—are generally rejected by the community as too formal because the scientists feel that these processes constrain them in experimenting with their software (Segal 2008). Therefore, any development approach to be adopted by the computational science community must be very lightweight and integrate well with the software/theory method depicted in Figure 6.1.

6.2.2 Characteristics Resulting From the Limitations of Computers

In this section, we discuss characteristics of software development in computational science that are due to limitations regarding available computing resources and their efficient programming.

a) Development Is Driven and Limited by Hardware
Complex simulation software is never perceived as "finished" by the computational scientists. Since it always can only be an imperfect representation of the highly complex reality, one could constantly hope to improve the software and its output by modeling more of the relevant scientific processes or increasing the resolution of discretizations. Therefore, scientific software is typically not limited by theory but by the available computing resources and their efficient utilization (Easterbrook and Johns 2009).

The development of scientific software is not only limited but also driven by the available compute hardware in two ways. First, every time new hardware that increases computational power by an order of magnitude becomes available, completely new types of coupled multi-physics simulations suddenly become possible. This necessitates the implementation of new simulation software or, at least, the coupling of simulations in a more complex way. Second, new hardware platforms regularly introduce changes in the underlying hardware architecture. Harnessing the power of these new architectures typically requires to adapt existing simulation software for performance optimization (Faulk et al. 2009).

b) Use of "Old" Programming Languages and Technologies
Especially HPC applications tend to be written in "older," lower-level pro-

gramming languages like Fortran or C and use long-established technologies like Message Passing Interface (MPI) (Message Passing Interface Forum 2012). This is due to several reasons, one being the long lifetime of HPC software (see Section 6.2.3 c)). In this context, Fortran and C are "safe choices" because it is likely that for many years to come every hardware platform is going to support these languages (Faulk et al. 2009). Scientific programmers are skeptical about new technologies because the history of HPC is full of tools and programming languages that promised productivity increases but were discontinued after a while. Additionally, the low abstraction level of languages such as Fortran or C implies that the developers are operating closer to the underlying hardware platform. Therefore, these languages provide predictable performance and allow for more hand-crafted performance optimizations (Basili et al. 2008).

The scientists do not see any reason to adopt newer programming languages as the established ones are easy to learn (which is important for self-teaching; see Section 6.2.3 a)) and there is a huge amount of legacy code written in those languages (Carver et al. 2007). Their decision is also highly influenced by cultural traditions and believes: interviewees of Sanders and Kelly (2008) reported that object orientation did not "buy [them] anything" and that "a couple lines of C would take a large amount of C++ code." To be accepted by the computational science community, a new programming language would have to be easy to learn, offer reasonably high performance, exhibit stability, and transform language constructs into machine instructions in a predictable way (Carver et al. 2007).

The HPC community uses higher-level languages such as Matlab almost exclusively for prototyping algorithms, which are later re-implemented for higher performance using lower-level languages (Kendall et al. 2008). In disciplines that are less technology-affine—such as biology or psychology—newer languages such a Matlab and Python are more widely adopted for small-scale projects (Prabhu et al. 2011). For larger projects, new technologies have better chances of being accepted if they can coexist with older ones and do not immediately require a full buy-in. This explains why frameworks that dictate the user how to structure their program are seldom used. The scientists prefer re-implementing a lot of existing functionality to giving up control over the code that they want to experiment with (Basili et al. 2008).

When adapting software engineering methods for computational science, one has to take into consideration the reluctance especially of HPC developers regarding any technology that is not tested by time and that runs the risk of ceasing to be supported. Therefore, it is important to make all software aimed at scientific programmers available under open source licenses and *not* to force them to use newer programming languages. This allows the scientists to, at least in principle, keep maintaining discontinued software by themselves. Also a stepwise buy-in into proposed technologies should be made possible.

c) Intermingling of Domain Logic and Implementation Details
The use of older procedural programming languages in computational science (Section 6.2.2 b)) and a focus on performance (Section 6.2.2 d)) often impede the separation of domain logic and implementation details in the solution artifacts. This makes it difficult to evolve scientific theory and implementation-specific aspects (such as optimizations for a particular hardware platform) independently of one another and ultimately leads to software that is hard to maintain. It also results in an expertise problem: if all aspects of the implementation are intermingled, the developer should be—but rarely is—equally proficient in all those aspects ranging from the domain knowledge to numerical methods to the specifics of certain processor designs (Faulk et al. 2009). Software engineering approaches, thus, should focus on separating these concerns without negatively affecting performance levels.

d) Conflicting Software Quality Requirements
The ISO/IEC 25010 standard lists eight categories of product quality characteristics that software can be evaluated for: functional suitability, reliability, performance efficiency, usability, security, compatibility, maintainability and portability (ISO 25010 2011). In their field studies, Carver et al. (2007) find that scientific software developers rank the following characteristics as the most important ones in descending order:

1. Functional correctness
2. Performance
3. Portability
4. Maintainability

It seems clear that scientists perceive the correctness of the results of their software as the topmost priority. After all, the results are supposed to accurately represent processes in the real world and are used as a starting point for scientific reasoning.

Especially in the HPC context, it is also not surprising that the scientists value performance as large simulations can take days or even months to run. But however valuable performance is to the scientists, it is not an end in itself—the real goal is to do science. Therefore, the most adequate performance metric for scientific software is not given in Floating Point Operations Per Second (FLOPS) but rather in "scientifically useful results per calendar time" (Basili et al. 2008; Carver et al. 2006). Furthermore, performance is in conflict with portability and maintainability because it is usually achieved by introducing hardware-specific optimizations that reduce the readability of the code. The additional quality attributes, portability and maintainability, are also of great importance to the scientists as scientific software is long-lived (Section 6.2.3 c)). During its long lifetime, hardware platforms change frequently, which limits the possibility for hardware-specific performance tuning (Kendall et al. 2008).

The conflict between performance and portability is experienced as problematic by the scientists. However, software engineering can, so far, offer little guidance in this aspect because performance and portability are among the least significant quality characteristics for most software engineering approaches (Faulk et al. 2009). Therefore, adaptations of software engineering techniques for computational science must pay special attention to alleviating the performance/portability issue.

6.2.3 Characteristics Resulting From the Cultural Environment of Scientific Software Development

The characteristics that are listed in this section result from the cultural environment in which scientific software development takes place. This environment is shaped, for example, by the training of computational scientists and the funding schemes of scientific research projects.

a) Few Scientists Are Trained in Software Engineering
Segal (2007) describes computational scientists as "professional end user

developers" who work in very technical and "knowledge-rich" domains and typically develop software solely to advance their own professional goals. What they have in common with conventional end user developers is that most of them lack any kind of formal computer science training and do not perceive themselves as software engineers but as domain experts even though they spend a considerable amount of their research time on developing software.[5] In contrast to most conventional end user developers, however, computational scientists rarely experience any difficulties learning general-purpose programming languages.

The self-perception of scientific software developers as scientists rather than developers is grounded in the cultural values of the community: because the ultimate goal is to further scientific knowledge, domain expertise is seen as "intellectual capital," whereas software development skills are just "techniques"—a means to an end. This also implies that possessing software engineering skills is not valued when it comes to recruitment and promotion decisions. Jobs are awarded to those candidates who are qualified best for what is usually viewed as the highest priority in computational science: scientific theory (Sanders and Kelly 2008).

In addition to not being appreciated, learning software engineering skills is perceived as an excessive demand by computational scientists as they already have enough to do with performing as a scientists (write papers and grants, give presentations, etc.) and keeping up with their fast-developing fields of study (Killcoyne and Boyle 2009). This problem is reinforced by the fact that computational science is already becoming more and more interdisciplinary, and thus more complicated, purely from the scientific side. As more and more effects are to be considered by ever more complex simulations, computational scientists already have to be able to collaborate with researchers from other disciplines and "speak their language" (Carver et al. 2007). All of this leaves little room for software engineering education. The knowledge of programming languages that scientists possess—which is obviously not identical with software engineering knowledge—is usually acquired by self-study or from co-workers (Basili et al. 2008; Carver et al. 2013).

[5] Prabhu et al. (2011) in their study of 114 research scientists from diverse fields find that more than a third of their subjects' research time is spent on software development tasks.

Even though software development is largely perceived as a burden, computational scientists do not like delegating it to others. They feel they possess the necessary technical skills and find it easier to do it themselves than to explain their needs to somebody else (Easterbrook and Johns 2009). Furthermore, the development process critically depends on domain knowledge (Segal 2009; Segal and Morris 2008). It is perceived as easier to teach scientists how to program than to make software engineers understand the domain science because many of the applications "require a Ph.D. in physics or a branch of engineering just to understand the problem" (Carver et al. 2007). This view is backed up by a study from Segal (2005) in which software engineers implemented a scientific software library based on requirement and specification documents written by scientists. Even though formal minuted meetings were held during the development process to establish a shared understanding between the scientists and the software engineers, the final product did not meet the requirements of the scientists.

Although it seems neither desirable nor feasible to delegate the work of computational scientists to external software engineers, it is regarded beneficial to have a few software engineers working in scientific research institutions to provide development support (Killcoyne and Boyle 2009). However, such positions have typically not been supported by funding agencies in the past (Carver et al. 2007).

b) Different Terminology

Due to the isolated development of computational science and software engineering, both fields have established distinct terminologies even for shared concepts (Faulk et al. 2009). The terms and metaphors of the computational scientists are typically drawn either from the scientific method itself or from rather low-level concepts of computation. For example, scientific programmers do not call their applications "software" but rather speak of "codes." A "serial code" is a piece of software that does not utilize parallelism and "scaling" such a code means adapting it for parallel execution etc.

Because of their distinct terminology, scientific programmers sometimes (have to) re-invent existing software engineering techniques: they just do not find the existing methods that would fit their needs because they look for them using the "wrong" vocabulary. For them, these techniques are just "natural" aspects of a research method rather than being a general

tool for software development. Therefore, scientists in some cases do not recognize that they are already using software engineering methods if they are confronted with them in the vocabulary of software engineering (Easterbrook and Johns 2009).

It appears that software engineers have to adapt their vocabulary in order to be understood and taken seriously in the domain of computational science. The terminology of software engineering is often regarded by computational scientists as consisting mostly of "glitzy" marketing terms that are nothing but empty promises (Killcoyne and Boyle 2009).

c) Scientific Software in Itself Has no Value but Still it Is Long-Lived
For computational scientists, the software they produce has no value in itself; its value is solely based on its ability to efficiently solve problems at hand and make new scientific discoveries (Faulk et al. 2009). This focus on novelty and discovery leads to the perception that software skills are just a "necessary craft," just a means to an end, and that acquiring them is not "real work" (Killcoyne and Boyle 2009). While domain knowledge is considered "intellectual capital," software development knowledge is merely a "technique," which consequently renders all technical decisions comparably unimportant (Segal 2007).

Additionally, many computational scientists do not regard software as an entity in its own right. In their mind, source code is a more or less direct representation of the underlying scientific theory (Sanders and Kelly 2008). Thus, the only value even a code that has been developed and maintained for decades has, does not stem from the engineering effort put into it but from the scientific knowledge accumulated in it.

Such a perspective on software leads to a situation in which code quality is not considered important either—even though it is strongly related to the quality of the scientific results (Hatton and Roberts 1994). Instead of defect rates, the only code metric that is applied to scientific software is that of novel, publishable results per LOC (Easterbrook and Johns 2009). There are even cases in which non-trivial software is implemented for the mere purpose of getting a single article published. Because the time-to-solution has to be low in such cases, not much thought is spent on quality attributes like maintainability, extensibility, or reusability. If such a rather poorly engineered code happens to keep being extended—which is how many

large codes emerge—, it is hard to remedy these deficiencies (Killcoyne and Boyle 2009).

Even though the scientists see no value in scientific software in itself, many codes have a long lifetime on the order of decades. The software may not be valuable as such but the accumulated knowledge of the researchers that is embodied in it makes it a long-time investment (Faulk et al. 2009).

During such a long life cycle, the software continuously needs to be developed further in order to reflect the advances in scientific theory and computational hardware (Carver et al. 2007; Easterbrook and Johns 2009). Because of the potentially very long lifetimes, many scientific software developers try to avoid dependencies on technologies that could become unavailable. For this reason, the number of dependencies, such as software libraries, is kept to a minimum and only such tools and programming languages are used that have already withstood the ravages of time. This is especially true in the HPC community as their codes are those most likely to be long-lived.

Despite the long life span of scientific software, the effort devoted to its maintenance is low because of a focus on the implementation of new features. Carrying out maintenance tasks is discouraged, firstly, by simply not being rewarded as it does not lead to new publishable results and, secondly, by putting the burden on the developers to demonstrate that their changes do not affect the accuracy of the simulation results (Easterbrook and Johns 2009). Additionally, the grant-based funding schemes in many branches of science make it hard to assume a long-term perspective on "caring" for scientific software, which is why "quick and dirty" solutions are selectively favored (Howison and Herbsleb 2011; Killcoyne and Boyle 2009). Consequently, any software engineering approach for computational science should try to ensure that quality properties like maintainability are built into the software right from the beginning "quasi-automatically."

d) Creating a Shared Understanding of a "Code" Is Difficult
While all scientists eagerly document their scientific results in papers and technical reports, they typically do not produce documentation for the software they implement. User guides are created only in the less frequent case that the software is indented to be used by a larger user base outside of the research group of the original developers (Sanders and Kelly 2008). Instead

of relying on documentation, the scientists prefer informal, collegial ways of knowledge transfer to create a shared understanding of a piece of software. As the users and developers of scientific codes usually overlap, they can rely on a shared background knowledge. Therefore, the scientists find it harder to read and understand documentation artifacts than to contact the author of a certain part of the software and discuss their questions with them (Segal 2007).

The high personnel turnover rates in scientific software development, however, render such an informal knowledge transfer problematic. Most developers in this area are novices (Ph.D. students and early post docs) because scientists typically do not develop software for their whole careers. As they ascend the career ladder—and often move to other institutes—, their knowledge of the software becomes harder to access (Shull et al. 2005). This means that over and over again novices, without the help of any documentation material, have to familiarize themselves with codes that have not been written with program comprehension in mind (Carver et al. 2006; Segal 2007). Therefore, software engineering methods for computational science have to raise the abstraction level of the implementation artifacts produced by scientific developers to make these artifacts, at least to some extent, self-documenting.

e) Little Code Re-Use

Scientific software developers tend to rarely re-use code developed by others. Frameworks, for example for abstracting from the often tedious details of using MPI (Message Passing Interface Forum 2012), are not adopted because they make certain assumptions as to how their users should structure their code. The scientists fear that later on in a project, these structural assumptions could turn out to be too restrictive but cannot be circumvented. Instead, the researchers tend to re-develop such frameworks by themselves for every application to make them exactly match their needs (Basili et al. 2008; Carver et al. 2006). The same is true even for the use of software libraries. For example, many scientists implement their own linear algebra libraries while there are numerous well-tested, cache-optimized, parallel implementations available under open source licenses. Thereby, these scientists waste much effort on re-inventing existing technologies and, very likely, re-create them with inferior quality (Prabhu et al. 2011).

Limited re-use of existing code cannot only be observed for software developed by others but is even prevalent when it comes to the scientists' own. Because the majority of scientific codes is not programmed with comprehensibility in mind, scientists prefer re-writing code for new projects instead of spending a large amount of time on understanding the old one—even if they are the author of the old code (Segal 2007). Raising the level of abstraction in implementation artifacts could help to promote code re-use among scientists because it simplifies the comprehension process.

f) Disregard of Most Modern Software Engineering Methods
Surveys among scientific software developers show that they believe to have adequate software engineering knowledge to achieve their development goals. However, when asked about their knowledge and adoption of specific modern software engineering best practices and techniques (such as testing, profiling, and refactoring), both knowledge and adoption are relatively low. Therefore, it appears as if the scientists simply "don't know what they don't know" (Carver et al. 2013; Hannay et al. 2009). And even if the scientists are familiar with tools such as profilers, they rarely actually use most of them—either because of prejudice against the tools ("will not help") or because they think they do not really need them ("I know where time is spent in my code"; Prabhu et al. 2011).

But it is not just ignorance that leads to the non-adoption of software engineering methods. Many methods and tools are just not a good match for the scientists because their functioning is based on (often implicit) assumptions that are violated in the computational science context (Heaton and Carver 2015). Or they do not fit because they ignore the specific requirements that the scientists have (especially when it comes to tools that could support them). An example of the first type of mismatch due to wrong assumptions are software engineering processes that do not adequately consider the long life cycles of scientific software or the lack of up front requirements (Carver et al. 2007). IDEs for the HPC community are an example of the second mismatch due to neglecting the specific requirements for tools. The use of IDEs is limited in this community because the development environments usually do not feature convenient support for building, profiling, and deploying HPC applications on large-scale distributed systems. Therefore, the scientists only feel constrained by IDEs and, hence, do not adopt them (Carver et al. 2006; Prabhu et al. 2011).

The failure of software engineering to adequately address the needs of computational science leads to a situation in which the scientists are suspicious about software engineers' claims and overwhelmingly favor handcrafted solutions (Faulk et al. 2009). However, if the scientists are exposed to a certain software engineering technique that they find well-matched for their specific working environment, it is readily adopted. Examples of this are version control systems, regression testing frameworks that can be adapted to the scientists' needs for testing, and reuse in the small via libraries for equation solvers, mesh handling, etc. (Basili et al. 2008). In order to be accepted by the scientists, these tools must introduce a minimum of technicalities as the scientists are busy enough following the fast developments in their own field (Killcoyne and Boyle 2009).

All in all, we can conclude that software engineering approaches will only be adopted by scientists if these approaches honor the distinct characteristics and constraints of scientific software development which we described above.

6.3 What Software Engineering has to Offer to Computational Science

Our detailed analysis of the specific characteristics of scientific software development enables us to identify some shortcomings of existing proposals that are concerned with bridging the "chasm" between software engineering and computational science (Section 6.3.1). Based on this, we argue that there is a need of adapting existing software engineering methods for computational science and we give reasons why focusing on Model-Driven Software Engineering (MDSE) techniques is a promising starting point for this endeavor (Section 6.3.2).

6.3.1 Existing Attempts at Bridging the Software Chasm

Previous attempts at addressing the credibility and productivity crisis of computational science can be categorized into three groups:

1. Publish and review source code along with scientific articles to ensure reproducibility or at least repeatability of *in silico* experiments.

2. Let software engineers build or re-engineer (parts of) the scientific software.
3. Train scientists to enable them to use state-of-the-art software engineering methods.

In the context of the credibility crisis of computational science, a discussion about the *reproducibility* of scientific results that rely on computation emerged both within science itself (Peng 2011) and in the software engineering community (LeVeque et al. 2012). Being able to—at least in principle—validate the findings of other scientists by reproducing their experiments is at the heart of the scientific method. However, it is common practice in many areas of computational science *not* to release the source code on which the findings of a publication are based. This practice impedes the reproduction of published results or even renders it outright impossible. Therefore, several authors suggest to make public disclosure of the source code mandatory for peer-reviewed publications and some even propose to include the code itself in the peer review process (Barnes 2010; Ince et al. 2012; Morin et al. 2012). In the software engineering community, for example, several large conferences recently started employing a peer-reviewed implementation artifact evaluation process (Krishnamurthi and Vitek 2015).

These suggestions and efforts are certainly important steps in the right direction and could help to increase the appreciation of software and its quality in the computational science community. However, publishing source code alone does not adequately address the fundamental problem that the scientists lack the software engineering skills to tackle the underlying problems of both the credibility and the productivity crisis.

A second attempt to a solution is to try to have software engineers implement the software for the scientists. The experiences of Segal (2005), who put this approach to test, and the considerations given in Section 6.2.3 a) suggest that this is not a practicable way.

So far, the most promising attempt to solve the dual scientific software crisis seems to be education via workshop-based training programs focusing on Ph.D. students, such as the ones organized by Wilson (2014) and Messina (2015). While the education approach does address the skill gap that is central to the "software chasm," it does so with inadequate means. Our analysis in Section 6.2 clearly indicates that just exposing scientists to software engineering methods will not be enough because these methods

often fail to consider the specific characteristics and constraints of scientific software development. We therefore conclude that we have to select suitable software engineering techniques and *adapt* them specifically to the needs of computational scientists.

6.3.2 Embracing the Characteristics of Scientific Software Development

The results of our literature study clearly show that computational scientists are only "accidentally" involved in software development: ultimately, their goal is *not* to create software but to obtain novel scientific results (Section 6.2.3 c)). At the same time, however, they are very concerned about having full control over their applications and how these actually compute their results, which is why many prefer "older" programming languages with a relatively low level of abstraction from the underlying hardware (Section 6.2.2 b)).

Among the techniques and tools that software engineering has to offer, Model-Driven Software Engineering (MDSE) and specifically Domain-Specific Languages (DSLs) are promising starting points for addressing the needs of computational scientists. Since DSLs are designed to express solutions at the abstraction level of the domain, they allow the scientists to care about what matters most to them: doing science without having to deal with technical, implementation-specific details. While they use high-level domain abstractions, they still stay in full control over their development process as it is *them* who directly implement their solutions in formal and executable (e. g., through generation) programming languages. Additionally, generation from a formal language into a low-level GPL permits to examine the generated code to trace what is actually computed.

DSLs can also help to overcome the conflict between the quality requirements of performance on the one hand and portability and maintainability on the other hand, which is responsible for many of the difficulties experienced in scientific software development (Section 6.2.2 d)). DSL source code is maintainable because it is often pre-structured and much easier to read than GPL code, which makes it almost self-documenting. This almost self-documenting nature of DSL source code and the fact that it can rely on an—ideally—well-tested generator for program translation ensure the

reliability of scientific results based on the output of the software. Portability of DSL code is achieved by just replacing the generator for the language with one that targets another hardware platform. With DSLs, the high abstraction level does not have to result in performance losses because the domain-specificity first of all enables to apply—at compile time—domain-specific optimizations and greatly simplifies automatic parallelization (see Chapter 2).

In the way described above, DSLs integrated into a custom MDSE approach could help to overcome both the productivity and the credibility crisis of computational science. A first indicator that supports this hypothesis can be found in the survey report of Prabhu et al. (2011), who find that those scientists who program with DSLs "report higher productivity and satisfaction compared to scientists who primarily use general purpose, numerical, or scripting languages." In Chapter 7, we introduce an MDSE approach that takes into consideration the characteristics of scientific software development presented in this chapter and aims to take full advantage of the potential benefits of DSLs for computational science.

Chapter 7

The Sprat Approach: Hierarchies of Domain-Specific Languages

In the previous chapter, we demonstrated the existence of both a productivity and a credibility crisis of computational science. The challenges posed by these crises are most pressing for larger scientific simulation software projects which aim at incorporating a constantly increasing amount of scientific effects. We also showed that existing software engineering methods have to be adapted to the specific requirements of scientific software development to be of any real help for overcoming those challenges. As explained, MDSE techniques—and specifically DSLs—appear to be a good starting point for such adaptations.

Based on these findings, this chapter introduces *Sprat*, which is a model-driven software engineering approach targeted at collaborating scientists from different disciplines. The Sprat Approach aims to support computational scientists in efficiently implementing well-engineered simulation software without the need for extensive software engineering training. Its underlying idea is to provide a DSL for each (sub-)discipline that is involved in the development project and to integrate these modeling languages in a hierarchical fashion.

The rest of this chapter is structured as follows: in Section 7.1, we show that simulation software consisting of contributions from different scientific (sub-)disciplines can typically be constructed using a multi-layered software architecture in which the individual layers correspond to the contributions from the involved disciplines. As each layer in such an architecture can access entities only from the layer directly below, the contributions to the scientific software from the individual disciplines can be arranged in a strict hierarchy. Based on this observation, Section 7.2 introduces and formalizes

the concept of *DSL hierarchies*, which lays the methodological foundation for the Sprat Approach. Subsequent to the *method* of Sprat, we describe the engineering *process* that is associated with our approach (Section 7.3). We conclude by discussing the trade-offs which are necessary to keep the accidental complexity associated with Sprat to a minimum (Section 7.4).

Parts of this chapter are founded on previous publications regarding the Sprat Approach and its application (Johanson and Hasselbring 2014a,b).

7.1 The Architecture of Scientific Simulation Software

In this section, we demonstrate that typical scientific simulation software can be implemented using the multi-layered software architecture pattern (Buschmann et al. 1996). A software system conforms to this pattern if its components can be partitioned into a hierarchy of layers in which each layer corresponds to a particular level of abstraction of the system. In addition to that, every layer has to be implemented using only the abstractions of lower layers but never using abstractions from higher ones. Popular examples of the application of the layers pattern are networking protocols, which introduce layered levels of abstraction ranging from low-level bit transmission to high-level application logic.

We argue that the general structure of typical simulation software lends itself to the layers pattern and clarify this with an example. Additionally, we give evidence from the literature that existing implementations of complex climate models actually make use of the layers pattern.

7.1.1 Scientific Simulation Software in General

Generally speaking, scientific simulation software employs algorithms to analyze scientific models—i. e., mathematical abstractions of the real world—by means of computation. The scientific models can be formalized using different mathematical frameworks, such as differential equations, Individual-Based Models (IBMs) (Grimm and Railsback 2005), and Discrete Event System Specifications (DEVS's) (Zeigler et al. 2000). Based on the respective

mathematical framework and on the aspects that the model is supposed to be examined for, a suitable analysis algorithm is chosen.

For example, an ocean model would usually be based on the physical laws of fluid dynamics which are formulated as Partial Differential Equations (PDEs). An implementation of this model would employ a suitable PDE solver algorithm, such as an FCT FEM solver (see Chapter 12), and implement the concrete model equations using this solver.

Note that an analysis algorithm is appropriate not only for the specific model in question but for at least a whole sub-class of models in the respective mathematical modeling framework. Therefore, the analysis algorithm can be implemented independently of any concrete model and can be arranged in a way that the model component makes use of the algorithm component but not the other way around.

If additional scientific effects are to be included in the simulation, they can usually be interpreted as extensions to a base model. If, for example, sea ice is supposed to be included in an ocean model, it can be represented as a layer over the entire sea surface which may contain ice of variable thickness (Alexander and Easterbrook 2015). This layer would then influence certain processes that are modeled in the basic fluid dynamics equations.

Such model extensions introduce higher levels of abstraction and can be implemented atop the existing base model, which remains independent of the extension components. In this way, multiple model extensions can be stacked on top of each other, which leads to a layered software architecture as depicted in Figure 7.1.

7.1.2 Existing Global Climate Models

Alexander and Easterbrook (2015) analyze the software architecture of eight global climate models that represent both ocean and atmosphere. At the coarsest level, these complex simulations are divided into sub-systems such as *ocean* and *atmosphere* that are integrated horizontally via mediating coupler components (see Figure 7.2a). As these sub-systems are very coarse and all feature their own analysis algorithm/solver, they can be interpreted as independent simulations which are joined together horizontally.

A closer look at the individual sub-simulations reveals that each one of them in itself features a layered software architecture with hierarchically arranged components as described above. This can, for example, be seen in

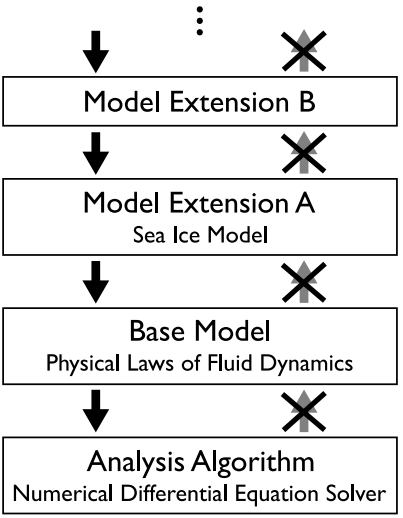
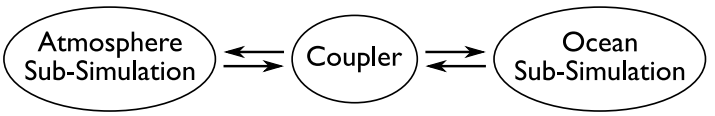


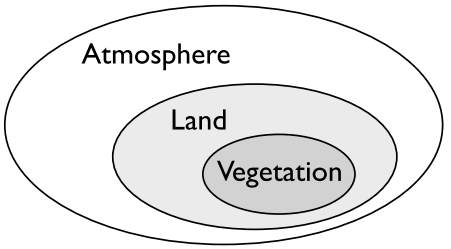
Figure 7.1. Usage relations in the layered architecture of scientific simulation software with examples for an ocean model.

the architecture diagram of the atmosphere sub-simulation of the MPI-ESM-LR model configuration depicted in Figure 7.2b. In this example, the *land* component extends the *atmosphere* sub-simulation, with the latter remaining independent from the former. The *land* component is, again, extended by the *vegetation* component in the same hierarchical fashion. From this, we can see that the multi-layered software architecture pattern is not only theoretically suitable for implementing scientific simulations but is actually applied in the implementation of complex global climate models.

Regarding the boundaries between the different components of the climate models, Alexander and Easterbrook (2015) point out that they "represent both natural boundaries in the physical world (e. g., the ocean surface), and divisions between communities of expertise (e. g., ocean science vs. atmospheric physics)." Therefore, the hierarchically arranged components in simulation software also belong to distinct scientific (sub-)disciplines. This, of course, does not only hold true for climate models but also applies to general simulation software: the analysis algorithm, the base model, and



(a) High-level architecture



(b) Detail of the atmosphere sub-simulation

Figure 7.2. Architecture diagrams for the MPI-ESM-LR model configuration. The atmosphere sub-simulation is depicted in detail to show nested components. Arrows indicate data exchange. Figures adapted from Alexander and Easterbrook (2015).

all model extension components are separated from each other along the boundaries of different "communities of expertise."

We will make use of the possibility to partition scientific simulation software along discipline boundaries into hierarchically arranged layers by constructing a DSL *hierarchy* that mirrors this hierarchical structure.

7.2 Hierarchies of Domain-Specific Languages

Even though scientific simulation software can typically be engineered using a layered architecture (or actually features such an architecture), "code modularity remains a challenge" (Alexander and Easterbrook 2015). This challenge arises because high performance is required and old programming languages are used (see Chapter 6). Schnetter et al. (2015) demonstrate this with the simple example of a solver for the scalar wave equation in first-

```
#pragma omp parallel for
for(i=1; i<N-1; i++) {
    dt_u[i]   = rho[i];
    dt_rho[i] = (v[i+1] - v[i-1]) / (2*dx);
    dt_v[i]   = (rho[i+1] - rho[i-1]) / (2*dx);
}
```

Listing 7.1. Code snippet from a fictitious C implementation of a finite difference solver for the wave equation.

order form given as

$$\partial_t u = \rho \tag{7.1}$$

$$\partial_t \rho = \delta^{ij} \partial_i v_j \tag{7.2}$$

$$\partial_t v_i = \partial_i \rho. \tag{7.3}$$

An efficient parallel implementation in C of a finite difference solver for this equation in one dimension would very likely contain a loop like the one in Listing 7.1. It is clearly visible that different concerns are mixed within these few lines of code: the physical model to be simulated (wave equation), the numerical approximation algorithm (finite difference method), and its mapping to hardware resources (memory layout of the vectors, parallelization via OpenMP (Dagum and Menon 1998)). A real world application would, of course, be much more complex and would, thus, contain even more intertwined concerns such as memory layout and communication/synchronization for distributed computing nodes. Currently, with low-level programming languages such as C, there is no straightforward way to evade this problem without negatively affecting performance levels.

The aforementioned problems with the modularization of scientific simulation software impede the realization of a layered software architecture that would clearly separate the concerns of different scientific (sub-)disciplines. This makes the software unnecessarily hard to maintain and hinders the cooperation of experts focusing on different scientific aspects of the simulation. Furthermore, it makes it difficult for scientists with only basic programming skills to participate in the development effort at all.

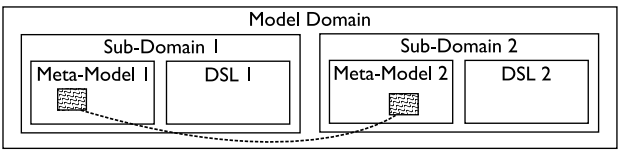


Figure 7.3. Horizontal integration of multiple DSLs. Figure adapted from Stahl and Völter (2006).

In order to meet these challenges, we propose a software engineering approach called *Sprat*, which is specifically designed for interdisciplinary teams of scientists collaborating on the implementation of scientific (simulation) software. Sprat introduces a DSL for each (sub-)discipline involved in the development project and integrates the languages in a hierarchical fashion based on the layered architectural structure of scientific software outlined in Section 7.1.

7.2.1 Foundations of DSL Hierarchies

Typically, DSLs are integrated horizontally as depicted in Figure 7.3 (Stahl and Völter 2006). In this way, a single domain can be divided into multiple sub-domains that share some common aspects of their respective domain meta-models. Through these shared concepts, the DSLs of the different sub-domains can interact with one another.

For our purpose, however, we need to integrate DSLs from completely different domains (such as numerical mathematics and fish stock modeling). To do so, we extend Stahl and Völter's (2006) concept of a *domain-specific platform* (see Section 2.2.2). Instead of having a single, pre-implemented domain-specific platform, we introduce multiple, vertically-aligned domain-specific layers that are semantically oriented towards each other as illustrated in Figure 7.4. Each layer is associated with a different DSL which is used to implement a certain part (defined by domain boundaries) of the software system to be constructed. Together, these layers form what we call a *DSL hierarchy*. The layers establish a hierarchy in the sense that at least a portion of the application part associated with each layer forms the (domain-specific) implementation platform for the part on the next higher level. This means that each layer uses abstractions provided by the next

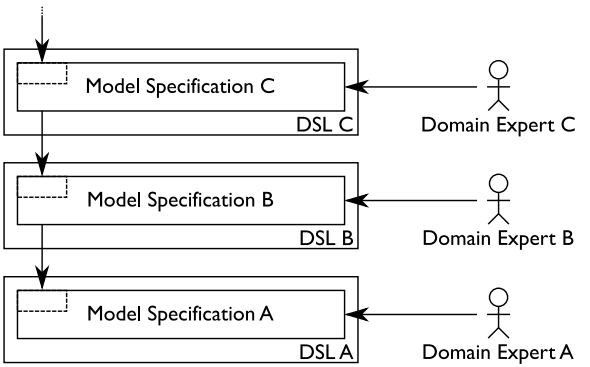


Figure 7.4. Multiple layers acting as domain-specific platforms for each other.

lower hierarchy level but never uses abstractions from higher levels. For a detailed description of how the levels of the hierarchy can interact and, thus, form domain-specific platforms for each other, see Sections 7.2.2 and 7.2.3.

Each level in a DSL hierarchy is associated with a modeler role which uses the DSL of the level to model the application part of this level. Together, the application parts of all hierarchy levels form the whole scientific simulation application to be implemented. Note that we assign a *role* to each level and not a *person*. This implies that a single person can fulfill multiple roles in a DSL hierarchy and one role can be assumed by several persons at once.

By employing an individual DSL for each discipline that is involved in an interdisciplinary scientific software project, we achieve a clear separation of concerns. Additionally, this ensures that all participating scientists (who assume modeler roles) are working only with abstractions that they are already familiar with from their respective domain. Due to the high specificity of a well-designed DSL, the code of an implemented solution that uses this language can be very concise and almost self-documenting. This simplifies writing code that is easy to maintain and to evolve, which allows scientists to implement well-engineered software without extensive software engineering training.

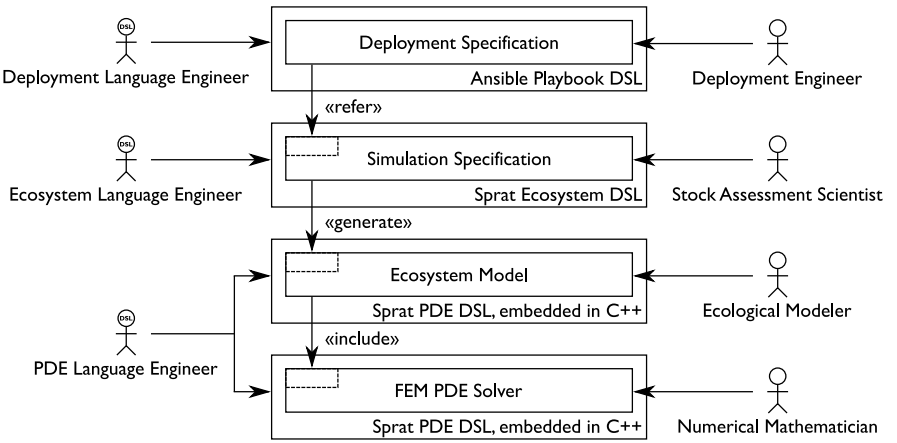


Figure 7.5. DSL hierarchy for the Sprat Marine Ecosystem Model.

7.2.2 An Example Hierarchy

Before giving a formal definition of the term *DSL hierarchy* in the next section, we complete our informal introduction to the notion by presenting an example of such a hierarchy. For this purpose, we depict in Figure 7.5 the DSL hierarchy for the Sprat Marine Ecosystem Model, which is introduced in Part III of this thesis. Regarding details of the notation that we use for drawing DSL hierarchies, see Section 7.2.4. For additional information concerning the different DSLs of the hierarchy, refer to Chapter 8.

The Sprat Marine Ecosystem Model is based on Partial Differential Equations (PDEs) and introduces fish into existing biogeochemical ocean models. Four different disciplines are involved in the implementation and application of the corresponding simulation software (see the right side of Figure 7.5). At the basis of the hierarchy, we find the role of the Numerical Mathematician, who models a special-purpose FEM PDE solver for the model equations. The solver is implemented using the Sprat PDE DSL, which is embedded into C++.

Using the abstractions provided by the bottommost level, the Ecological Modeler implements the concrete equations to be solved for the ecosystem model. Since both the Numerical Mathematician and the Ecological Modeler

work with the same abstractions (mathematical equations) to express their application parts, the ecosystem model is implemented with the same DSL as the FEM solver. The interaction between the first and the second layer is of the type *inclusion*: the higher level reuses existing abstractions from the lower level in the same DSL by including them into the model on the higher level.

To apply the simulation to a specific ecosystem (say, the Baltic Sea), it has to be parametrized by a fish stock assessment scientist for that particular ecosystem. For this purpose, the Stock Assessment Scientist creates an ecosystem simulation description using the external Sprat Ecosystem DSL. From such a description, information that is missing for a simulation to be complete is *generated* on the second hierarchy level.

While the first three layers complete the ecosystem simulation as such, it is still undefined how to build and execute the simulation in a (possibly distributed) compute environment. To formally describe this process, the Deployment Engineer models a deployment specification using the external Ansible Playbook DSL. Such a specification interacts with the other levels of the DSL hierarchy by *referring* to names of model artifacts without assuming any knowledge about the internal structure (i. e., the meta-model) of these models. This level of knowledge is sufficient to, for example, compile application parts.

One could argue that the deployment is a concern orthogonal to the implementation of the simulation and should, hence, not be included in the DSL hierarchy. We decided to incorporate the deployment into the hierarchy nonetheless because it allows us to have a single structure that can be used to abstractly describe to the scientists the whole development process of the simulation up to its execution. This is part of the effort to minimize the accidental complexity of the Sprat Approach (for a more detailed discussion of this aspect, see Section 7.4).

The last elements of Figure 7.5 which we have not discussed yet are the language engineer roles. Each DSL has a language engineer role assigned to it that is responsible for the design, implementation, and maintenance of the language. For a description of the individual tasks of a language engineer role and of how to assign this role, see Section 7.3.

7.2.3 Formal Specification

So far, our introduction to DSL hierarchies—which are the core of the Sprat Approach—has been an informal prose explanation. While informal explanations are well-suited to give an intuitive overview of a topic, they may be interpreted differently by different people and, thus, leave room for uncertainties. To eliminate these uncertainties, this section presents a rigorous formal specification of the concept of a DSL hierarchy, including a detailed analysis of how the different levels of such a hierarchy can interact with each other.

In the formalization of the DSL hierarchy notion, we follow Hasselbring (2015), who combines a semi-formal description using the Unified Modeling Language (UML) (Object Management Group 2015c) with a formal specification in Object-Z (Smith 2000). Figure 7.6 uses the UML to provide a graphical representation of the meta-model of DSL hierarchies. Such a diagram provides a top-down overview on the involved abstractions and their relationships. In itself, however, the diagram is only a *semi-formal* specification because the semantics of the UML notation are not specified formally. Additionally, the UML only allows to define simple constraints on the meta-model, for example, via the use of multiplicities. In order to prescribe more complex constraints—such as that all artifacts of an application part must belong to the DSL of the corresponding hierarchy level—, a textual formal specification language like Object-Z or the Object Constraint Language (OCL) (Object Management Group 2014) is necessary. We choose Object-Z over the OCL because an Object-Z specification stands for its own (the UML diagram in Figure 7.6 acts only as an overview), while an OCL specification usually has to reference elements from a UML diagram.

Object-Z is an object-oriented extension to the specification language Z (Spivey 1992) which offers a notation for formally describing the behavior of a system using first order predicate logic and set theory. Object-Z is a strongly typed language that follows the principle of *definition before use*. Therefore, Object-Z specifications have to be presented in a bottom-up style beginning with the most basic abstractions of which the higher-level concepts are composed step by step. To maintain a global perspective on the formal specification of the DSL hierarchy notion and its parts, we suggest to refer back to Figure 7.6 from time to time. For a short introduction to the Z and Object-Z notation, we recommend the corresponding chapters

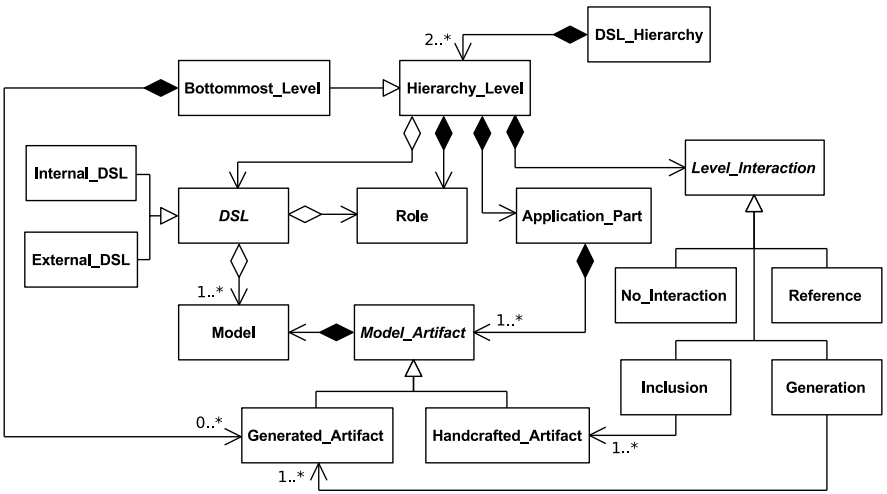


Figure 7.6. Meta-model for the concept of DSL hierarchies using the UML notation for class diagrams (attributes of classes not shown, default multiplicity is 1).

of Derrick and Boiten (2014). To ensure type-correctness, we validated our Object-Z specification using the Community Z Tools.[1] Note that on some of the following pages, there is blank space at the bottom to avoid breaking class schemata.

The most basic abstractions used in the specification of a DSL hierarchy are *models* and *names*. Since these abstractions are atomic for our modeling purposes, we do not say anything about their internal structure and introduce these types as *given sets*.

$$[MODEL, NAME]$$

Additionally, we introduce *roles* that feature a descriptive name.

[1] `http://czt.sourceforge.net`

Role

name : *NAME*

With these three types, we can construct a *DSL*, which mainly is a non-empty set of models ($\mathbb{P}$ stands for the power set) complemented with a name and a language engineer role.

DSL

name : *NAME*
models : $\mathbb{P}_1$ *MODEL*
languageEngineer : *Role*

We further differentiate between *internal* and *external* DSLs. An internal DSL has the name of its host language as an attribute (in addition to the attributes inherited from the *DSL* type).

Internal_DSL

DSL

hostLanguage : *NAME*

External_DSL

DSL

Models are represented by *model artifacts*, which make the former accessible via a given name.

Model_Artifact

name : *NAME*
model : *MODEL*

Artifacts can be *handcrafted* or automatically *generated*.

Handcrafted_Artifact	*Generated_Artifact*
Model_Artifact	*Model_Artifact*

A non-empty, finite set of model artifacts combined with a name forms an *application part* of the whole software to be implemented. The artifacts of an application part are required to be handcrafted or generated (which essentially makes the *Model_Artifact* class abstract). We give some comments regarding the notation used in the definition of the *Application_Part* class:

- $\mathbb{F}\, S$ is the set of all finite subsets of the set S.
- $o : \downarrow C$ means that o refers to an object from class C or from any of the classes that are inherited from C.
- The "©" symbol signifies *containment*. For the *artifacts* attribute of the *Application_Part* class this means that the same artifact cannot occur in two different application parts.

Application_Part

$$
\begin{array}{|l}
\hline
name : NAME \\
artifacts : \mathbb{F}_1(\downarrow Model\_Artifact©) \\
\hline
artifacts \subseteq Handcrafted\_Artifact \cup Generated\_Artifact \\
\hline
\end{array}
$$

We have already seen in Section 7.2.2 that there are different ways in which hierarchy levels can interact with each other: either by means of *generating* models, *including* models, or *referencing* model artifacts by name. In the first two cases of generation and inclusion, the *level interaction* has to be aware of the meaning of the models it targets. In the last case of reference via names, the target artifacts remain black boxes for the interaction. The first two cases are handled by the *targets* attribute of the *Level_Interaction* class while the latter case is modeled by *targetNames*. We have to define both attributes in

the superclass *Level_Interaction* because a variable

$interaction : \downarrow Level\_Interaction$

can only be used in expressions where *any* of its possible types could occur.

Level_Interaction

$$targets : \mathbb{F}(\downarrow Model\_Artifact)$$
$$targetNames : \mathbb{F}\, NAME$$

For a level interaction of *generation* type, there has to be at least one target artifact and all target artifacts must be generated ones.

Generation

Level_Interaction

$$targets \neq \varnothing$$
$$targets \subseteq Generated\_Artifact$$
$$targetNames = \varnothing$$

The same holds true for the *inclusion* type except that the target artifacts must be handcrafted (since there are no generated ones).

Inclusion

Level_Interaction

$$targets \neq \varnothing$$
$$targets \subseteq Handcrafted\_Artifact$$
$$targetNames = \varnothing$$

The *reference* interaction type must only contain names of target artifacts, which prevents the internal structure of those artifacts from being accessible.

Reference

Level_Interaction

$targets = \varnothing$
$targetNames \neq \varnothing$

Lastly, we need an interaction type that represents *no interaction* (for the bottommost layer).

No_Interaction

Level_Interaction

$targets = \varnothing$
$targetNames = \varnothing$

With the definitions from above, we have all the necessary abstractions to specify a *hierarchy level*. The latter consists of a modeler role that uses either an internal or an external DSL to model an application part. Furthermore, the hierarchy level interacts with other levels of the hierarchy and may contain additional generated artifacts (which is only relevant for the bottommost level).

Hierarchy_Level

$dsl : Internal\_DSL \cup External\_DSL$
$modeler : Role©$
$appPart : Application\_Part©$
$interaction : \downarrow Level\_Interaction©$
$genArtifacts : \mathbb{F}\ Generated\_Artifact©$

All artifacts of the application part must belong to the DSL of this level.
$\forall a : appPart.artifacts \bullet a.model \in dsl.models$

The *bottommost level* of a *DSL* hierarchy either has no interaction with other levels at all or employs a DSL that generates artifacts. In the latter case, these artifacts are cataloged in the *genArtifacts* attribute.

Bottommost_Level

Hierarchy_Level

$interaction \in Generation \cup No\_Interaction$

The artifacts generated on this level have to match *genArtifacts*. If $interaction \in No\_Interaction$, this implies $genArtifacts = \varnothing$.
$interaction.targets = genArtifacts$

With all the previous abstractions at hand, a *DSL hierarchy* can now be described as a sequence of hierarchy levels with some additional constraints regarding their relationships. Concerning the Object-Z notation of the *DSL_Hierarchy* class, we specify its *levels* attribute as an "iseq," which stands for *injective sequence*.

Sequences in Object-Z are represented as a set of tuples containing an index and the actual element of the sequence. Thus, for any non-empty set $S \neq \varnothing$, sequence $s : \text{seq}\, S$, and sequence element $e : s$, it holds that $e \in \mathbb{N} \times S$. The index (in $\mathbb{N}$) of element e can be accessed via $first(e)$, and the actual element (in S) via $second(e)$. The j-th element of sequence s is given by $s(j)$

(which is in S). The *range* of a sequence $\operatorname{ran} s$ is the set of all elements of S that appear in s. In an *injective* sequence $is : \operatorname{iseq} S$, each element of S can appear at most once. The indices of a sequence are continuously numbered starting with 1.

To form a hierarchy, there have to be at least two hierarchy levels and the interactions between levels must always be top-down. For generation and inclusion relationships, the target artifacts must reside in the level directly below. The reference interaction can target artifact names from all the levels below (therefore, artifact names have to be *globally* unique). For the inclusion interaction, it is important that the DSLs of the involved levels are compatible in the sense that models on the lower level are part of the DSL on the higher one. However, we do not require the DSLs of both levels to be the same because this would unnecessarily limit the use of language composition (e. g., via language inheritance; cf. Efftinge et al. 2012).

DSL_Hierarchy

$levels : \mathrm{iseq}(\downarrow Hierarchy\_Level©)$

$\#levels \geqslant 2$

The bottommost (but only the bottommost) level is of corresponding type.

$\forall\, l : levels \bullet first(l) = 1 \iff second(l) \in Bottommost\_Level$

All levels (except the bottommost) must *not* have any additional generated artifacts and must interact with lower levels either by means of generation, inclusion, or reference.

$\forall\, l : levels \mid first(l) > 1 \bullet (second(l)).genArtifacts = \varnothing \wedge$
$(second(l)).interaction \in Generation \cup Inclusion \cup Reference$

Names of artifacts must be globally unique.

$\forall\, a, b : \{x : \downarrow Model\_Artifact \mid \exists\, l : \mathrm{ran}\, levels \bullet x \in l.appPart.artifacts \vee$
$x \in l.genArtifacts\} \mid a.name = b.name \bullet a = b$

If a level above the bottommost generates artifacts, the generated artifacts must match the ones on the level below; and if a level has generated artifacts, they must stem from generation on the level above.

$\forall\, l : levels \mid first(l) > 1 \wedge (second(l)).interaction \in Generation \bullet$
$(second(l)).interaction.targets =$
$(levels(first(l) - 1)).appPart.artifacts \cap Generated\_Artifact$

$\forall\, l : levels \mid (second(l)).appPart.artifacts \cap Generated\_Artifact \neq \varnothing \bullet$
$\#levels > first(l) \wedge$
$(levels(first(l) + 1)).interaction \in Generation$

If a level includes artifacts, they must be handwritten and be present on the level below and the DSLs of the two levels must be compatible.

$\forall\, l : levels \mid (second(l)).interaction \in Inclusion \bullet$
$(second(l)).interaction.targets \subseteq$
$(levels(first(l) - 1)).appPart.artifacts \cap Handcrafted\_Artifact \wedge$
$(levels(first(l) - 1)).dsl.models \subseteq (second(l)).dsl.models$

If a level refers to artifacts from lower levels by name, the referenced artifacts must exist.

$\forall\, l : levels \mid (second(l)).interaction \in Reference \bullet$
$\forall\, n : (second(l)).interaction.targetNames \bullet$
$\exists\, m : levels \bullet first(m) < first(l) \wedge$
$(\exists\, a : (second(m)).appPart.artifacts \bullet a.name = n \vee$
$\exists\, a : (second(m)).genArtifacts \bullet a.name = n)$

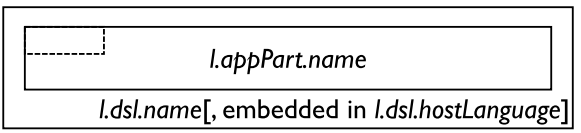


Figure 7.7. Concrete graphical syntax for hierarchy level l : ran $H.levels$.

(a) Modeler role

(b) Language engineer role

Figure 7.8. Concrete graphical syntax for modeler and language engineer roles.

7.2.4 Suggested Notation

In this section, we introduce a graphical notation for DSL hierarchies, of which we have already seen an example in Figure 7.5 on page 83. We use the Object-Z specification from above to describe how to visualize H : *DSL_Hierarchy* and its elements (though not all elements are explicitly visible, as we will see).

Each hierarchy level l : ran $H.levels$ is depicted as shown in Figure 7.7. The outer box represents the hierarchy level l itself. In its lower right corner, we mention the name of the DSL of this level. If the DSL is internal ($l.dsl \in Internal\_DSL$), we append the name of the host programming language. The inner box represents the application part $l.appPart$ and bears its name. The smaller region inside the application part box (delimited by the dashed line) represents the fraction of the artifacts of this application part that higher levels interact with. If there are no hierarchy levels above level l, we omit this small region.

The modeler role $l.modeler$ is depicted using the UML syntax for an *actor* (Object Management Group 2015c) along with the name of the role (see Figure 7.8a). Similarly, for each dsl : DSL that is referenced by a

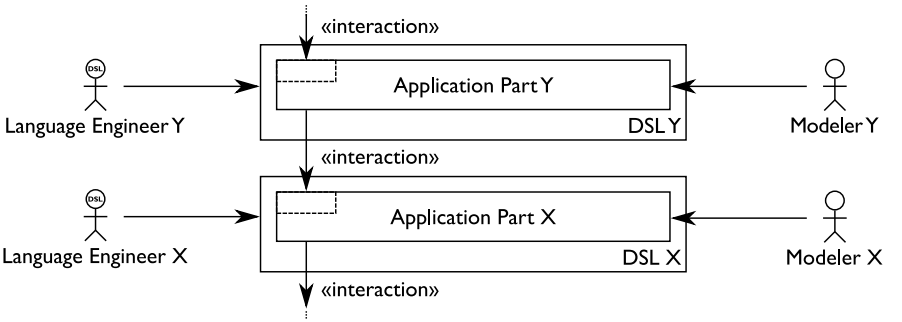


Figure 7.9. Concrete graphical syntax for associations between elements of a DSL hierarchy.

hierarchy level in ran $H.levels$, we visualize the role $dsl.languageEngineer$ of each language engineer as shown in Figure 7.8b.

The different hierarchy levels of H are arranged vertically in the order they appear within the sequence $H.levels$, starting with the first level at the bottom. For each level $L : H.levels$ with $first(L) > 1$, we draw an association arrow from the application part box (the inner one) to the small region delimited by the dashed line within the application part box of the next lower layer (which is $H.levels(first(L) - 1)$). We put either «generate», «include», or «refer» next to this association arrow depending on the type of $(second(L)).interaction$ (see Figure 7.9).

For simplicity, we omit the «generate» association that the bottommost level can have with itself. As another simplification of the notation, we let «refer» associations point towards the level directly below without considering which (other) lower levels they may actually target (regarding our reasons for doing so, see Section 7.4).

Lastly, we add association arrows from the modeler roles to their corresponding application part box (the inner one) and from language engineer roles to the corresponding hierarchy level box(es) as shown in Figure 7.9.

7.3 Applying the Sprat Approach

This section describes the engineering process of the Sprat Approach, which builds upon the concept of hierarchies of DSLs introduced above. Sprat acknowledges that computational scientists want to have full control over the implementation of their simulation software. At the same time, however, it also recognizes that computational science will not be able to face its current challenges related to its productivity and credibility alone (see Chapter 6). To mediate between the desire for independence on the one hand and the need for assistance on the other hand, the Sprat Approach allows the scientists to continue developing their simulations on their own but with programming languages specifically designed to help them create well-engineered software. Therefore, the Sprat Process, which is shown in Figure 7.10, involves both scientists and DSL engineers (Kleppe 2008), with the latter playing a supporting role.

7.3.1 Separating Concerns

Scientists typically only have a "vague idea" (cf. Figure 6.1 on page 59) of the simulation software they need for answering their scientific questions. However, such a very general idea is sufficient to construct a DSL hierarchy for the software project. The first step in doing so is for the team of scientists to identify the scientific (sub-)domains that correspond to the classes of scientific effects that need to be modeled.

In a second step, these domains are arranged hierarchically as described in Section 7.1. In some cases (especially for very large and complex simulation software), not all sub-systems of the software can be organized in a *single* hierarchy. In these cases, however, one can usually identify independent sub-simulations that have to be coupled horizontally but in themselves, again, exhibit a hierarchical structure (cf. Section 7.1.2). This means that while coupler code has to be written to connect the bottommost levels of two (or more) DSL hierarchies, the Sprat Process can still be applied to each of the DSL hierarchies individually (therefore, we do not specifically mention this case in Figure 7.10).

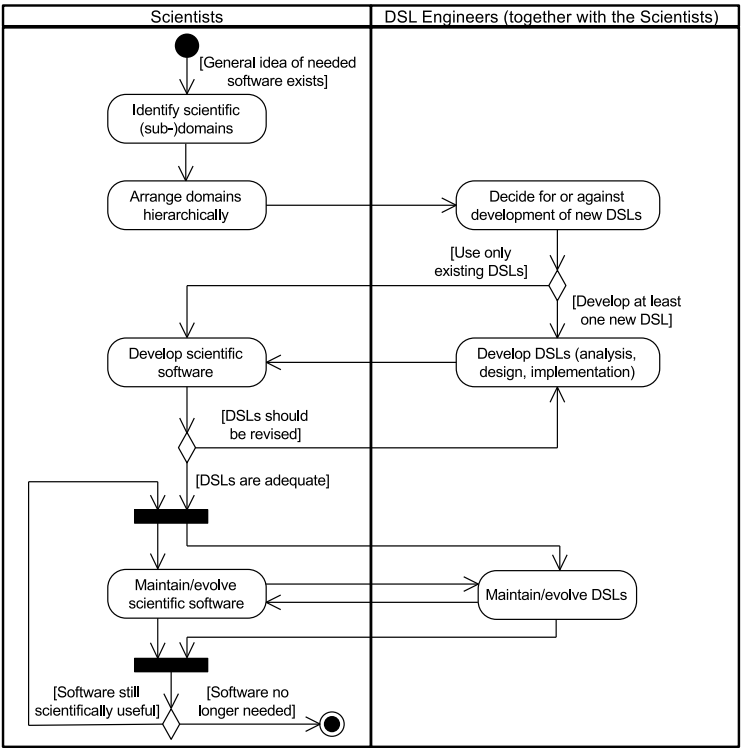


Figure 7.10. Engineering process of the Sprat Approach.

7.3.2 Determining Suitable DSLs

Once the levels of the DSL hierarchy and their corresponding application parts have been established, the language engineers must determine whether or not suitable DSLs for the target domains already exist (adopting an existing DSL obviously requires much less effort than creating a new one). For this purpose, Mernik et al. (2005) give a collection of patterns that can act as guidelines for deciding whether to develop a new DSL in a given situation. Note, however, that for this activity, the DSL engineers have to take into account a number of factors that are not commonly considered

for DSL selection but are of special importance in the context of scientific software development:

1. Many computational scientists are reluctant to adopt "newer" technologies (for the reasons given in Section 6.2.2 b)). Therefore, it has to be ensured that the technologies associated with candidate DSLs are accepted among the computational scientists who are supposed to use them.
2. The DSLs have to integrate well with the tools and workflows that the scientists are used to.
3. Candidate DSLs have to be easy to learn for domain specialists (the concrete syntax must appear "natural" to them) and offer good tool support. In this way, the scientists require only minimal training to use the languages.
4. As performance is a very important quality requirement in computational science (see Section 6.2.2 d)), it must be made sure that the increased level of abstraction of a candidate DSL does not compromise the runtime performance of programs significantly. Additionally, the DSL should introduce as few dependencies as possible.
5. The language engineers must ensure that candidate DSLs can be integrated with each other vertically in a DSL hierarchy.

Clearly, the language engineers have to cooperate closely with the scientists and obtain feedback from them continuously to make sure that the selected DSLs actually meet the needs of the scientists and that the latter are really willing to use the languages. For this reason, it is important for the language engineers to know about and to respect the characteristics of software development in computational science, which we discussed in Section 6.2.

If no suitable DSLs can be identified for some or all levels of the DSL hierarchy, the language engineers have to develop corresponding languages by themselves. In principle, the development of DSLs for computational science is not different from DSL engineering for other domains. Generally, the DSL development process can be divided into a domain analysis, a language design, and an implementation phase, for which Mernik et al. (2005) identify several patterns. A more detailed approach to DSL engineering that focuses on meta-modeling is given by Strembeck and Zdun (2009). Of course, for DSL development the language engineers have to pay special attention to the same factors that were already discussed above in the context of DSL

selection for scientific software development. Again, it cannot be overemphasized that the language engineers have to work in close collaboration with the scientists all the time and that they have to respect (and at least partially embrace) the characteristics of scientific software development. For a DSL to be accepted by the target user community, the accidental complexity (both linguistic and technical) introduced along with it must be kept to a minimum.

Concerning the order in which the DSLs should be constructed, we generally propose to develop all languages of the language hierarchy at the same time. Preferably, the development of each DSL takes place in an incremental fashion using agile methods. This approach provides large flexibility because potential incompatibilities between different languages in the DSL hierarchy can be addressed early on. Since DSLs on higher levels of the hierarchy depend on those on lower ones, each development iteration for the languages should begin on lower hierarchy levels moving on to higher ones.

7.3.3 Development and Maintenance

After DSLs have been assigned to or created for all hierarchy levels, the scientists start to implement the simulation software by assuming the different modeler roles of the DSL hierarchy. The Sprat Process does not impose any restrictions on this activity as this would very likely lead to the rejection of the whole approach (see Section 6.2.1 c)).

If it turns out during the development that some of the DSLs are insufficient for the implementation (e. g., missing elements in the meta-model or an overly technical concrete syntax), the languages have to be adapted by the language engineers. For externally developed DSLs, this very likely means that they have to be replaced by another DSL (possibly one developed "in-house"). This iterative process of the adaptation of the DSLs continues until the simulation software is "finished" in the sense that it can answer the scientific questions it was designed for (or the ones that emerged along the way during the implementation).

After reaching a state of relative maturity and stability, the simulation software enters its maintenance phase. In this phase, the number of changes applied to the software per unit of time is typically much lower than during the initial implementation phase. Note, however, that especially

in computational science, the boundaries between the development and maintenance phase are rarely clear-cut (see Section 6.2.2 a)).

During the maintenance phase, the simulation software is evolved in order to enable answering new scientific questions. Typically, the DSLs of the hierarchy should be able to support the changes to be introduced to the simulation. However, if previously ignored aspects of a domain have to be included in the simulation, also the DSLs have to be evolved in parallel to the scientific software. Minor maintenance tasks regarding a DSL can potentially be carried out by the domain experts themselves if the language has been designed with this option in mind (see, for example, our experiment concerning DSL maintenance by domain experts in Chapter 14).

New scientific questions could also make it necessary to add new levels to the DSL hierarchy because it may be required to model effects from totally different domains. In this case (which is not depicted in Figure 7.10 for reasons of clarity), one would have to start with the decision for or against the development of a new DSL for this level. The rest of the process for this specific hierarchy level would be the same as for the other levels.

After each maintenance iteration, when the new scientific questions could—or could *not*—be answered, the question arises whether it is still scientifically useful to maintain the simulation software. Depending on the answer to this question, either a new maintenance iteration is started or the software is not developed any further and the Sprat Process comes to its end.

7.4 Preventing Accidental Complexity

Segal (2008) reminds us that software engineers "should not try to impose the full machinery of traditional software engineering on scientific software development." Any tool or development approach which assumes that scientific programmers will invest time and effort into mastering it is deemed to fail because "scientists tend to want results immediately" (Prabhu et al. 2011). Therefore, Prabhu et al. (2011) conclude that while educating scientists in software engineering methods is worthwhile, "a more promising approach is to develop solutions that are customized to the requirements of scientists" and "require little training." Such solutions have to adopt the frame of reference of the scientists and must necessarily make

compromises with regard to their generality and formality (Kelly 2007). If a tool confronts scientists with too many formal software engineering complexities—which might seem natural for a software engineer but are "accidental" from a scientist's perspective—, the tool will inevitably face rejection (Wilson 2006b).

The Sprat Approach achieves a compromise between formality and pragmatism by making two central concessions. First, we do not impose any restrictions on the concrete development activities of the scientific programmers, as discussed in Section 7.3.3. Second, we refrain from too much formality in the artifacts that are necessary for carrying out a scientific software development project with the Sprat Approach.

The only artifact that the scientists produce together with the language engineers to communicate the development process among themselves is a diagram of the DSL hierarchy following the notation introduced in Section 7.2.4. Therefore, *all* development aspects have to be represented in this diagram. This includes even concerns that could be modeled as orthogonal to the actual development of the software, such as the deployment process (cf. Figure 7.5 on page 83). Thus, the hierarchy diagram represents a combination of different concerns and even mixes structural and procedural elements (e. g., x must be present before y can be deployed). Also, we omit some details from the diagram (such as some associations for reference relationships) that would probably render the diagram too inaccurate in the eyes of some software engineers. This approach minimizes the complexity that the scientific programmers have to deal with but still enables meaningful reasoning about the software, its development process, and the different responsibilities of the personnel involved.

Another concern regarding the applicability of the Sprat Approach is that it depends on the availability of trained DSL engineers. We discuss this aspect in Section 18.1.5 of Chapter 18.

Chapter 8

Domain-Specific Languages for a Marine Ecosystem Model

This chapter introduces the DSLs that are used for applying the Sprat Approach (Chapter 7) to the implementation of the Sprat Marine Ecosystem Model (Chapter 11), which serves as our evaluation example for the software engineering part of this thesis. For this purpose, Section 8.1 establishes an overview on the DSL hierarchy for the development of the ecosystem model and describes key requirements for the modeling languages. The remaining sections of this chapter give detailed information on the individual DSLs of the language hierarchy; namely on the Sprat PDE DSL in Section 8.2, on the Sprat Ecosystem DSL in Section 8.3, and on the Ansible Playbook DSL in Section 8.4.

Concerning the Sprat PDE DSL and the Sprat Ecosystem DSL, which were developed as part of this thesis, we give insights into the meta-models of the languages and into important aspects related to their implementation. For the reused Ansible Playbook DSL, we introduce the core concepts of the language and describe its application in the deployment of the Sprat Marine Ecosystem Model.

Note that the presentation of the DSLs in this chapter does not provide complete formal specifications of the languages. For the two DSLs that were developed as part of this thesis, the corresponding implementations serve as their reference. These implementations can be obtained online (Johanson 2015d). An extensive documentation of the Ansible Playbook DSL is available from Ansible Incorporated (2015).

8.1 The DSL Hierarchy for the Sprat Marine Ecosystem Model

The aim of the Sprat Marine Ecosystem Model is to introduce fish into existing biogeochemical ocean models. This is supposed to facilitate studying both bottom-up and top-down interactions between the different biotic and abiotic components of marine ecosystems and how these interactions are influenced by changes in the environment (e. g., induced by climate change). For this purpose, we choose a modeling approach based on systems of PDEs (see Chapter 10). As described in Section 7.3, these few pieces of high-level information alone are sufficient to deduce the fundamentals of a DSL hierarchy for the Sprat Marine Ecosystem Model in the context of the Sprat Approach. Since the ecosystem model is PDE-based, we know that we need (the role of) a numerical mathematician to develop a numerical solver for the type of equation system employed by the model. The PDE solver is used by the role of the ecological modeler, who has to implement the concrete equations of the fish stock model. For the model to be then applied to a specific ecosystem, it has to be parametrized by the role of a fish stock scientist. Lastly, in order to be executed, a deployment engineer has to specify how the model is deployed in a (possibly distributed) compute environment.

The resulting DSL hierarchy and the interactions of the four different levels have already been explained in detail in Section 7.2.2 of the previous chapter and are, again, depicted in Figure 8.1. Complementary to this description, the current section focuses on how to design and reuse the DSLs employed on the different hierarchy levels in order to implement the ecosystem model.

8.1.1 A DSL for the Numerical Mathematician and the Ecological Modeler

Since standard off-the-shelf solvers have problems with accurately approximating the solution of the special type of PDE system employed in the Sprat Marine Ecosystem Model, we have to design (and implement) our own special-purpose solver (see Chapter 12). A suitable DSL for the implemen-

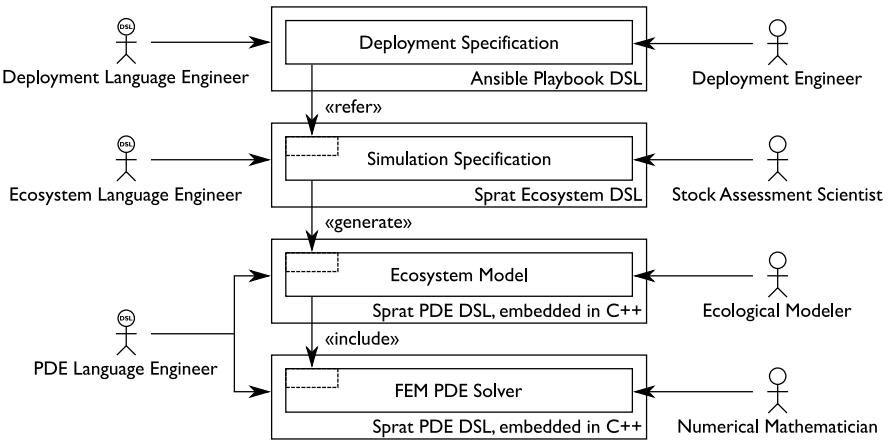


Figure 8.1. DSL hierarchy for the Sprat Marine Ecosystem Model.

tation of our special-purpose solver, which is based on a Finite Element Method (FEM), must fulfill the following requirements:

1. The abstraction level of the modeling language must *not* be as high as that of a language for modeling PDEs in general. Instead, it must focus on the domain of *mesh-based (FEM) PDE solvers*, which has a comparably lower level of abstraction. This is because we do not just want to model PDEs and have them solved by the DSL but we want to have full control over *how* the equations are solved.
2. The DSL has to support PDE domains with more than three spatial dimensions (for details, see Chapter 11).
3. The language should be embedded into a GPL that allows the user to easily implement and interface with problem-specific data structures and algorithms outside the relatively narrow domain of mesh-based PDE solvers. The host language has to be well-accepted in computational science (specifically in numerical mathematics).
4. The DSL should be textual because graphical languages are generally not well-accepted by the targeted HPC experts (cf. Section 6.2.2 b)). Additionally, all established GPLs in the domain are textual languages.

There already exists a number of DSLs—both internal and external—for the domain of PDE solvers. These languages, however, fail to meet the requirements stated above, as discussed in Section 17.2.1 of the related work chapter. To address this gap, we developed the Sprat PDE DSL, which is embedded into C++ (see Section 8.2). In the implementation of the Sprat Marine Ecosystem Model, this language is not only used by the numerical mathematician on the bottommost hierarchy level but also by the ecological modeler on the level above. The two roles share the same DSL because both of them work with mathematical equations, which is the abstraction level of the language. In the design of the Sprat PDE DSL, we payed close attention to making the language equally suitable for experts from both disciplines.

8.1.2 A DSL for the Fish Stock Assessment Scientist

Fish stock assessment scientists apply the Sprat Marine Ecosystem Model to a concrete ecosystem in order to answer scientific questions about that system. For this purpose, they have to give a detailed description of the intended simulation run which has to include the parameters of all fish species and of the ecosystem itself (e. g., the topography of the ocean region in question) as well as which simulation output data is supposed to be aggregated and recorded.

Stock assessment scientists can generally be assumed to be less technology-affine than, for example, numerical mathematicians and they are usually not well-versed in GPLs. However, acceptance of new technologies is typically much higher among stock assessment scientists (who often have a background in a subject related to biology) than among most HPC experts (Prabhu et al. 2011). For these reasons, an external DSL with a concise declarative syntax is a good fit for the third level of the DSL hierarchy for the Sprat Marine Ecosystem Model. Another reason why an external DSL is well-suited is that it can easily be integrated with the C++ part of the simulation through code generation without any loss of runtime performance.

We decided to develop a new external DSL (the Sprat Ecosystem DSL) to be used by the role of the stock assessment scientist (see Section 8.3). Of the few existing DSLs that target the domain of ecosystem simulation specifications, NetLogo (Wilensky 1999) comes closest to fulfilling our requirements (see Section 17.2.2 in the related work chapter). However, NetLogo is designed to procedurally model agent-based simulations alongside with their

whole simulation logic instead of just declaratively specifying parametrizations of arbitrary ecosystem simulations. Additionally, NetLogo is based on the Java Virtual Machine (JVM), which makes its integration with the C++ part of our simulation unnecessarily cumbersome.

Our reference implementation of the Sprat Ecosystem DSL (Johanson 2015d) uses a textual modeling interface. This decision is justified by the wide-spread use of textual scripting languages in the domain. However, it is worth considering a semi-graphical version of the Sprat Ecosystem DSL. A semi-graphical prototype developed with JetBrains MPS (cf. Chapter 2) by a student as part of his master's thesis, which was supervised in the context of this thesis, showed promising results in this respect (for details refer to Section 19.1.4 of Chapter 19).

8.1.3 A DSL for the Deployment Engineer

The role of the deployment engineer has to automate the deployment of the ecosystem simulation in complex distributed computing environments (possibly using cloud computing resources). There already exists a number of different DSLs for this purpose (see Section 17.2.3 in the related work chapter), out of which the Ansible Playbook DSL[1] is most appropriate for us.

In contrast to most other solutions, Ansible uses a push rather than a pull scheme for applying changes to nodes, which enables it to immediately configure these nodes instead of waiting for the node to fetch the desired configuration description. This push strategy allows Ansible to require only SSH and Python to be installed on the compute nodes rather than a full-blown maintenance client. Such a "lightweight" client model suits our purpose best because we want to configure (cloud) compute nodes only for a single simulation run that is manually executed for a relatively short amount of time instead of providing a service that is supposed to be available continuously.

In the following sections, we introduce in detail the three DSLs that are used for the development of the Sprat Marine Ecosystem Model.

[1] http://www.ansible.com

8.2 The Sprat PDE DSL

The Sprat PDE DSL is embedded into C++ via the piggyback pattern (Mernik et al. 2005), which means that the DSL is implemented completely in C++ and all DSL models are valid C++ code. This offers the advantage of acquiring full tool support (editors, debuggers, compilers, etc.) without any additional implementation effort. The choice of C++ as the host language is mainly due to the wide-spread use of C and C++ among numerical mathematicians (user acceptance). In addition to that, the operator overloading capabilities of C++ allow the DSL to feature matrix-vector expressions with a "natural" syntax.

While the implementation of the Sprat PDE DSL itself uses features that are specific to C++ and that are not present in C, the language is designed in a way that should enable it to be used also by domain experts proficient only in C and not in C++. As the Sprat PDE DSL does not enforce a specific program structure (in contrast to a framework) and can be used alongside existing code, a stepwise adoption of the DSL in existing projects is possible. All the aspects mentioned before are part of an effort to make the Sprat PDE DSL as easy to learn and use as possible and to ensure its acceptance in the target community.

The focus of the Sprat PDE DSL is on the implementation of special-purpose FEM solvers. It addresses developers of such algorithms rather than FEM practitioners, who are not necessarily interested in *how* a certain PDE or system of PDEs is solved. Therefore, the language does not feature the most abstract concepts of the FEM (say, variational forms) but concentrates on entities that allow to conveniently model mesh-based PDE solvers.

From a technical perspective, the language is comprised of a set of header files written in C++11 that can be used by the application programmer via include statements. These headers expose a set of classes, macros, and functions that interact with each other to implement the following four key feature areas, which are illustrated by Listing 8.1:

1. Coherent abstractions for the *mesh topology*. Solver algorithms implemented with the Sprat PDE DSL can be expressed independently of the employed mesh type and of the number of its spatial dimensions. Therefore, the mesh type and its dimension can be varied without having to modify the algorithm itself. This is made possible by abstractions—such

```
1  DistributedVector  u, q;
2  ElementVectorArray F_L;
3  ElementMatrixArray C;
4  ElementMatrix      D;
5
6  foreach_omp(tau, Elements(mesh), private(D), {
7      foreach(i, ElementDoF(tau), {
8          foreach(j, ElementDoF(tau), {
9              D(i, j) = max(i.globalIndex(), j.globalIndex());
10         })
11     })
12     F_L[tau] = C[tau]*q + D*u;
13 })
14 u *= u.dotProduct(q);
15 u.exchangeData();
```

Listing 8.1. Sprat PDE DSL code snippet.

as elements and Degrees of Freedom (DoF)—that are coherent for all mesh types and any number of dimensions and "know" how to handle common mesh-specific tasks. An example of this can be seen in line 6 of Listing 8.1, where—depending on the type of `mesh`—`tau` automatically is chosen to be of a corresponding element type that adjusts its behavior accordingly (e. g., it would "know" how to correctly compute the partially integrated integral of two of its degrees of freedom for assembling a discrete Laplace operator).

2. Lazily-evaluated *matrix-vector arithmetic* with a natural and declarative syntax, parallel execution, and DSOs. Lazy evaluation of matrix-vector expressions means that no temporary variables are created during their evaluation. For example, in line 12 of Listing 8.1, the assignment of the expression on the right-hand side to the element vector `F_L[tau]` is computed by directly adding the individual contributions of `C[tau]*q` and `D*u` to `F_L[tau]`. Additionally, we employ DSOs that fuse the computation of `C[tau]*q` and `D*u` in a single loop, which prevents multiple iterations over matrices with a common structure. And, moreover, the evaluation

of matrix-vector expressions is automatically executed in parallel. For further details on lazy evaluation and DSOs in the Sprat PDE DSL, see Section 8.2.2.

3. (Parallel) *iterations over sets*. One of the most common tasks found in numerical algorithms is to use an integer variable to iterate over some index range. The drawback of this approach is that the iteration variable has no semantic connection with the objects that is iterated over. Because of that, we added iterations over sets, which explicitly state that, e. g., the variable `tau` in line 6 iterates over the set of all elements of the mesh. Moreover, as the iteration variables are thin wrappers around indices, functionality related to the object they represent can directly be requested from them (for example, in line 9, we ask the element DoF `i` for its global index). Iterations over sets can be parallelized using OpenMP clauses (Dagum and Menon 1998) as shown in line 6.

4. Optional *Single Program, Multiple Data (SPMD) abstractions* for parallel computing (Duncan 1990). We use MPI (Message Passing Interface Forum 2012) to distribute computations across different compute nodes. Many data types, such as the `DistributedVector` or the mesh classes, feature high-level abstractions to transparently handle data exchange between compute nodes. For example, in line 15 the distributed vector `u` is instructed to exchange data regarding duplicated ghost DoF in the mesh after being updated in line 14. The method for calculating the dot product of `u` and `q` in line 14 is also aware of the distributed nature of the problem and automatically computes the right value for the *global* problem and not just the node-local answer. Furthermore, meshes can be automatically partitioned and distributed across different compute nodes. Since many computational scientists have already implemented their own parallelization framework (cf. Section 6.2.3 e)), we made using the parallelization features of the Sprat PDE DSL completely optional (via compile-time switches).

   To achieve good data locality, we combine MPI with OpenMP. OpenMP is used to automatically execute vector operations, such as the update of `u` in line 14, in parallel. Additionally, as mentioned before, OpenMP-enabled versions of our set-based iterations exist.

By combining these features, the Sprat PDE DSL allows to express mesh-based PDE algorithms in a concise way that closely resembles their representation in mathematical text books or articles. Because of this high abstraction level, it is relatively easy to apply typical changes to a solver algorithm (such as generalizing it to more dimensions) and to check whether the implemented algorithm actually corresponds to the algorithm in a formal description like one in a paper (since they almost look the same). The compact notation for matrix-vector arithmetic also simplifies writing tests because checking, for example, whether the assertion holds that all entries of the sum of two vectors are positive, can be expressed in just a one-line statement (`assert(u+v > 0)`).

Furthermore, it can be assumed that the language is easy to learn for a numerical mathematician who is already familiar with C or C++. The user has to interact only with a handful of data types and is most certainly already acquainted with the concepts of matrix-vector arithmetic and iterations over sets. Additionally, many FEM algorithms share a similar structure, which makes it possible to supply the user with a skeleton for their implementation. Such a skeleton also encourages users to employ all the features of the DSL rather than implementing existing features again in the host language.

8.2.1 Domain Meta-Model

In order to have a closer look at the features of the Sprat PDE DSL, this section presents an overview on the meta-model of the language. As for the description of the meta-model of DSL hierarchies in Chapter 7, we use a combination of the Unified Modeling Language (UML) and Object-Z for this purpose. Note that for the sake of clarity, our presentation of the PDE DSL meta-model only focuses on key concepts of the language. Therefore, the formal specification omits several details that are irrelevant to a high-level understanding of the DSL. For a complete reference of the language, we refer to the implementation available online (Johanson 2015d).

To make the presentation easy to follow, we divide the meta-model into the four main feature areas presented above. The *SPMD abstractions* for parallel computing, however, are represented only by the single meta-class *Parallel_Execution_Environment*.

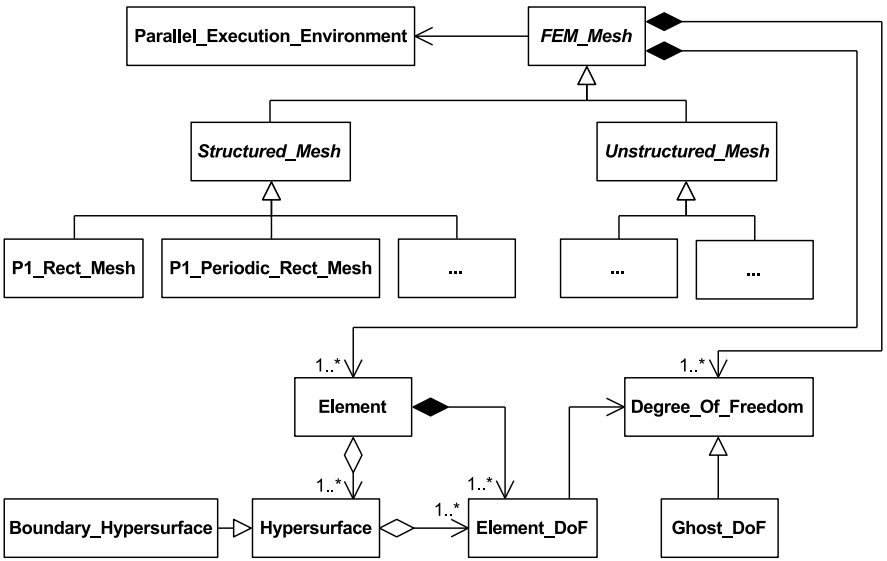


Figure 8.2. Meta-model elements of the Sprat PDE DSL associated with the mesh topology.

Mesh Topology

All meta-model elements associated with the *mesh topology* are depicted in Figure 8.2. As one would expect from the theory of FEM solvers (Chapter 4), a *FEM_Mesh* consists of *Elements* and *Degrees_Of_Freedom*, which are linked together via *Element_DoF*. Elements have *Hypersurfaces*, from which—again—the *Element_DoF* of the element can be accessed. Furthermore, a *FEM_Mesh* is associated with a *Parallel_Execution_Environment*, which transparently handles all SPMD parallelization tasks that are related to the mesh, such as mesh partitioning. Regarding such mesh partitions, ghost DoF are represented by the corresponding specialization of the *Degrees_Of_Freedom* meta-class.

A *FEM_Mesh* can either be structured or unstructured. The Sprat PDE DSL provides several default mesh implementations such as a rectangular mesh with P1 elements. Code skeletons are provided for users of the DSL to develop their own mesh types (as indicated by the meta-classes with

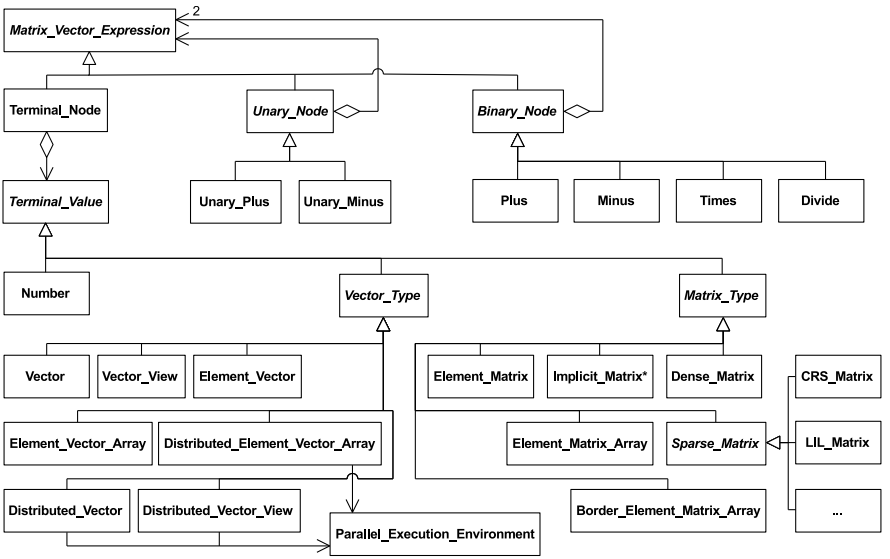


Figure 8.3. Meta-model elements of the Sprat PDE DSL related to matrix-vector expressions.

the dots). All meshes have to be implemented for arbitrarily many spatial dimensions.

Matrix-Vector Expressions

The second feature area are lazily-evaluated *matrix-vector expressions* as illustrated in Figure 8.3. A *Matrix_Vector_Expression* is represented as a tree containing the typical arithmetical operators. A *Terminal_Value* either is a floating-point scalar (*Number*), a *Vector_Type*, or a *Matrix_Type*.

Besides usual dense vectors, the Sprat PDE DSL features views on parts or strides of vectors, element vectors (see Chapter 12), and distributed vector types as well as combinations of those types. Distributed vectors use the *Parallel_Execution_Environment* discussed above to transparently handle data exchange for ghost DoF between different compute nodes.

Among the matrix types supported by the DSL are dense, sparse, and implicit operators as well as element matrices (see Chapter 12). Sparse matrices can be stored in different formats, such as Compressed Row Storage (CRS) or List of Lists (LIL) (Pissanetzky 2014). Implicit matrices represent operators that are not stored explicitly (i. e., one cannot access individual entries) but it is known how to apply them to a vector. The *Implicit_Matrix* meta-class is starred in Figure 8.3 because it was not present in the initial reference implementation but was suggested to be implemented during the evaluation of the Sprat PDE DSL by domain experts (see Chapter 13).

In order to be able to handle matrix-vector expressions efficiently, their evaluation must not require the creation of temporary matrices or vectors. This is not only efficient but also allows the user of the Sprat PDE DSL to stay in full control of memory allocation. To model the constraints necessary to guarantee that no temporaries are required for the evaluation, we introduce all possible categories of *Terminal_Values* as Object-Z classes.

Terminal_Value

$$size : \mathbb{N}$$

A *Null_Terminal* is introduced to model empty sub-trees in Object-Z.

Null_Terminal

Terminal_Value

$$size = 0$$

Number

Terminal_Value

$$size = 1$$

For simplicity, we only consider quadratic matrices with $size \times size$ entries in this specification. The implementation of the Sprat PDE DSL, however, also features general rectangular matrices.

Vector_Type

Terminal_Value

$size > 0$

Matrix_Type

Terminal_Value

$size > 0$

Due to limitations of the Object-Z specification language, we do not model the different types of matrix-vector expression nodes via inheritance as shown in Figure 8.3. Instead, each node is of class *Matrix_Vector_Expression* and its node type is identified by a type attribute. In the Sprat PDE DSL, all binary operators (i. e., nodes that are of type *plus*, *minus*, *times*, or *divide*) with two vector-valued operands indicate the element-wise application of the corresponding operation between those operands.

NODETYPE ::= *terminal* |
unary_plus | *unary_minus* |
plus | *minus* | *times* | *divide* |
null

If the node of the expression tree is a terminal, a corresponding terminal value is associated with it. The *nodes* attribute represents all nodes of the (sub-)expression tree, which are partitioned in a *left* and a *right* sub-tree (each possibly being empty).

Matrix_Vector_Expression

$type : NODETYPE$
$value : \downarrow Terminal\_Value$
$left, right : Matrix\_Vector\_Expression$
$nodes : \mathbb{P}_1\, Matrix\_Vector\_Expression$

$\langle left.nodes, \{self\}, right.nodes\rangle$ partitions $nodes$

Depending on its type, the expression node has left and right children (binary), only a left child (unary), or no children (terminal and null type). The type of the value associated with the node is constrained in accordance.

$type = null \iff value \in Null\_Terminal \land$
$\quad left.type = null \land right.type = null$

$type = terminal \iff value \in Number \cup Vector\_Type \cup Matrix\_Type \land$
$\quad left.type = null \land right.type = null$

$type = unary\_plus \lor type = unary\_minus \iff$
$\quad value \in Null\_Terminal \land left.type \neq null \land right.type = null$

$type = plus \lor type = minus \lor type = times \lor type = divide \iff$
$\quad value \in Null\_Terminal \land left.type \neq null \land right.type \neq null$

The set of all possible matrix-vector expressions can now be constrained to those valid in the Sprat PDE DSL, which are described by the set *Valid_MV_Expressions*. In order not to require any temporary variables for the assignment of an expression to a vector, the expression must contain matrix types only as the left-most operand of a top-level term and this matrix must be multiplied with an expression containing only numbers and vectors. This implies that, in particular, we cannot allow any matrix-valued expressions in the Sprat PDE DSL.

Translating this constraint into the context of our expression trees, it means that any matrix node must be the left operand of a node of type *times*. Furthermore, all parents of this *times* node must *not* be *times* or *divide* nodes (to ensure that we are in a top-level term) and the right operand of the *times* node must itself not contain any matrix types (to ensure that

we do not allow any matrix-valued expressions). Of course, all matrix and vector terminals must be of compatible size.

$$
\begin{array}{l}
\mathit{Valid\_MV\_Expressions} == \{\mathit{expr} : \mathit{Matrix\_Vector\_Expression} \mid \\
\quad (\forall\, n : \mathit{expr.nodes} \mid n.\mathit{value} \in \mathit{Matrix\_Type} \bullet \\
\quad\quad (\exists\, o : \mathit{expr.nodes} \mid o.\mathit{type} = \mathit{times} \bullet o.\mathit{left} = n \land \\
\quad\quad\quad (\forall\, p : o.\mathit{right.nodes} \bullet p.\mathit{value} \notin \mathit{Matrix\_Type}) \land \\
\quad\quad\quad (\exists\, p : o.\mathit{right.nodes} \bullet p.\mathit{value} \in \mathit{Vector\_Type})) \land \\
\quad\quad (\forall\, o : \mathit{expr.nodes} \mid o \neq n \land n \in o.\mathit{nodes} \bullet \\
\quad\quad\quad o.\mathit{type} \neq \mathit{times} \land o.\mathit{type} \neq \mathit{divide})) \land \\
\quad (\exists\, i : \mathbb{N} \bullet \forall\, n : \mathit{expr.nodes} \bullet \\
\quad\quad (n.\mathit{value} \in \mathit{Vector\_Type} \lor n.\mathit{value} \in \mathit{Matrix\_Type}) \implies \\
\quad\quad\quad n.\mathit{value.size} = i)\}
\end{array}
$$

If matrix-vector expressions appear in the context of comparisons (with another vector or a floating-point value), even stricter constraints apply that are characterized by the definition of *Valid_Comparison_MV_Expressions*: no matrix terminals must appear at all.

$$
\begin{array}{l}
\mathit{Valid\_Comparison\_MV\_Expressions} == \{\mathit{expr} : \mathit{Matrix\_Vector\_Expression} \mid \\
\quad (\forall\, n : \mathit{expr.nodes} \bullet n.\mathit{value} \notin \mathit{Matrix\_Type}) \land \\
\quad (\exists\, i : \mathbb{N} \bullet \forall\, n : \mathit{expr.nodes} \mid n.\mathit{value} \in \mathit{Vector\_Type} \bullet n.\mathit{value.size} = i)\}
\end{array}
$$

Iteration Over Sets

In PDE solvers, one often needs to iterate over all instances of a particular aspect of the mesh topology, such as over all elements or DoF. Therefore, *Iterations* in the Sprat PDE DSL feature an *Iteration_Variable* that successively assumes the value of every *Set_Element* in a given *Set* as depicted in Figure 8.4. The set elements can be traditional integer-valued indices but can also be any entity of the mesh topology. The loop body of the iteration can contain arbitrary C/C++ statements, which in turn might consist of matrix-vector expressions.

An iteration itself can either be executed serially or in parallel. Parallel iterations rely on the OpenMP technology (Dagum and Menon 1998) and, therefore, might require additional *OpenMP_Clauses* to control data sharing between threads. A special case of the parallel iteration is the *Parallel_*

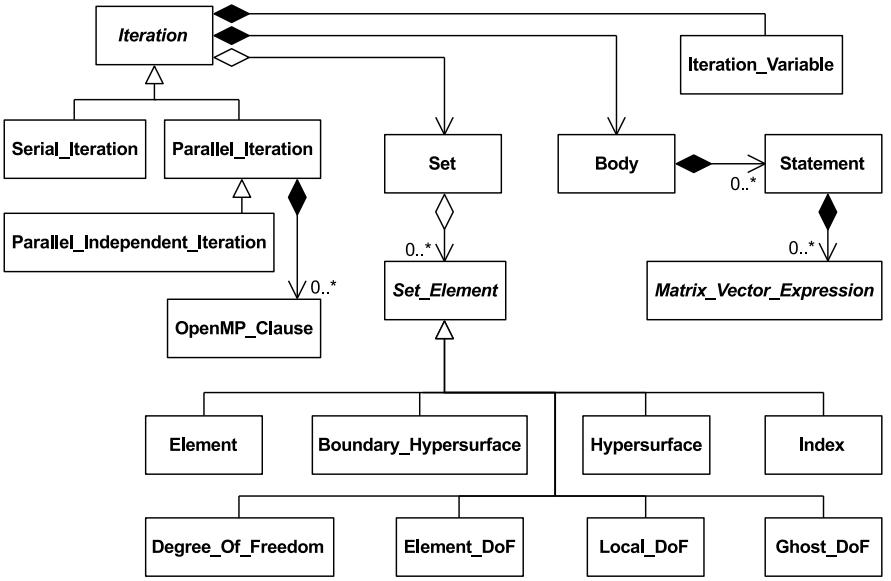


Figure 8.4. Meta-model elements of the Sprat PDE DSL associated with iterations over sets.

Independent_Iteration, which is only available for sets of mesh elements. It guarantees that the elements are iterated over in an order that allows to modify values associated with the DoF of each element in parallel without the need for synchronization between threads. To accomplish this, the elements are divided into subsets forming *maximal independent sets* (Robson 1986) for their respective DoF.

8.2.2 DSL Implementation

In order for the Sprat PDE DSL to be accepted by the HPC community, the abstractions provided by the language must not compromise the runtime performance of programs significantly. This goal has been achieved (cf. Chapter 13) by observing three key principles:

1. Avoid using inheritance relationships that would require type introspection. Although we extensively employed inheritance to express the meta-model of the DSL, inheritance is generally avoided in the implementation to circumvent the computational cost of looking up type information at runtime.
2. Ensure that the compiler can apply code inlining as often as possible. For example, when using abstractions for iterating over sets with the iteration variable being an object (e. g., a DoF) that can also be used as an index (e. g., the index of a DoF), the index look-up must not result in a function call. Utilizing function calls in this context would slow down the execution considerably because index look-ups tend to occur very often (e. g., in the body of a loop over all DoF that is executed for every time step of the algorithm). To allow inlining as often as possible, the Sprat PDE DSL is implemented using only header files.
3. Make sure that matrix-vector expressions are evaluated lazily instead of eagerly and apply DSOs to such expressions. We achieve this by using template meta-programming techniques (Abrahams and Gurtovoy 2004) and exploit optimization potential arising, e. g., from sparse matrices with the same sparsity patterns.

By default, expressions in C++ are evaluated eagerly, which means, for example, that for vectors `u`, `v`, and `w` the assignment `u = u + v * w` would be computed by creating a temporary vector `t1 = v * w`, then another temporary vector `t2 = u + t1`, which is finally copied over to `u = t2`. This results in unnecessary temporary variables and unnecessarily many iterations over the index range of the vectors. Instead, we would like the expression `u + v * w` from above to be evaluated lazily only when its result is actually needed (when it is assigned to `u`) and the whole assignment statement should be computed in a single loop equivalent to:

```
for(int i=0; i<u.size(); i++) {
    u[i] = u[i] + v[i] * w[i];
}
```

To achieve this, an AST representation of the right-hand side `u + v * w` would be needed. Since the AST of the compiler is not available to us in C++, we have to reconstruct it as a template type using template meta-programming. For example, a binary operator node can be represented by

a type that is templated with its two child nodes (for details, see Abrahams and Gurtovoy 2004). Since the implementation of this is tedious and error-prone (esp. regarding the type system and transformations of the AST), we use Boost Proto (Niebler 2007), which is itself an embedded DSL for embedding DSLs into C++. Boost Proto provides means for constructing, transforming, and executing template expressions in the form of an AST. It allows specifying a grammar for a DSL and automatically takes care of the necessary operator overloading.

Embedding a DSL into C++ this way offers the great advantage of getting full language support without any additional effort. There are, however, two drawbacks to this approach. First, the generation step from the DSL to the target language is implicit and, thus, there is no generated code that could be inspected. This can partly be overcome by looking at the output of different compiler stages, although we recognize that this can hardly compete with well-formatted code from an explicit generation step. A second drawback is concerned with error reporting. It is well known that many C++ compilers generate long and complicated error messages when it comes to errors concerning template types. But even if this was overcome, the error reporting would still not be on the level of abstraction on which the users write their code in the DSL (i. e., matrices and vectors and not template data types). To mitigate this problem, the Sprat PDE DSL implementation makes extensive use of static assertions, which are checked at compile time and produce meaningful error messages on the level of abstraction of the model.

Concerning the maintenance of the Sprat PDE DSL, users would most likely want to add new data types, such as special-purpose matrix formats or mesh types. For a language that is as closely embedded into its host language as the PDE DSL, adding new matrix or vector types will likely prove to be difficult: while Boost Proto simplifies the process of introducing new types, it is still far from trivial to correctly implement all their interactions with other data types. However, we do not consider this a serious concern since it is very unlikely that the provided set of matrix types is inadequate. Apart from that, new mesh types can be introduced by simply filling in the gaps of a class skeleton.

For a quantitative analysis of the performance of the Sprat PDE DSL as well as a qualitative evaluation of the whole language, see Chapter 13.

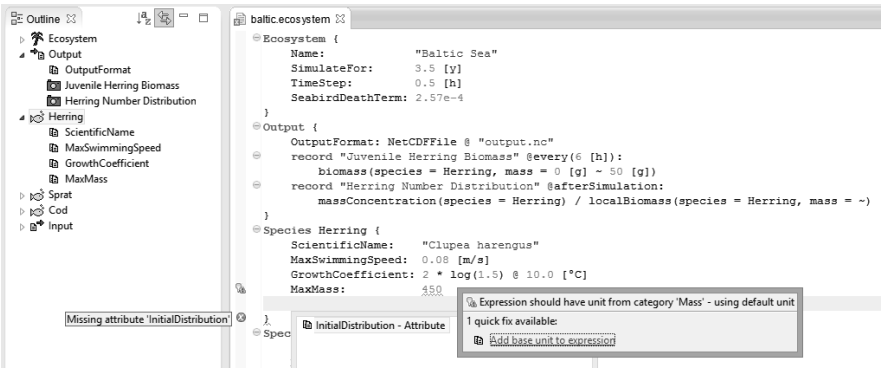


Figure 8.5. Editor of the Sprat Ecosystem DSL featuring syntax coloring, high-level error messages, autocomplete, and quick fixes.

8.3 The Sprat Ecosystem DSL

The Sprat Ecosystem DSL allows to specify ecosystem simulations in a declarative way as shown in Figure 8.5. A simulation description 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.

Unit support is vital for such a description language as there are numerous popular examples of mission-critical failures resulting from unit inconsistencies in numerical software (e. g., the Mars Climate Orbiter crash due to the mixed use of non-/metric units; Knight 2002). If a unit is missing, the editor issues a warning and offers a quick fix that adds a unit of the correct quantity category to the expression (which would be kilograms in the case depicted in Figure 8.5). Unit conversions (e. g., from degree Fahrenheit to degree Celsius) are automatically carried out by the DSL.

As some properties might be specified in relation to another quantity (e. g., a growth coefficient is specified for a certain temperature), a modifier can be introduced to these properties with the `@` keyword:

```
GrowthCoefficient: 0.1 @ 10 [°C]
```

Another keyword of the language is `record`. It can be used in the output entity to let the user describe which data should be collected during a simu-

lation run via record expressions. This allows to aggregate the information already while the simulation is running and, thus, makes it unnecessary to store *all* the data generated by the simulation (which typically is a huge amount). Within record expressions, there are special functions to refer to model data. These functions are called using named parameters for better readability and the arguments can be intervals (`from ~ to`) with optional endpoints.

As can be seen from the example in Figure 8.5, the structure of the DSL is straightforward and only contains concepts that the target domain experts should be familiar with. Nonetheless, it is critical for the acceptance of any DSL to guide users while they construct models in the language. To this end, the editor offers a list of content proposals at any given position in the document. Not only do these context-sensitive suggestions include isolated items, such as keywords, functions, and units, but also complete templates for, say, a new species entity and all its necessary properties. An example of this feature is displayed at the bottom of Figure 8.5, where the name of the last missing species property (and only this missing one) is proposed. As long as not all necessary properties are specified, meaningful error messages are raised in appropriate locations.

Beyond model completeness, the validator of the DSL checks various other constraints to ensure that the description of the simulation is sound and will result in a successful simulation run. Especially on the higher levels of the DSL hierarchy of the Sprat Approach, it is important to make sure that all errors are detected before generating source code for the next lower level, since, during this process, abstraction will be lost. Thus, on the lower layer, it would no longer be possible to communicate problems to the domain experts on the level of abstraction that they are familiar with and that they implemented their model in.

With the high-level abstractions presented above and its seamless integration with the lower levels of our DSL hierarchy, the Sprat Ecosystem DSL facilitates the collaboration between experts from different disciplines. Its concise and declarative syntax ensures that the resulting models are descriptive and maintainable. The intuitive language design and the full tool support minimize hurdles in adopting the DSL (see Chapter 14 for the empirical evaluation of the language).

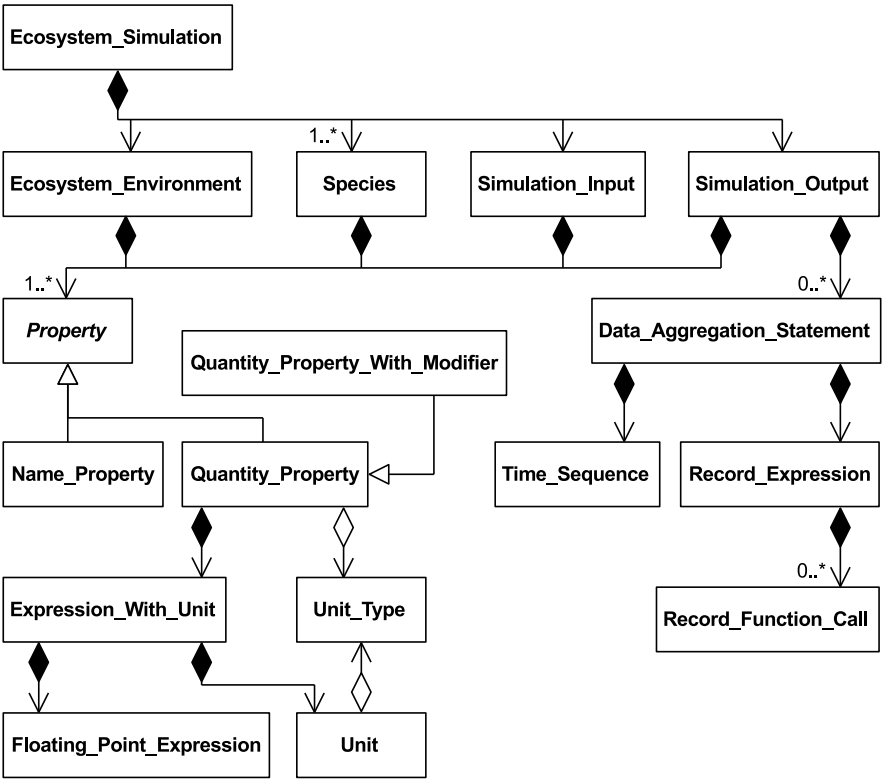


Figure 8.6. Meta-model of the Sprat Ecosystem DSL.

8.3.1 Domain Meta-Model

In this section, we give an overview on the meta-model of the Sprat Ecosystem DSL using a combination of the UML and Object-Z, again. As with the Sprat PDE DSL, we present Object-Z specifications only for the most important meta-classes. A complete Object-Z specification of the Sprat Ecosystem DSL can be found in Appendix B.

The characterization of an *Ecosystem_Simulation* consists of the description of the *Ecosystem_Environment*, the *Species* present, as well as the *Simula-*

tion_Input and *Simulation_Output* configurations as depicted in Figure 8.6. All these four meta-classes contain certain *Properties* which are key-value pairs. Properties can be categorized depending on their value type. Their value can be a name (e. g., a species name; *Name_Property*), a floating-point value with a unit (*Quantity_Property*), or a pair of two floating-point values with their own units (*Quantity_Property_With_Modifier*). Units are of a certain *Unit_Type* and feature a conversion factor to the base unit of that unit type.

Unit

name : *NAME*
type : *UNIT_TYPE*
conversionFactor : *FP_NUMBER*

Depending on the concrete simulation that is targeted by the Sprat Ecosystem DSL, different properties have to be present. These are described by *Property_Description*, which is *not* an element of the abstract syntax of the DSL and, hence, does not appear in Figure 8.6. Such descriptions define the name of a property and—depending on the concrete type of the property—the unit types associated with its values.

Property_Description

key : *NAME*
unitType : *UNIT_TYPE*
unitTypeModifier : *UNIT_TYPE*

There are three sub-types of *Property_Description* corresponding to the three value types that concrete properties can have (*Name_Property_Description*, *Quantity_Property_Description*, and *Quantity_Property_With_Modifier_Description*). We introduce sets of properties for each of the four main areas of an ecosystem simulation description, e. g., for the ecosystem environment:

$$EnvironmentProperties : \mathbb{F}_1 \downarrow Property\_Description$$

$$\forall a, b : EnvironmentProperties \bullet a.key = b.key \implies a = b$$

With these sets, we can constrain the properties that are actually present in the model.

Ecosystem_Environment

$$properties : \mathbb{F}_1 \downarrow Property©$$

The *properties* are exactly those described by *EnvironmentProperties*.

$$\#properties = \#EnvironmentProperties$$

$$\forall p : properties \bullet \exists q : EnvironmentProperties \bullet$$
$$p.key = q.key \wedge$$
$$(p \in Name\_Property \implies q \in Name\_Property\_Description) \wedge$$
$$(p \in Quantity\_Property \implies q \in Quantity\_Property\_Description \wedge$$
$$p.unitType = q.unitType) \wedge$$
$$(p \in Quantity\_Property\_With\_Modifier \implies$$
$$q \in Quantity\_Property\_With\_Modifier\_Description \wedge$$
$$p.unitType = q.unitType \wedge p.unitTypeModifier = q.unitTypeModifier)$$

The *Simulation_Output* meta-class does not only consist of properties but also of *Data_Aggregation_Statements*, which describe when to record which data (given as a record expression) under a certain name.

Data_Aggregation_Statement

$$name : NAME$$
$$when : TIME\_SEQUENCE$$
$$what : RECORD\_EXPR$$

Record expressions are like floating-point expressions but can contain record functions as an additional terminal (e. g., the total biomass of a fish species at a certain point in time; for examples, see Figure 8.5 on page 121).

Just as with properties, the available record functions depend on the target simulation and are similarly specified using a *Record_Function_Description* class (which, again, is not part of the abstract syntax of the DSL itself but rather configures the DSL).

$$validRecordFunctions : \mathbb{F}_1\ Record\_Function\_Description$$

With the set of valid record functions, we are able to constrain the *Record_Function_Calls* that can appear in a model. The arguments of a record function can be of different types: a name (e. g., for specifying a species; *RF_Name_Argument*), a constant floating-point expression (*RF_Expression_Argument*), or a range expression (e. g., specifying an interval of fish sizes; *RF_Range_Expression_Argument*).

Record_Function_Call

$$name : NAME$$
$$args : \text{seq}\downarrow Record\_Function\_Argument©$$

Record function calls must match one of the record function descriptions given in *validRecordFunctions*.

$$\exists f : validRecordFunctions \bullet name = f.name \land$$
$$\#args = \#f.args \land \forall a : f.args \bullet$$
$$((args(first(a))).name = (second(a)).name \land$$
$$(second(a) \in RF\_Expression\_Argument\_Description \implies$$
$$args(first(a)) \in RF\_Expression\_Argument) \land$$
$$(second(a) \in RF\_Range\_Expression\_Argument\_Description \implies$$
$$args(first(a)) \in RF\_Range\_Expression\_Argument) \land$$
$$(second(a) \in RF\_Name\_Argument\_Description \implies$$
$$args(first(a)) \in RF\_Name\_Argument))$$

8.3.2 DSL Implementation

Our reference implementation of the Sprat Ecosystem DSL is built using Xtext (cf. Chapter 2). The Java-based framework is structured in a way that all functional components of the DSL runtime reside in their own module

```
MASS.add(new SpratUnit("kg", 1.0   ));
MASS.add(new SpratUnit("g" , 1.0e-3));
SPECIES_ATTRIBUTES.add(new SpratAttribute("MaxMass", MASS));
```

Listing 8.2. Java snippet for configuring the Sprat Ecosystem DSL runtime.

and are composed using dependency injection. This makes it possible to overwrite and customize nearly all aspects of the language runtime in a modular way. For our purpose, this is especially interesting in the context of the code generator as it allows us to easily switch between different generator implementations at runtime in order to target different fish stock models with exactly the same simulation description.

The generator module that we implemented for our specific DSL hierarchy produces C++ code. We took care to separate the generated code from user-written code because generated files are just overwritten without any warning by our generator. In C++ and some other object-oriented target languages, one option to achieve this separation is the generation gap pattern, which works with inheritance (Fowler 2010). Even though the generated code is not meant to be consulted by the developers on the lower hierarchy level, we made sure that it is well-formatted and tried to preserve some of the abstractions of the Ecosystem DSL (e. g., by stating the unit of every property as a comment in the generated code). This way, we maximize clarity and help to prevent problems with different interpretations that might appear at the transition of two DSL hierarchy levels due to model transformations.

While Xtext encourages the creation of a DSL runtime infrastructure comprised of loosely coupled modules, we introduced a single central configuration class used by all these modules. In this configuration class, we describe all the properties, units, record functions, etc. that can be used within the language. The Java code (Listing 8.2) is as declarative as possible and focuses on readability for non-programmers. Therefore, one could say that the code uses another embedded DSL to configure the external Sprat Ecosystem DSL. Such a design which uses a central configuration class with a DSL-like syntax, makes it possible to introduce new units or properties in all modules of the DSL implementation—such as the generator, the content

```
---
- hosts: localhost
connection: local
vars_files:
- ./openstack_config.yml

tasks:
- name: Make sure the cloud instances are running
nova_compute:
state:     present
hostname:  sprat{{ item }}
image_id:  "{{ os_image_id }}"
flavor_id: "{{ os_flavor_id }}"
with_sequence: start=1 end={{ nInstances }}
```

Listing 8.3. Excerpt from the Ansible Playbook for deploying the Sprat Simulation on an OpenStack cloud.

assist, etc.—by merely adding a single line of code to one file. This allows the users of the Ecosystem DSL (who are likely not trained as software developers) to carry out basic maintenance tasks related to the language by themselves by simply copying, pasting, and customizing intuitively readable code fragments. For an empirical analysis of this hypothesis and the evaluation of the Sprat Ecosystem DSL as a whole, see Chapter 14.

8.4 The Ansible Playbook DSL

The Ansible Playbook DSL is reused to describe configuration states that certain systems are supposed to be in. In doing so, the user neither has to specify the initial state of the system nor the transformations that have to be applied to achieve the desired state. The syntax of the DSL—an example of which is given in Listing 8.3—is based on YAML.[2]

[2] http://www.yaml.org/spec/

A *playbook*—as Ansible models are called—begins with the specification of the target *host group* (in case of the example, it consists only of the machine on which Ansible is executed). A host group is a label for a list of hostnames or IP addresses that all are to be configured in a common way (e. g., the compute workers of a simulation run). The host groups are specified in so-called *inventories*, which are text files given to Ansible as command-line parameters. Next, a list of variable files (`vars_files`) can be specified that may contain information like login details for a cloud provider etc. This header information is followed by the key element of each playbook: a list of *tasks* that each use an Ansible *module* to describe an aspect of the state that the target hosts are supposed to be in. The only task in the example from Listing 8.3 uses the `nova_compute` module to ensure that a number of cloud instances with specific properties are present. Playbooks can also include tasks from other playbooks, which enables a modular structure of deployment descriptions.

Once a playbook is executed, the Ansible execution engine (running on the machine on which the playbook is stored) connects to the target hosts via Secure Shell (SSH), gathers information about their current state, and takes appropriate actions to transform these states into the desired configuration specified by the playbook.

8.4.1 Deploying the Sprat Marine Ecosystem Model

In the deployment process of the Sprat Marine Ecosystem Model, the modular structure of Ansible playbooks allows us to separate the phase of configuring the compute environment from the phase of deploying and running the simulation (see Figure 8.7). In this way, we can independently implement different back-ends for configuring compute nodes in various computing environments from bare-metal clusters to different private or public cloud computing providers. Concretely, we implemented such a back-end for a private cloud based on OpenStack.[3]

The back-end makes sure that a user-configurable amount of compute nodes with a suitable environment is spawned and then uploads the simulation data as well as a second "payload" playbook to the master node. This second playbook takes care of configuring the compute nodes from within

[3] `http://www.openstack.org`

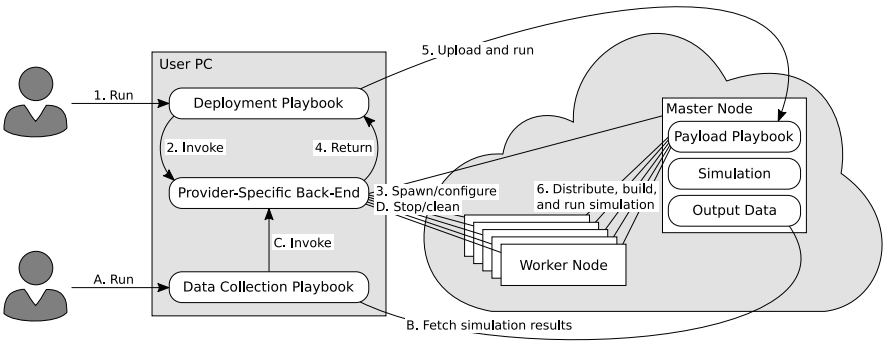


Figure 8.7. Architectural design of the deployment process for the Sprat Marine Ecosystem Model.

the cloud (distribute, build, and run the simulation) and, thus, only has to be aware of the addresses of the other nodes but, apart from that, can be completely agnostic of the concrete infrastructure it is deployed to.

Once a simulation run is finished, the scientists can run another playbook that handles the collection of simulation output. It fetches the results from the master node and invokes the same provider-specific back-end to stop or to clean the compute nodes.

A more detailed account of the sequence of events during the deployment and the data gathering process is given in the sequence diagram in Figure 8.8 on the following page.

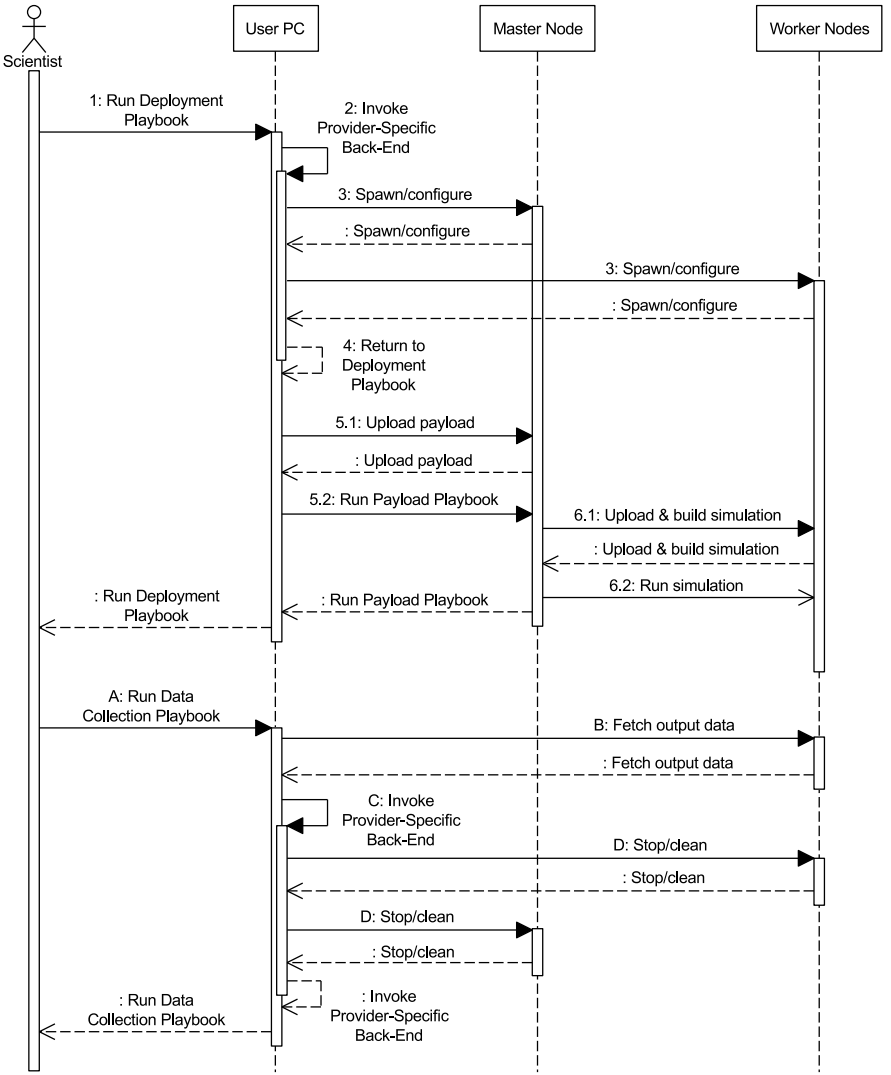


Figure 8.8. Sequence diagram for the deployment and the data gathering process.

Part III

The Sprat Marine Ecosystem Model

Chapter 9

Ecological Modeling Research Design

In this chapter, we present the research design that we employ to meet the challenges associated with the development of a fish stock model for marine end-to-end modeling (cf. Chapter 1). For the research design regarding software engineering approaches for computational science, refer to Chapter 5.

Section 9.1 states our main research goal for the ecological modeling part of this thesis and infers research questions to characterize this goal. In Section 9.2, we describe the research plan to accomplish the aforementioned goal and give an overview on the work packages of the plan as well as on the methods we employ to address the research questions.

9.1 Goal and Research Questions

According to our discussion of the challenges related to the development of marine end-to-end models in Section 1.1.1, we can formulate the research goal for the ecological modeling part of this thesis as:

> Develop a fish stock model for marine end-to-end modeling that integrates well with existing biogeochemical ocean models, that efficiently represents large numbers of fish, and that can be parametrized with existing observational data.

This goal is further explicated by the following set of Ecological Modeling Research Questions (EMRQs).

- **EMRQ1:** How can fish be integrated with (existing) biogeochemical ocean models from a conceptional and technical perspective?

- **EMRQ2:** Which interactions of fish with each other, with other components of the ecosystem, and with the environment have to be modeled and how can these interactions be represented in the context of population balance equations?
- **EMRQ3:** Which state-of-the-art PDE solver is suitable for approximating the solution of the equation system employed in the Sprat Marine Ecosystem Model?
- **EMRQ4:** Which fish population dynamics within an ecosystem can be reproduced by the Sprat Marine Ecosystem Model?
  - **EMRQ4.1:** How can the Sprat Model be parametrized and how sensitive is the model to small changes of the parameter values?
  - **EMRQ4.2:** How important is the explicit representation of space in the Sprat Model for its predictive capabilities?
  - **EMRQ4.3:** How can the Sprat Model be employed in the assessment of different fisheries management strategies?
  - **EMRQ4.4:** How can the Sprat Model be coupled with existing biogeochemical ocean models?

9.2 Research Plan

Based on our research goal and the corresponding research questions presented in Section 9.1, we structure the ecological modeling research conducted in this thesis into the following four Ecological Modeling Work Packages (EMWPs):

- **EMWP1:** Introducing Fish Into Biogeochemical Ocean Models
- **EMWP2:** Designing a Fish Model Based on Population Balances
- **EMWP3:** An Explicit Flux-Corrected Transport Solver
- **EMWP4:** Evaluation

For each work package, we give a short description of the research conducted as part of this thesis. In particular, we highlight which research questions are answered using which methods.

9.2.1 EMWP1: Introducing Fish Into Biogeochemical Ocean Models

Our first work package consists of selecting a suitable modeling approach for integrating fish into biogeochemical ocean models to establish an end-to-end model. For this purpose, we survey existing modeling approaches and identify their main advantages and drawbacks regarding end-to-end modeling from a theoretical perspective. To combine the advantages of these existing approaches while avoiding their main drawbacks, we introduce a novel modeling approach based on population balance equations.

The results of this work package, which can be found in Chapter 10, answer the research question **EMRQ1** (*How can fish be integrated with (existing) biogeochemical ocean models from a conceptional and technical perspective?*).

9.2.2 EMWP2: Designing a Fish Model Based on Population Balances

The second work package covers the design of the Sprat Marine Ecosystem Model, which is a spatially-explicit fish stock model for end-to-end modeling based on population balance equations. Studying other end-to-end models in **SEWP1** enables us to identify relevant biological processes that need to be resolved in the Sprat Model. The main research of this work package focuses on how to formalize these processes in the context of the PDE-based population balance approach.

By presenting a formal mathematical specification of the Sprat Model in Chapter 11, this work package addresses the research question **EMRQ2** (*Which interactions of fish with each other, with other components of the ecosystem, and with the environment have to be modeled and how can these interactions be represented in the context of population balance equations?*).

9.2.3 EMWP3: An Explicit Flux-Corrected Transport Solver

Our third work package is concerned with identifying a suitable state-of-the-art PDE solver to approximate the solution of the Sprat Model. Since we want to take advantage of the benefits of modern FEM solvers (irregular meshes, good convergence even with low regularity of the solution), in Chapter 12, we develop a Flux-Corrected Transport (FCT) FEM solver that uses explicit

multi-step methods to integrate the solution in time. FCT methods allow to obtain high-order accuracy without introducing spurious oscillations into the solution of advection-dominant problems like the main equation system of the Sprat Model. We extend existing FCT methods to enable them to employ explicit multi-step methods for time integration. Explicit time integration is important for our purposes because the non-linear fluxes in the equation system of the Sprat Model are quite costly to evaluate (and with an implicit solver, they would have to be evaluated multiple times per time step, compared to only once per time step with an explicit solver). We present mathematical proofs and numerical experiments to demonstrate that our solver indeed produces high-order accurate approximations which are free from spurious oscillations.

The results of this work package address the research question **EMRQ3** (*Which state-of-the-art PDE solver is suitable for approximating the solution of the equation system employed in the Sprat Marine Ecosystem Model?*).

9.2.4 EMWP4: Evaluation

The fourth work package covers the evaluation of the Sprat Marine Ecosystem Model. To assess our model, we apply it to the eastern Scotian Shelf ecosystem, in which a regime shift from the dominance of benthic predatory fish to the dominance of planktivorous forage fish species was observed in the 1990s. In Chapter 15, we describe how to parametrize the Sprat Model to fit the observed fish stock dynamics and conduct a sensitivity analysis of the model. Our parametrization of the Sprat Model allows us to study the drivers of the regime shift on the eastern Scotian Shelf as well as to quantitatively evaluate the effectiveness of management strategies that could have been used to try to prevent the regime shift. In a second step, in Chapter 16, we explain how to couple the Sprat Model with an existing biogeochemical ocean model for the Scotian Shelf. This enables us to examine the influence of different environmental forcing factors on the spatial distribution of fish in our model. Moreover, it allows us to analyze the impact of resolving space at all on the dynamics of the simulated fish stocks.

The results of our evaluation answer the research question **EMRQ4** (*Which fish population dynamics within an ecosystem can be reproduced by the Sprat Marine Ecosystem Model?*) with its sub-questions **EMRQ4.1** to **4.4**.

Chapter 10

Introducing Fish Into Biogeochemical Ocean Models

This chapter presents a novel approach to introducing fish into biogeochemical ocean models that is based on population balance equations. Before outlining our approach, in Section 10.1, we examine existing modeling frameworks for the fish component of end-to-end models. In Section 10.2, we describe population balance equations in general and in particular how they can be applied in end-to-end modeling. Finally, in Section 10.3, we discuss and compare the different modeling frameworks and highlight the methodological advantages that our novel population balance approach has over the currently used ones.

10.1 Existing Approaches to End-To-End Modeling

This section introduces the three modeling frameworks that have been employed for the fish component of end-to-end models so far:

1. The approach of spatial replication of aggregated stock models
2. Individual-Based Models (IBMs)
3. Advection-Diffusion-Reaction (ADR) models

The following sub-sections describe the fundamental concepts of each of these modeling frameworks as well as a concrete end-to-end model that uses the respective framework.

10.1.1 Spatial Replication of Aggregated Stock Models

We use the term *spatial replication of aggregated stock models* to refer to a modeling approach that extends traditional aggregated stock models in order to make them spatially explicit. Those traditional fish models represent a whole stock only via a single aggregated variable, such as the total number of individuals or the total biomass of a stock, and, thereby, ignore its spatial structure (see Chapter 3). The spatial replication approach reintroduces space into these aggregated models by dividing the spatial domain to be represented by the model into a grid of boxes and by assigning an instance of the aggregated model to each of these boxes. This means that each spatial compartment is associated with an individual set of state variables governed by the same aggregated stock model.

To attain an end-to-end model, each spatial box is furthermore equipped with a Nutrient, Phytoplankton, Zooplankton (NPZ) model (Franks 2002) that represents the lower trophic levels of the food web (the NPZ model can also be replaced by input from an existing biogeochemical ocean model). Each NPZ model is coupled with its corresponding fish stock model via feeding and excretion interactions. In addition, rules for the migration of fish between neighboring spatial boxes have to be specified.

An early example of this type of model is that by Hilborn and Walters (1987), which was used to investigate spatial interactions of fish stocks and fishing fleets. This model, however, is not an end-to-end model as it does not incorporate the NPZ compartments of the food web.

A more recent example of the application of the spatial replication approach is the end-to-end model for the Baltic Sea by Fennel (2008). In this model, a horizontal grid of traditional size-structured fish models with a prescribed food web structure is coupled with a corresponding array of NPZ models in a mass-conserving way. The biomass in each discrete fish size class changes because of aging and because of predation processes (fish either prey on younger fish from a certain species or on zooplankton). Fish movement between adjacent spatial cells is governed by the fish distributing themselves proportionally to the amount of available food in those cells. The model has been employed to study the effects of fishing on fish stock structure (Fennel 2010) and the influence of fish on the distribution of matter in the sea (Radtke et al. 2013).

10.1.2 Individual-Based Models

Individual-Based Models (IBMs) capture the dynamics of populations and ecosystems by describing the interactions of the individuals that form these systems. In an IBM, each one of these individuals (e. g., a single fish) is represented by an attribute vector that characterizes the properties of this individual, such as its location, its size, its hunger state, etc. In discrete time steps, the model is evolved by letting the individuals interact with each other and the environment according to a set of rules. For example, individuals might move in space or ingest other individuals based on the hunger state of the predator and the relative size of predator and prey. It has been shown that complex patterns on the system level can be modeled by using just a few simple rules on the level of the individual (Huston et al. 1988). For a more in-depth discussion of the fundamentals of IBMs, we refer to the works of DeAngelis and Gross (1992), Grimm and Railsback (2005), and Railsback and Grimm (2012).

When employing IBMs for modeling fish stocks, a practical problem arises: as the number of individual fish in a stock is typically very large, it is not computationally feasible to represent every individual of this stock. Two approaches have been proposed to tackle this problem (cf. Hellweger and Bucci 2009). The first is the *representative space* approach, in which only a relatively small reference area instead of the whole region of interest is simulated. This decreases the number of individuals to be taken into account but the approach is only applicable if space is homogeneous at larger scales. Since, in the ocean, there are considerable spatial fluctuations of environmental parameters (temperature, currents, nutrients, etc.), the representative space approach is not appropriate for marine end-to-end modeling.

An alternative to using a reference area in order to reduce the individual count in IBMs is the *super individual* approach (Scheffer et al. 1995). In this approach, individuals that share similar characteristics are replaced by a so-called super individual—i. e., an individual that has parameters similar to those of the individuals it represents plus an additional parameter that describes the number of individuals it stands for. While the super individual approach can be applied in spatially heterogeneous environments, it reduces the accuracy with which inter-individual interactions are resolved (if the whole population is abstracted as a single super individual, no interaction

occurs at all). In essence, an IBM employing the super individual approach can be viewed as an approximation to an "original" IBM that contains all individuals which actually should be modeled.

Since, at ocean scale, the individual fish is neglectable, the super individual approach has become the method of choice for individual-based fish stock models that are used in the context of end-to-end modeling. A popular example of such an IBM that can be coupled with many established biogeochemical ocean models is OSMOSE (Shin and Cury 2001). The model has, for example, been employed to investigate effects of overexploitation on the structure of marine food webs (Shin et al. 2004). OSMOSE is a two-dimensional multi-species model that assumes size-based opportunistic predation (i. e., fish forage regardless of the species of their prey). In the model, space is divided into grid cells and time advances in relatively large steps (one year is the default time step, with a month being the shortest possible time interval). In every time step, super individuals can emigrate to adjacent cells depending on the levels of available prey in these neighboring cells compared to the levels in the current cell. Fish can forage on all other fish that are in the same cell and that are a certain number of times smaller than they are. With OSMOSE, the trophic structure of an ecosystem is an emergent property of the model and does not have to be prescribed as is the case with the spatial replication model by Fennel (2008), which we discussed above.

10.1.3 Advection-Diffusion-Reaction Models

In Advection-Diffusion-Reaction (ADR) models, fish are described via a population density distribution P that depends on both time and space. Such a distribution represents either the average number or the average biomass of fish present at a specific location, which means that for every subset $S \subseteq \Omega$ of the spatial domain Ω, the value of

$$\int_S P(t,x)\, dx \tag{10.1}$$

is the number/the biomass of the fish contained in S at the point in time t. The evolution of P is governed by a so-called ADR equation, which—in one

spatial dimension—is given by

$$\frac{\partial P}{\partial t} + \frac{\partial}{\partial x}(vP) - \frac{\partial}{\partial x}\left(D\frac{\partial P}{\partial x}\right) = H. \tag{10.2}$$

This Partial Differential Equation (PDE) characterizes the movement of fish in space as consisting of a random (diffusive) and a directed (advective) component. The magnitude of the random component is set by the diffusion coefficient $D \geqslant 0$ and the direction and speed of the directed component by the advection velocity v. The generation and decay of fish is modeled via the reaction term H (also called *source term*) that allows to incorporate effects such as recruitment and fishing into an ADR model. Furthermore, the reaction term can be used to exchange mass between an ADR fish model and a plankton model, thereby, making ADR models suitable for end-to-end modeling purposes. Each D, v, and H may depend on time, space, and also on the population distribution P itself. Note that if there is no import or export of fish on the spatial boundaries and if the total net reaction rate is zero, which means

$$\int_\Omega H\, dx \equiv 0, \tag{10.3}$$

the ADR model given by Equation 10.2 conserves the total number/total biomass of individuals in the modeled system.

If more than one species or different age classes of the same species are to be considered by an ADR model, they each have to be represented by an individual density distribution P_i. In this case, the exchange of individuals/biomass between those different density distributions (e. g., due to predation or aging) can be modeled via the reaction term H.

An example of an ADR fish stock model that can be coupled with existing biogeochemical ocean models is SEAPODYM by Bertignac et al. (1998). The model aims at studying a single top-predator fish species and was recently used to explore the effects of climate change on tuna stocks and fisheries (Lehodey et al. 2013). In the model, the age structure of the simulated species is represented via discrete age classes that together cover the whole life cycle of an individual. For example, Lehodey et al. (2013) divide the tuna population they investigate into a total of 18 age classes consisting of a larval stage, a juvenile stage, four classes for young fish, and twelve classes for mature individuals.

The SEAPODYM model utilizes a habitat index function $I(t,x) \geqslant 0$ to characterize fish movement. The more suitable a location x is for the fish (e. g., because there is much food), the greater the value of I and vice versa. To let the modeled fish seek out locations that are favorable to them, the advection velocity v in Equation 10.2 is set to be proportional to the spatial gradient of the habitat index:

$$v = \chi_0 \frac{\partial I}{\partial x} \tag{10.4}$$

This ensures that the fish locally maximize the suitability of their habitat. However, if there was only the advective movement component, fish would not swim away from locations with a homogeneously low habitat index. To make sure that fish leave such unsuitable areas, the diffusive term for random movement is defined as

$$D = D_{\max} \left(1 - \frac{I}{\xi_0 + I}\right). \tag{10.5}$$

This choice for D implies that diffusion is high in unsuitable areas ($D(I = 0) = D_{\max}$) and low in suitable ones ($D(I \to \infty) = 0$), which lets fish stay in favorable habitats but forces them away from disadvantageous locations.

SEAPODYM generalizes the one-dimensional approach presented in the paragraph above to two spatial dimensions. To numerically approximate the solution of Equation 10.2 (or its two-dimensional counterpart, respectively), Bertignac et al. (1998) suggest to employ a finite difference method with the relatively coarse time step of one month. The core of SEAPODYM can be extended to an end-to-end model by having the habitat index function I and the feeding processes of the model in the reaction term H (not described here) depend on data from a biogeochemical ocean model.

10.2 Population Balances for End-To-End Modeling

Our novel approach to a modeling framework for the fish component of marine end-to-end ecosystem models is based on the theory of *population*

balances (Ramkrishna 2000). Population balance modeling is employed in a wide variety of disciplines (mainly related to engineering) to describe processes that include particulate entities that can break or aggregate (e. g., crystallization and polymerization). In a population balance model, the particulate entities are—as with an ADR model—represented via population density distributions. However, unlike in an ADR model, these density distributions do not only depend on time and spatial location but also on other continuous properties of the particles (so-called *internal coordinates*), such as their mass. Note that the term *balance* in population balances does not refer to an equilibrium in a steady state but is derived from the principle of mass conservation: in a population balance model, the entities that are tracked might change (e. g., due to growth or breakage) but the overall balance of their mass stays constant if no particles are introduced from the outside or are removed from the system. In fact, population balance models can represent time-dependent dynamics far away from a steady state.

To describe fish with population balances, let $\Omega_S = [a, b]$ be the—for the sake of simplicity one-dimensional—spatial domain and $\Omega_R = [r_{\min}, r_{\max}]$ be the interval of all possible masses of individuals of the fish stock to model. The density distribution

$$m : [0, t_{\max}] \times \Omega_S \times \Omega_R \to \mathbb{R},\ (t, x, r) \mapsto m(t, x, r) \tag{10.6}$$

specifies the average amount of fish (amount can either be the number of individuals or their combined biomass) that is present at the point in time t, at the spatial location x, and in which every individual has the mass r. By "average" amount, we mean that for any subset $S \subseteq \Omega_S \times \Omega_R$, the value of the integral

$$\int_S m(t, x, r)\ d(x, r) \tag{10.7}$$

equals to the amount (count or mass) of the fish that have a spatial location and an individual mass contained in S.

The evolution of m is governed by a *population balance equation*, which is a PDE that bears some similarity to an ADR equation:

$$\frac{\partial m}{\partial t} + \frac{\partial}{\partial x}(vm) + \frac{\partial}{\partial r}(gm) = H \tag{10.8}$$

In comparison to an ADR equation (Equation 10.2), a population balance equation does not include a diffusion term but, instead, features an additional advection term ($\frac{\partial}{\partial r}(gm)$). This additional advection term describes the transport of fish in the mass dimension—i. e., their growth. Thus, the growth of individuals is directly incorporated into the main model equation in a continuous way and does not have to be represented via discrete age/size classes as it is the case with ADR models.

Since no diffusion term is present in Equation 10.8, all fish movement in a population balance model is directed and has no random component. As with an ADR model, Equation 10.8 conserves the amount of fish present in the system (again, count or mass) if no fish are imported or exported at the boundaries of $\Omega_S \times \Omega_R$ and if the total net rate of birth and death in the population vanishes:

$$\int_{\Omega_S \times \Omega_R} H \, dx \equiv 0 \tag{10.9}$$

10.3 Discussion and Comparison of Approaches

This section discusses and compares the different approaches to a modeling framework for the fish component of end-to-end ecosystem models from a methodological perspective. We point out advantages and drawbacks of the existing modeling frameworks and describe how our novel approach based on population balances can help to overcome these drawbacks while maintaining most of the desirable properties of the other frameworks. A tabular summary of the advantages and drawbacks of the methodological approaches discussed in this section can be found in Table 10.1.

10.3.1 Spatial Replication of Aggregated Stock Models

Since the approach of spatial replication of aggregated stock models reuses traditional stock models, it can benefit from the vast amount of research that was conducted in this field. At the same time, however, the approach inherits some of the fundamental drawbacks of aggregated stock models. Specifically, the parameters of these models often have no clear biological meaning and their values are, therefore, difficult to determine from observational data. Additionally, established replication models, such as that by Fennel (2008),

Table 10.1. Comparison of the modeling frameworks.

Framework	Advantages	Disadvantages
Spat. repl.	• Reuse of existing models	• "Heuristic" generalization of traditional models • Difficult to parametrize • Typically, fixed structure of food web
IBMs	• Parameters observable in individual fish • Fish life cycle directly represented • Dynamic food web structure	• No formal mathematical framework • No guaranteed approximation quality of the super individual approach • (Pseudo-)random processes can have a strong effect on model results
ADR	• Derived from first principles (mass conservation) • Established numerical methods • Guaranteed approximation quality • Integrates well with existing biogeochemical models	• Difficult to parametrize • Fish life cycle not directly represented in main model equation
Pop. bal.	• Parameters observable in individual fish • Dynamic food web structure • Derived from first principles (mass conservation) • Established numerical methods • Guaranteed approximation quality • Integrates well with existing biogeochemical models • Fish life cycle directly represented	• Treatment of non-local effects in size dimension necessary

prescribe a fixed structure of the food web, which makes it impossible to investigate its dynamics.

From a methodological perspective, the spatial replication approach appears rather heuristic as its underlying idea is to use models which are characterized by the fact that they largely *ignore* space only to reintroduce it thereafter. This raises the question of what the spatial replication of an aggregated stock model actually represents. If we let the spatial cells that comprise a replication model become smaller and smaller, a mathematically-sound model should describe the real-world ecosystem to be modeled more accurately. However, it appears questionable whether high-level concepts of traditional stock models, such as recruitment and carrying capacities, are actually adequate at a small-scale level (what does it mean that the carrying capacity of this 1 cm^2 area of water is x?). It seems as if other abstractions—that are plausible on the level of the individuals which in the end determine the complete ecosystem with their interactions (cf. Huston et al. 1988)—are more suitable in this case.

10.3.2 Individual-Based Models

The insight that the dynamics of an ecosystem are in the end governed by the individuals comprising the system motivates the individual-based modeling approach. In this approach, all processes are described by rules that are formulated from the perspective of single individuals, which has the advantage that most parameters of such a model can be observed in individuals of the corresponding species. Additionally, the full life cycle of fish can easily be represented by different sub-models (i. e., specific sets of rules) for the different life stages. Another important advantage of individual-based ecosystem models is that the trophic structure of the ecosystem is an emergent feature of the model and does not have to be determined in advance.

The rule-based description of the actions of individuals in IBMs allows to easily specify relatively simple models that capture complex system dynamics. However, the way in which these rules are formalized (typically, a decision tree is used to select rules; see Grimm et al. 2006) implies that there is no rigid mathematical framework that would allow to study the mathematical properties of these models. This becomes especially apparent when focusing on the approximation quality of the predominantly applied

super individual approach. In Section 10.1.2, we explained that a super individual IBM can be viewed as an approximation to an "original" IBM which contains all individuals that are actually present in a system but cannot all be represented for performance reasons. This poses the question of how many super individuals are enough to adequately capture the dynamics of the system (clearly, an IBM with just two super individuals will not suffice to describe the spatial fish dynamics of a whole ocean basin). For the discretization of PDE models, such approximation guarantees—which depend on how fine or coarse the employed mesh is—can be given (see Chapter 4). The lack of a mathematical framework to study IBMs, however, makes it impossible to theoretically answer the question of how many super individuals have to be present for the model to produce accurate results.

Another disadvantage of IBMs—at least in the context of fish stock modeling—is that these models often incorporate (pseudo-)random processes (e. g., in the placement of individuals or in the order in which they feed on each other) that can potentially have strong effects on modeling results (DeAngelis and Rose 1992). Again, due to the lack of a rigid mathematical framework, we have no way of theoretically studying and quantifying the impact of randomness on model results.

10.3.3 Advection-Diffusion-Reaction Models

These disadvantages of IBMs are not present in ADR models as these are based on PDEs. For PDEs, there exist numerous well-developed and well-studied numerical approximation methods that are mathematically guaranteed to achieve a certain quality of approximation. An additional advantage of ADR models is that they use the same PDE-based modeling framework as biogeochemical ocean models typically do. In such an ocean model, chemical elements and plankton have to be advected (by currents) and diffused just like fish in an ADR model. This means that an ADR fish model can potentially be integrated into an existing biogeochemical model by merely adding another distribution field for fish (like one for nitrate or any other tracer) that has a particular advection velocity and diffusion coefficient. In this way, the ADR fish model would automatically reuse the whole solver infrastructure of the biogeochemical model implementation and biomass transfer between the two models would be trivial to implement.

An additional advantage of the ADR approach is that the ADR equation is not just an empirical model but can be derived from the principle of mass conservation. This strong theoretical foundation on mass conservation makes ADR models well suitable to track the transfer of biomass through the different trophic levels of an ecosystem.

A drawback of ADR models is that they can be hard to parametrize because they include some parameters that have no direct biological meaning (such as the maximum diffusion rate in SEAPODYM). Furthermore, the full life cycle of fish cannot be represented in the main equation of the model but, instead, different discrete age classes have to be assumed that are described by individual density distributions.

10.3.4 Population Balance Models

Our modeling approach based on population balances combines the advantages of IBMs and ADR models while avoiding their main drawbacks. As there is no random movement component in a population balance equation, all processes can be formulated from the perspective of individual fish ("how would a fish behave at these coordinates given these current population distributions?"). This implies that, like with IBMs, the parameters of a population balance model are mostly observable in individual fish. Also as with IBMs, the foraging process in a system of population balance equations can be modeled to be opportunistic, which results in a dynamic food web structure.

Like the ADR approach, the theory of population balances is based on PDEs and can be derived from the same theoretical principle of mass conservation (see Ramkrishna 2000). This implies that population balance models can also rely on the well-established theory of numerical PDE solvers and can benefit from the same approximation guarantees that hold for ADR models. Likewise, the integration of the population balance approach with existing biogeochemical models is relatively simple as both share the same mathematical framework.

In contrast to the ADR approach, population balance models incorporate the different life stages of fish directly into the main model equation in a continuous way. While this allows to seamlessly represent the full life cycle of individuals, it introduces the drawback of non-local effects. By non-local effects, we mean that the source term $H(t, x, r)$ in a population

balance model (see Equation 10.8) does not only depend on the distribution of fish at the point (x, r) but also at points "further away" (as we will see shortly, this refers to a distance in the size and not in the space dimension, hence the quotation marks). Specifically, this is relevant to the predation process, which is described by H. Predatory fish of size r_0 prey upon all other fish that are smaller than a certain fraction of their body size ($r < r_0/\tau$). Therefore, $H(t, x, r)$ depends on integral terms of the form

$$\int_{r_{\min}}^{r/\tau} m(t, x, \phi)\, d\phi. \tag{10.10}$$

Non-local effects, such as this one, can tremendously increase computational costs when solving a PDE system because for each evaluation of H, we have to evaluate m in $\mathcal{O}(n_r)$ points (with n_r being the number of discretization points in the size dimension). A typical PDE solver algorithm would require $\mathcal{O}(n_x n_r)$ operations for a single time step if the evaluation of H would be of complexity $\mathcal{O}(1)$. However, due to the non-local effect, we end up with computational costs of $\mathcal{O}(n_x n_r^2)$ because for each discrete spatial point—of which there are $\mathcal{O}(n_x n_r)$—, we have to evaluate H, which is of complexity $\mathcal{O}(n_r)$.

Luckily, we can circumvent this problem and make the approximation of a population balance model as efficient as the approximation of an ADR model: if we choose a regular mesh (cf. Chapter 4) for our numerical approximation algorithm with discretization points $r_1 < r_2 < \ldots < r_{n_r}$ in the size dimension, it holds for $i = 2, \ldots, n_r$ that

$$\int_{r_{\min}}^{r_i/\tau} m(t, x, \phi)\, d\phi = \int_{r_{\min}}^{r_{i-1}/\tau} m(t, x, \phi)\, d\phi + \int_{r_{i-1}/\tau}^{r_i/\tau} m(t, x, \phi)\, d\phi. \tag{10.11}$$

This implies that we can iteratively compute the integrals in $H(t_0, x_0, r_i)$ by using the value of the integral for r_{i-1} and then adding an integral that only covers a small integration interval. The computation of this latter integral requires the evaluation of m in just two points. Therefore, the evaluation of H at time t_0 and location x_0 in *all* r_i *together* only requires $\mathcal{O}(n_r)$ operations. This again means that the overall algorithm costs are of the order $\mathcal{O}(n_x n_r)$, which is as expensive as if there were no non-local effects.

In the following chapter, we present a fish stock model based on population balances that translates the methodological benefits of the approach into practice.

Chapter 11

A Fish Model Based on Population Balances

This chapter introduces the *Sprat Marine Ecosystem Model*, which couples a fish model with existing biogeochemical models to attain an end-to-end representation of the ecosystem. In the Sprat Model, fish are described by density functions on a combined space-body size domain and the evolution of the densities is governed by population balance equations (Ramkrishna 2000). The model can be formulated using either two or three spatial dimensions besides the size dimension. In this chapter, we present a vertically integrated 2D+1D version (two spatial plus the size dimension) of the Sprat Model but all our modeling approaches can be extended to a 3D+1D version in a straightforward manner.

In Section 11.1, we introduce a simple Nutrient, Phytoplankton, Zooplankton (NPZ) model which can be coupled with the Sprat Model in place of a full-blown biogeochemical ocean model. We employ this NPZ model in our first evaluation of the Sprat Model in Chapter 15. A summary of the parameters of the Sprat Model (including their units) can be found in Appendix A.

11.1 A Simple NPZ Model

NPZ models are a well-established tool in oceanography to describe the basic dynamics of plankton in the ocean (Franks 2002). The NPZ model we present here only contains the three state variables given in Table 11.1 and does not explicitly feature space. Regarding how to couple this model with the spatially-explicit Sprat Model, see Section 11.1.3.

Table 11.1. State variables of the NPZ model.

Symbol	Description	Unit
$N(t)$	Nutrients	mmol N m^{-2}
$P(t)$	Phytoplankton	mmol N m^{-2}
$Z(t)$	Zooplankton	mmol N m^{-2}

Table 11.2. Functions and parameters of a general NPZ model.

Symbol	Description	Unit
$\iota(t)$	Daily integrated solar irradiance	MJ m^{-2}
$f(\iota)$	Phytoplankton response to irradiance	dimensionless
$g(N)$	Phytoplankton nutrient uptake	s^{-1}
$h(P)$	Zooplankton grazing rate	s^{-1}
$i(P)$	Phytoplankton mortality rate	s^{-1}
$j(Z)$	Zooplankton natural mortality rate	s^{-1}
$r(N)$	Vertical nutrient transport rate	mmol N m^{-2} s^{-1}
γ	Zooplankton assimilation efficiency	dimensionless
$G(t)$	Plankton consumption rate by fish	mmol N m^{-2} s^{-1}

The general time-dependent dynamics of the three state variables are given by

$$\frac{dP}{dt} = f(\iota)g(N)P - h(P)Z - i(P)P \tag{11.1}$$

$$\frac{dZ}{dt} = \gamma h(P)Z - j(Z)Z - G \tag{11.2}$$

$$\frac{dN}{dt} = -f(\iota)g(N)P + (1-\gamma)h(P)Z + i(P)P + j(Z)Z + r(N), \tag{11.3}$$

where the meaning of the symbols is described in Table 11.2. The transfer functions (f, g, h, i, j, r, and G) define the mass fluxes (expressed via nitrogen content) between the different compartments of the model.

11.1.1 Specification of the Transfer Functions

In the following, we present the functional forms we chose for the transfer functions of our NPZ model. For an overview on alternative modeling approaches for these forms, see Franks (2002).

The growth of phytoplankton biomass P depends on the availability of both light and nutrients. To model the effect of light on the growth process, we use the saturating response function

$$f(\iota) = \frac{\iota}{\iota_s + \iota}, \tag{11.4}$$

where ι_s is the half saturation constant for the available daily integrated irradiance ι. For the calculation of $\iota(t)$ itself, see Section 11.1.2.

Uptake of nutrients by phytoplankton is given by the Michaelis-Menten equation

$$g(N) = \frac{V_m N}{k_s + N} \tag{11.5}$$

with V_m being the maximal uptake rate and k_s the half saturation constant for the available nutrients.

The rate of phytoplankton consumption by zooplankton is given via Ivlev's functional response

$$h(P) = R_m(1 - \exp(-\lambda_s P)), \tag{11.6}$$

where R_m is the maximal phytoplankton uptake rate and λ_s—the Ivlev constant—influences how fast saturation is achieved. The actual fraction of phytoplankton that is incorporated into the zooplankton biomass is influenced by the assimilation efficiency γ. The remaining part ($(1-\gamma)h(P)Z$) is excreted and, therefore, added to the nutrient pool again.

Phytoplankton mortality is modeled via the quadratic density-dependent mortality rate

$$i(P) = \varepsilon_P P. \tag{11.7}$$

For zooplankton, we define two mortality rates: first, a quadratic rate

$$j(Z) = \varepsilon_Z Z \tag{11.8}$$

that is directly applied in the model in Equation 11.2. Second, a linear rate

$$j_f(Z) = \varepsilon_f \tag{11.9}$$

that is used to limit the amount of zooplankton consumed by fish G (cf. Section 11.2.5).

For reasons of simplicity and model stability, we do not explicitly describe the vertical transport of water and remineralization processes associated with this transport but parametrize them via

$$r(N) = \eta_r(N_m - N), \tag{11.10}$$

where η_r sets the speed of remineralization and N_m is the maximum nutrient concentration. The functional form of $r(N)$ mimics diffusion processes by assuming a constant nutrient concentration N_m outside the region covered by the NPZ model (e. g., in the deep ocean). Depending on whether N is below or above N_m in the model, nutrients diffuse into or out of the model region at a rate depending on the concentration difference.

11.1.2 Solar Irradiance

To model the daily integrated solar irradiance that can be used for photosynthesis $\iota(t)$, we follow the approach of Brock (1981). First, we describe the daily integrated amount of solar radiation at the top of the atmosphere, which depends on the day of the year and latitude as well as three derived quantities:

1. Distance from the Earth to the Sun
2. Declination of the Sun
3. Sunset hour angle

In a second step, we specify how the solar radiation is attenuated on its way through the atmosphere to finally compute $\iota(t)$.

Solar Constant and Distance From the Earth to the Sun

The energy received from the Sun per unit time and unit area perpendicular to the rays just outside the atmosphere of Earth is given by the empirical

solar constant

$$I_0 = 1353 \text{ W m}^{-2} = 4.8708 \text{ MJ h}^{-1} \text{ m}^{-2} \tag{11.11}$$

for the mean distance of Earth and Sun. Since the distance between Earth and Sun is not constant throughout the year, we introduce a correction factor for I_0:

$$R_1(d) = \frac{1}{\sqrt{1 + \frac{1}{30} \cos\left(2\pi \frac{d}{365}\right)}} \tag{11.12}$$

To adjust the solar constant for the ellipticity of the orbit of the Earth, I_0 has to be divided by $R_1(d)^2$ (see Equation 11.15 below). Here and in the following, d is the day of the year (with January 1 being $d = 1$). We write $d(t)$ where we want to express the day of the year for time t from the NPZ model.

Declination of the Sun

The tilted rotational axis of Earth relative to its orbital plane makes it necessary to consider the declination of the Sun, which is the (north-positive) angle between the Sun and Earth's equatorial plane at solar noon. It depends only on the time of the year and can be approximated by

$$D_1(d) = 23.45^\circ \cdot \frac{\pi}{180^\circ} \cdot \sin\left(2\pi \frac{284 + d}{365}\right). \tag{11.13}$$

Sunset Hour Angle

The length of daytime influences the total amount of radiation that is received during a day and varies with time of year and latitude. It can be modeled via the angle between the south point (where the Sun is at solar noon) and the point of sunset. This angle—which corresponds to half the length of the daytime period—is called *sunset hour angle* and can be computed via

$$W_1(L, d) = \arccos\left(-\tan\left(\frac{\pi L}{180^\circ}\right) \cdot \tan\left(D_1(d)\right)\right) \tag{11.14}$$

with L being latitude in degrees (north-positive).

Daily Integrated Solar Radiation at the Top of the Atmosphere

With the quantities introduced so far, we can calculate the daily integrated amount of solar radiation at the top of the atmosphere

$$I_1(L,d) = \frac{24\text{ h}}{\pi} \cdot \frac{I_0}{R_1(d)^2} \cdot \left[W_1(L,d) \cdot \sin\left(\frac{\pi L}{180^\circ}\right) \cdot \sin\left(D_1(d)\right) \right. \tag{11.15}$$

$$\left. + \sin\left(W_1(L,d)\right) \cdot \cos\left(\frac{\pi L}{180^\circ}\right) \cdot \cos\left(D_1(d)\right) \right], \tag{11.16}$$

which has the unit MJ m^{-2}. Note that this equation only holds for latitudes where the Sun does rise and set on all days of the year.

Attenuation of the Radiation

Since we are interested only in the amount of radiation that is available to phytoplankton for photosynthesis, we have to consider three main influences that attenuate the solar radiation available at the top of the atmosphere given by I_1:

1. Transmission losses: even without clouds, only a fraction of the radiation is actually transmitted from the top of the atmosphere through the surface of the ocean.
2. Photosynthetically Active Radiation (PAR): phytoplankton can utilize only certain parts of the transmitted wavelength spectrum for photosynthesis.
3. Cloud cover: clouds decrease the amount of radiation that reaches the ocean surface.

In accordance with Evans and Parslow (1985), we assume that transmission losses are constant and introduce the transmissivity constant $C_T = 0.75$. With regard to PAR, we also consider a constant fraction ($C_{PAR} = 0.5$) of the wavelength spectrum to be suitable for photosynthesis (Fasham et al. 1990). For attenuation due to cloud coverage, we follow Reed (1977), who derives

the empirical relation

$$\frac{Q_C}{Q_0} = 1 - 0.62 \cdot C_c + \frac{0.342}{\pi} \cdot Z_1, \tag{11.17}$$

where Q_C is the insolation under cloudy conditions, Q_0 the insolation under clear skies, C_c the percentage of cloud coverage, and Z_1 the solar altitude at noon (in radians). Noon solar altitude Z_1 can be derived from

$$\begin{aligned} \cos\left(\frac{\pi}{2} - Z_1(L,d)\right) &= \sin(D_1(d)) \sin\left(\frac{\pi L}{180^\circ}\right) && (11.18)\\ &\quad + \cos(D_1(d)) \cos\left(\frac{\pi L}{180^\circ}\right) && (11.19)\\ &= \cos\left(D_1(d) - \frac{\pi L}{180^\circ}\right). && (11.20) \end{aligned}$$

It follows that

$$Z_1(L,d) = \frac{\pi}{2} - \arccos\left(\cos\left(D_1(d) - \frac{\pi L}{180^\circ}\right)\right). \tag{11.21}$$

Therefore, for latitudes where the Sun rises and sets on all days of the year, it holds:

$$Z_1(L,d) = \frac{\pi}{2} - \left|D_1(d) - \frac{\pi L}{180^\circ}\right| \tag{11.22}$$

With all of this, we can finally express the daily integrated solar irradiance available for photosynthesis as

$$\begin{aligned} \iota(t) = C_T \cdot C_{PAR} \cdot &\left(1 - 0.62 \cdot C_c + \frac{0.342}{\pi} \cdot Z_1(L, d(t))\right) && (11.23)\\ &\cdot I_1(L, d(t)). && (11.24) \end{aligned}$$

11.1.3 Coupling With the Spatially-Explicit Fish Model

When coupling the NPZ model presented above with the spatially-explicit Sprat Model, we assume—for computational simplicity—that there are no currents in the ocean and no diffusion takes place. Therefore, we can assign

an instance of the NPZ model to every discrete spatial point of the mesh for approximating the solution of the Sprat Model (cf. Chapter 4) and compute the trophic interactions for each of these points in isolation.

In contrast to the NPZ model, which uses a nitrogen mass budget, the Sprat Model uses a carbon budget (see below). We convert between the two by relying on the Redfield stoichiometric ratio for plankton of C : N : P = 106 : 16 : 1 (Redfield et al. 1966). Assuming this ratio, it holds that

$$1 \text{ mmol N m}^{-2} \cdot \frac{106 \text{ C}}{16 \text{ N}} = \frac{106}{16} \text{ mmol C m}^{-2} \tag{11.25}$$

$$= 12 \cdot \frac{106}{16} \text{ mg C m}^{-2} \tag{11.26}$$

$$= 12 \cdot 10^{-6} \cdot \frac{106}{16} \text{ kg C m}^{-2}. \tag{11.27}$$

We define

$$\rho = 12 \cdot 10^{-6} \cdot \frac{106}{16} \frac{\text{kg C m}^{-2}}{\text{mmol N m}^{-2}} \tag{11.28}$$

as a conversion constant.

11.2 The Fish Model

In order to introduce our fish model, we first describe how in the model fish are represented as density functions (Section 11.2.1) and how the evolution of these densities is governed by a system of PDEs (Section 11.2.2). Since our model focuses on trophic interactions in the ecosystem, we discuss in detail how predation is represented in Section 11.2.3. We then proceed by specifying the functional forms of the terms in the main PDE system, namely, for the spatial transport of fish (Section 11.2.4), for the effects of forcing factors such as fishing (Section 11.2.5), and for the transport in the size dimension—i. e., fish growth (Section 11.2.6).

11.2.1 Representing Fish as Density Functions

As described in Chapter 10, we choose a modeling approach based on population balance equations in order to introduce fish into existing bio-

geochemical ocean models. In our model, fish of species $\kappa = 1, \ldots, n$ are represented by the average carbon mass distribution

$$m^{[\kappa]} \colon [0, t_{\max}] \times \Omega \to \mathbb{R},\ (t, x, y, r) \mapsto m^{[\kappa]}(t, x, y, r) \tag{11.29}$$

with the combined space-size domain

$$\Omega = \Omega_S \times [r_{\min}, r_{\max}]. \tag{11.30}$$

Here, Ω_S is a two-dimensional polygon domain representing the vertically integrated ocean and $r_{\min}$ and $r_{\max}$ are the minimal and maximal masses of individual fish in the model, respectively. Note that the model uses a continuous size dimension instead of discrete size classes as typically found in fish models (cf. Chapter 3).

We let the model capture the size (interpreted as mass) instead of the age of individuals for two reasons. First, we are interested in how biomass is transfered through the different components of the ecosystem and, hence, we have to know the mass of individual fish. For this purpose, we must represent the size of the individual explicitly as it is an unrealistic assumption that all fish of the same age share the same mass. The second reason for modeling fish mass rather than age is that most of the relevant processes in the ecosystem depend on size more than on age. For example, selectivity of fishing gear is mostly determined by size and fish can reach maturity at various ages (Schnute 1987).

For every point in time t, every spatial point (x, y), and every size r, the value $m^{[\kappa]}(t, x, y, r)$ describes how much carbon mass of species κ is "on average" at these coordinates. By "on average," we mean that for every $V \subseteq \Omega$, the value of

$$\int_V m^{[\kappa]}(t, x, y, r)\, d(x, y, r) \tag{11.31}$$

represents the carbon mass of fish from species κ contained in volume V at time t. With carbon mass, we refer to the absolute mass of the carbon content of the dry mass of fish (both $m^{[\kappa]}$ and r are carbon masses). Our model assumes constant ratios for carbon mass to dry mass ($C^{[\kappa]}_{\mathrm{C/dm}}$) and dry mass to wet mass ($C^{[\kappa]}_{\mathrm{d/wm}}$) for each species. For convenience, we define

$M^{[\kappa]}$ to denote the wet mass distribution of species κ:

$$M^{[\kappa]}(t,x,y,r) = \frac{1}{C^{[\kappa]}_{\mathrm{C/dm}}} \cdot \frac{1}{C^{[\kappa]}_{\mathrm{d/wm}}} \cdot m^{[\kappa]}(t,x,y,r) \tag{11.32}$$

Units in the fish model are always SI base units (kg, m, s; with the exception of temperature, which is expressed in °C). In particular, the unit of $m^{[\kappa]}$ is kg C m^{-2} (kg C)$^{-1}$ = m^{-2}.

We choose to let $m^{[\kappa]}$ represent carbon mass instead of individual counts because, in this way, it is straightforward to guarantee mass conservation as will become apparent below. We focus on mass conservation rather than number conservation because at ocean scale, the *individual* fish is neglectable and it is of greater interest how biomass is transported through the ecosystem and its different trophic levels. However, average carbon mass density $m^{[\kappa]}$ can still be converted to average individual count density $u^{[\kappa]}$ via

$$u^{[\kappa]}(t,x,y,r) = \frac{m^{[\kappa]}(t,x,y,r)}{r}. \tag{11.33}$$

Length-Weight Relationships

In some contexts, we need the lengths of individuals instead of their weight. To convert wet mass M (in kg) to length l (in m), we use the power law

$$M = a^{[\kappa]} l^{b^{[\kappa]}}, \tag{11.34}$$

which implies that

$$l = \left(\frac{M}{a^{[\kappa]}}\right)^{1/b^{[\kappa]}}. \tag{11.35}$$

In the literature, values for a and b are often given for combinations of units other than kg-m. In this case, the intercept a has to be transformed as in the following example for g-cm:

$$a'\,(\mathrm{kg, m}) = \frac{100^b}{1000} \cdot a\,(\mathrm{g, cm}) \tag{11.36}$$

Life Stages

Although our model represents fish size using a continuous dimension, we need to differentiate between different life stages, each with their specific characteristics. For example, fish are only able to reproduce after they have reached maturity. Individuals transition from one life stage to the next once they grow past a certain size/mass.

In the following, we list the masses that mark transitions relevant in our model:

- $w_{\text{egg}}^{[\kappa]}$: dry weight of one egg from species κ
- $w_{\text{forage}}^{[\kappa]}$: dry mass at which larvae of species κ start to forage on their own (prior to that point, the larvae feed on their yolk sac)
- $M_{\text{plankton}}^{[\kappa]}$: wet mass up to which individuals of species κ consume zooplankton
- $M_{\text{mature}}^{[\kappa]}$: wet mass at maturity of species κ
- $M_{\text{max}}^{[\kappa]}$: maximum wet weight of an individual from species κ

11.2.2 Evolution of the Fish Densities

The evolution of the fish carbon mass distributions $m^{[\kappa]}$ is governed by the following system of population balance equations:

$$\frac{\partial}{\partial t} m^{[\kappa]} + \frac{\partial}{\partial x} q_x^{[\kappa]} m^{[\kappa]} + \frac{\partial}{\partial y} q_y^{[\kappa]} m^{[\kappa]} + \frac{\partial}{\partial r} g^{[\kappa]} m^{[\kappa]} = H^{[\kappa]} \tag{11.37}$$

The vector $q^{[\kappa]} = (q_x^{[\kappa]}, q_y^{[\kappa]})$ is the spatial advection velocity of species κ with unit m s^{-1}. $g^{[\kappa]}$ is a growth rate with unit kg C s^{-1} and $H^{[\kappa]}$ is a source term with unit kg C m^{-2} (kg C)$^{-1}$ s^{-1} = m^{-2} s^{-1}. In order to be well-posed, Equation 11.37 has to be supplemented with appropriate initial and boundary conditions.

Using Reynolds' transport theorem, it can be shown that Equation 11.37 conserves mass if no sources and sinks are present ($H \equiv 0$) and no mass is lost or introduced at the boundary (Ramkrishna 2000). That means that in

this case

$$\sum_{\kappa=1}^{n} \int_{\Omega} m^{[\kappa]}(t,x,y,r)\, d(x,y,r) \equiv \text{const.} \tag{11.38}$$

holds. Appropriate numerical solvers preserve the property of mass conservation also for the discrete approximation of the solution but typically introduce a small amount of numerical (i. e., spurious) diffusion in all dimensions of Ω. Due to the numerical diffusion in the size dimension, conservation of the number of individuals $u^{[\kappa]} = m^{[\kappa]}/r$ can in general *not* be guaranteed for discrete approximations of the solution. The other way around, if we chose the individual count $u^{[\kappa]}$ as our unknown variable in Equation 11.37, we would be able to guarantee conservation of individual numbers but not conservation of mass. Since we are interested in tracing the transport of biomass through the different compartments of the ecosystem, we opt for mass conservation and formulate Equation 11.37 in terms of the carbon mass distribution of the fish $m^{[\kappa]}$.

11.2.3 Trophic Interactions

A key feature of marine food webs is opportunistic predation based on body size: fish tend to feed on organisms of all taxa, provided that the predator fish are physically able to ingest these prey organisms (Shin et al. 2010). Therefore, in ecological models of fish, it is typically assumed that a predator can prey on other fish as soon as the ratio of predator to prey body length is larger than the *minimal predator-prey length ratio* τ' (Shin and Cury 2001).

Since our model operates with body masses instead of fish lengths, we need to be able to convert minimal predator-prey *length* ratio τ' to minimal predator-prey *mass* ratio τ. Using Equation 11.34 for a minimal mass ratio $M_{\text{pred}}/M_{\text{prey}}$, this can be accomplished via

$$\tau = \frac{M_{\text{pred}}}{M_{\text{prey}}} = \left(\frac{l_{\text{pred}}}{l_{\text{prey}}}\right)^{b} = (\tau')^{b}, \tag{11.39}$$

where we ignore the dependency of a and b on κ. The latter is justified by the relatively small observed variation of these two parameters (Froese and Pauly 2015). Furthermore, $b \approx 3$ holds for many species (ibid.), which is in

agreement with the fact that the volume (and, hence, the mass) of a body changes with length to power 3. We therefore assume:

$$\tau = (\tau')^3 \tag{11.40}$$

For an individual from species κ at coordinates (t, x, y, r), the concentration of prey from species i is given by

$$\int_{r_{\min}}^{r/\tau} m^{[i]}(t, x, y, \phi)\, d\phi \tag{11.41}$$

and the concentration of potential predators from species i is described by

$$\int_{\min(\max(\tau r, r^{[i]}_{\text{forage}}), r_{\max})}^{r_{\max}} m^{[i]}(t, x, y, \phi)\, d\phi \tag{11.42}$$

with $r^{[i]}_{\text{forage}} = C^{[i]}_{\text{C/dm}} w^{[i]}_{\text{forage}}$. We require that $i \neq \kappa$ because we currently do not consider cannibalism in the Sprat Model (obviously, it is trivial to do so if necessary).

For convenience, we define density functions for prey and predator concentrations. The concentration of prey is given by

$$c^{[\kappa]}_{\text{prey}}(t, x, y, r) = \sum_{\substack{i=1 \\ i \neq \kappa}}^{n} \int_{r_{\min}}^{r/\tau} m^{[i]}(t, x, y, \phi)\, d\phi + c^{[\kappa]}_{Z}(t, x, y, r), \tag{11.43}$$

where $c^{[\kappa]}_{Z}$ is the concentration of zooplankton that individuals from species κ at (t, x, y, r) can prey on. It is defined as

$$c^{[\kappa]}_{Z}(t, x, y, r) = \begin{cases} \rho Z(t, x, y) & \text{if } r \leqslant C^{[\kappa]}_{\text{C/dm}} C^{[\kappa]}_{\text{d/wm}} M^{[\kappa]}_{\text{plankton}} \\ 0 & \text{otherwise} \end{cases} \tag{11.44}$$

with ρ from Equation 11.28. The concentration of predators able to feed on an individual from species κ at (t, x, y, r) is described by

$$c_{\text{pred}}^{[\kappa]}(t,x,y,r) = \sum_{\substack{i=1 \\ i\neq\kappa}}^{n} \int_{\min(\max(\tau r, r_{\text{forage}}^{[i]}), r_{\max})}^{r_{\max}} m^{[i]}(t,x,y,\phi)\, d\phi. \tag{11.45}$$

11.2.4 Spatial Velocity

The spatial transport velocity of fish from species κ is given by:

$$q^{[\kappa]} = \begin{cases} V_{\text{currents}} & \text{if } r < C_{\text{C/dm}}^{[\kappa]} w_{\text{forage}}^{[\kappa]} \\ V_{\text{currents}} + V_{\text{reactive}}^{[\kappa]} + V_{\text{predictive}}^{[\kappa]} & \text{otherwise} \end{cases} \tag{11.46}$$

For simplicity, we assume that fish which do not forage on their own have no active movement and are transported in space solely by currents (V_{currents}). All other individuals are also affected by V_{currents} but exhibit two additional movement terms that together represent their active locomotion.

In modeling the active movement of individuals, we follow the approach of Fernö et al. (1998), who describe fish swimming as being governed by two processes (see also Neill 1979). The first process is *reactive movement*, which means that the fish choose a swimming speed and direction based on their immediate surrounding. Thereby, they react to approaching predators, from which they flee, or to prey patches, by which they are attracted. The second process is *predictive movement*, which is based not on stimuli from the current environment of the fish but rather on its experience and its instinct (regarding learning in fish, see Brown et al. 2008). Examples of this second type of movement are feeding and spawning migrations.

Currents

If a body with zero velocity is put into a flow field with constant velocity everywhere, the difference in speed between the flow medium and the body will decay exponentially in time (Rieutord 2014). Therefore, we set V_{currents} to be identical with the velocity field supplied by the ocean model with which the Sprat Model is coupled.

Reactive Movement

Reactive movement can be modeled based on a *habitat index function* $\mathfrak{H}^{[\kappa]}$, which describes how favorable a location is to an individual fish (cf. Bertignac et al. 1998). Such a habitat index function can consider the amount of predators and prey at any given place as well as other environmental parameters, such as temperature and dissolved oxygen concentration. Using the index function, the fish can be modeled as trying to locally maximize $\mathfrak{H}^{[\kappa]}$ by swimming in a certain direction.

A well-known functional form for $\mathfrak{H}^{[\kappa]}$ was introduced by Gilliam and Fraser (1987), who model fish as seeking out locations with the highest ratio of foraging rate to mortality rate, which implies

$$\mathfrak{H}^{[\kappa]} = \frac{c_{\text{prey}}^{[\kappa]}}{c_{\text{pred}}^{[\kappa]}}. \tag{11.47}$$

However, Railsback et al. (1999) point out several practical problems with this approach. First, the habitat index is undefined in the absence of predators. Second, it underestimates the importance for the fish to avoid predation in comparison to the importance of foraging as both are equally relevant in Equation 11.47. Instead, fish would likely accept a period of starvation in order to avoid being killed. These concerns can be addressed by introducing an additive constant into the denominator of Equation 11.47 and a scaling exponent that influences the importance of predator avoidance in comparison to foraging:

$$\mathfrak{H}^{[\kappa]} = \frac{c_{\text{prey}}^{[\kappa]}}{(c_{\text{pred}}^{[\kappa]} + C)^{\alpha}} \tag{11.48}$$

An alternative to the functional forms of Equations 11.47 and 11.48 is to ignore the risk of predation and simply choose

$$\mathfrak{H}^{[\kappa]} = c_{\text{prey}}^{[\kappa]}. \tag{11.49}$$

Since this approach of ignoring predation risk has been successfully employed by Radtke et al. (2013) in their end-to-end model for the Baltic Sea,

we opt for defining the habitat index function for our model as given in Equation 11.49.

In order to let the fish find the habitat that is locally optimal for them, it has been proposed (e. g., by Bertignac et al. 1998) to set reactive movement velocity proportional to the (spatial) gradient of the habitat index:

$$V_{\text{reactive}}^{[\kappa]} = \xi_0 \left(\frac{\partial}{\partial x} \mathfrak{H}^{[\kappa]}, \frac{\partial}{\partial y} \mathfrak{H}^{[\kappa]} \right) \tag{11.50}$$

We found that this optimization strategy is unsuitable for our model because it is too locally confined and produces large accumulations of fish in local maxima of $\mathfrak{H}^{[\kappa]}$ that are relatively inadequate compared to other close-by locations. Therefore, we choose to assume that fish have perfect information about their surroundings within a certain radius r_{view} and swim to an absolute maximum of $\mathfrak{H}^{[\kappa]}$ within this area.

The area of perfect information is given by

$$K(x,y) = \left\{ (v,w) \in \mathbb{R}^2 : \|(v-x, w-y)\| \leqslant r_{\text{view}} \right\} \cap \Omega_S. \tag{11.51}$$

Within its bounds, we are looking for the spatial point with maximum habitat index and choose our swimming direction as

$$\delta_o^{[\kappa]}(t,x,y,r) = \left(\arg\max_{(v,w)\in K(x,y)} \mathfrak{H}^{[\kappa]}(t,v,w,r) \right) - (x,y). \tag{11.52}$$

Equation 11.52 assumes that there is exactly one absolute maximum—if there is more than one, we select the one that minimizes $\|\delta_o^{[\kappa]}\|$. The problem of finding an absolute maximum in $K(x,y)$ is computationally feasible since all densities are discretized for approximating the solution to Equation 11.37 (hence, we need to check only a finite amount of points to find the desired maximum).

Finally, we can define the reactive movement velocity as proportional to $\delta_o^{[\kappa]}$ as

$$V_{\text{reactive}}^{[\kappa]} = \zeta^{[\kappa]} \delta_o^{[\kappa]} \tag{11.53}$$

with

$$\xi^{[\kappa]} = \begin{cases} \frac{\varsigma^{[\kappa]} I^{[\kappa]}(r)}{\|\delta_o^{[\kappa]}\|} & \text{if } \|\delta_o^{[\kappa]}\| > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{11.54}$$

where $\varsigma^{[\kappa]}$ is the cruise speed of fish from species κ in body length per second.

As an alternative to supplying a constant cruise speed in body length per second, one could use the approach of Videler (1993), who models swimming speed using stride length and tail beat frequency at cruise speed. His approach has the advantage that tail beat frequency explicitly depends on environment temperature and body size. However, an important drawback (and the reason why we choose *not* to use it) is that there is little data published on stride lengths and tail beat frequencies at cruise speed for different fish species.

Predictive Movement

The fish in our model employ predictive movement strategies in two cases:

1. If prey abundance is too low within the radius of perfect information r_{view} to sustain the fish, they initiate a feeding migration until they find more suitable habitat.
2. During the mating season, fish travel to spawning grounds.

Therefore, we define a migration velocity which is composed of feeding and spawning migration:

$$V_{\text{migration}}^{[\kappa]} = V_{\text{feeding}}^{[\kappa]} + V_{\text{spawning}}^{[\kappa]} \tag{11.55}$$

If there is no reactive movement (because no location within r_{view} has a higher habitat index than the current one) and there are more fish at the current location than can be sustained by the respective prey concentration, the fish have to migrate. This can be formalized as

$$V_{\text{feeding}}^{[\kappa]} = \begin{cases} (1,0) & \text{if } \|V_{\text{reactive}}^{[\kappa]}\| = 0 \wedge m^{[\kappa]} > \zeta_0 c_{\text{prey}}^{[\kappa]} \\ 0 & \text{otherwise,} \end{cases} \tag{11.56}$$

where ζ_0 is a linear scaling constant. The movement strategy we employ for feeding migrations is simple: swim in the direction of the positive x axis (the swimming speed will be adjusted later on). A similar strategy ("swim westwards") has actually been observed in herring in the Norwegian Sea (Fernö et al. 1998). Of course, more sophisticated strategies can be used, such as swimming to known feeding locations etc.

For spawning migrations, we assume that there is a single spawning ground for each species given by its center $s^{[\kappa]} = (x_s^{[\kappa]}, y_s^{[\kappa]})$. Furthermore, a fixed mating season is described by the subset $S^{[\kappa]} \subset [0,1]$, where the interval represents time of year (with 0 being the first moment and 1 the last moment of the year). We introduce the function $\theta(t) \in [0,1]$ to convert simulation time t to time of year. With these prerequisites, the spawning migration velocity is given by

$$V_{\text{spawning}}^{[\kappa]} = \begin{cases} (x_s^{[\kappa]} - x, y_s^{[\kappa]} - y) & \text{if } \theta(t) \in S^{[\kappa]} \\ 0 & \text{otherwise.} \end{cases} \tag{11.57}$$

$V_{\text{spawning}}^{[\kappa]}$ formalizes that fish are attracted by the spawning region during the mating season. Its magnitude is larger the further away the fish are from the spawning ground (the actual swimming speed will be adjusted later on). The approach can, of course, be extended to multiple spawning areas represented by regions rather than by points if necessary.

If there is a reason for the fish to migrate (i. e., $\|V_{\text{migration}}^{[\kappa]}\| > 0$), we use the predictive movement velocity to combine migratory and reactive movement:

$$V_{\text{predictive}}^{[\kappa]} = \begin{cases} -V_{\text{reactive}}^{[\kappa]} + \\ \varsigma^{[\kappa]} I^{[\kappa]}(r) \frac{V_{\text{reactive}}^{[\kappa]} + V_{\text{migration}}^{[\kappa]}}{\|V_{\text{reactive}}^{[\kappa]} + V_{\text{migration}}^{[\kappa]}\|} & \text{if } \|V_{\text{migration}}^{[\kappa]}\| > 0 \wedge \\ & \quad \|V_{\text{reactive}}^{[\kappa]} + V_{\text{migration}}^{[\kappa]}\| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{11.58}$$

11.2.5 Sources and Sinks

The source term of the model is given by

$$H^{[\kappa]} = \begin{cases} -m^{[\kappa]} \left(H^{[\kappa]}_{\text{fishing}} + H^{[\kappa]}_{\text{backgr}} \right) + H^{[\kappa]}_{\text{repr}} \\ \quad +E^{[\kappa]} I^{[\kappa]} - L^{[\kappa]} - R^{[\kappa]} & \text{if } r \geqslant C^{[\kappa]}_{\text{C/dm}} w^{[\kappa]}_{\text{forage}} \\ -m^{[\kappa]} H^{[\kappa]}_{\text{backgr}} + H^{[\kappa]}_{\text{repr}} - L^{[\kappa]} & \text{otherwise.} \end{cases} \tag{11.59}$$

For fish which are able to actively forage, the source term consists of losses due to fishing ($H^{[\kappa]}_{\text{fishing}}$), background mortality ($H^{[\kappa]}_{\text{backgr}}$), predation ($L^{[\kappa]}$), and respiratory costs $R^{[\kappa]}$. The intake of food from predation is described by the term $E^{[\kappa]} I^{[\kappa]}$, where $I^{[\kappa]}$ is the gross carbon mass intake and $E^{[\kappa]}$ is the assimilation efficiency. The redistribution of mass during reproduction (from mature individuals to eggs) is covered by $H^{[\kappa]}_{\text{repr}}$.

For very early life stages that cannot forage on their own, only background mortality, reproduction, and losses due to predation have to be considered.

Fishing

Extraction of individuals due to fishing is described by an instantaneous fishing mortality rate (cf. Chapter 3) which can vary depending on spatial location and fish size. For example, in our evaluation of the Sprat Model in Chapters 15 and 16, $H^{[\kappa]}_{\text{fishing}}$ is given by

$$H^{[\kappa]}_{\text{fishing}}(t, x, y, r) = \begin{cases} F^{[\kappa]}(t) & \text{if } M^{[\kappa]} \geqslant \frac{1}{2} M^{[\kappa]}_{\text{mature}} \\ 0 & \text{otherwise,} \end{cases} \tag{11.60}$$

where $F^{[\kappa]}(t)$ is the observed fishing mortality for species κ.

Background Mortality

To account for losses of fish due to effects that are not explicitly considered in our model (such as predation by birds and marine mammals), we introduce

the background mortality

$$H_{\text{backgr}}^{[\kappa]} = \begin{cases} \varepsilon_B^{[\kappa]} m^{[\kappa]} & \text{if } M^{[\kappa]} > \frac{1}{4} M_{\text{mature}}^{[\kappa]} \\ \varepsilon_B^{[\kappa]} m^{[\kappa]} - \zeta_B^{[\kappa]} \min(0, \Delta T(t)) & \text{otherwise.} \end{cases} \tag{11.61}$$

For all individual sizes, we apply a quadratic instantaneous death term with mortality rate $\varepsilon_D^{[\kappa]}$. Since earlier life stages are especially vulnerable to fluctuations in temperature, we include an additional linear death term for smaller individuals that depends on the deviation $\Delta T(t)$ from the average habitat temperature (see Houde 2009).

Metabolic Costs

Respiratory costs are given by

$$R^{[\kappa]} = \frac{m^{[\kappa]}}{r} \left(R_S^{[\kappa]} + R_A^{[\kappa]} \right) = u^{[\kappa]} \left(R_S^{[\kappa]} + R_A^{[\kappa]} \right), \tag{11.62}$$

where $R_S^{[\kappa]}$ is the Standard Metabolic Rate (SMR) of an individual fish and $R_A^{[\kappa]}$ represents the additional respiratory costs due to swimming (net swimming costs). $R_A^{[\kappa]}$ corresponds to the Active Metabolic Rate (AMR) minus the SMR.

According to Clarke and Johnston (1999), SMR is fitted well for many teleostei by

$$SMR = \frac{1}{5.43} M^{0.8} \tag{11.63}$$

with M being the wet mass of an individual in g and SMR in mmol O_2 h^{-1}.

In order to convert oxygen consumption to carbon losses, we make use of the so-called respiratory quotient RQ, which describes the ratio of exhaled CO_2 volume to inhaled O_2 volume:

$$RQ = \frac{1 \text{ mol } CO_2}{1 \text{ mol } O_2} \tag{11.64}$$

While the value of RQ depends on the diet, Videler (1993) determines

$$RQ = 0.96 \tag{11.65}$$

to be a good average value for fish. Therefore, it holds that

$$1 \text{ mol } O_2 = \frac{1}{0.96} \text{ mol } CO_2, \tag{11.66}$$

which implies:

$$\begin{aligned} 1 \text{ mmol } O_2 \text{ h}^{-1} &= \frac{1}{0.96} \text{ mmol } CO_2 \text{ h}^{-1} && (11.67)\\ &= \frac{1}{0.96} \text{ mmol C h}^{-1} && (11.68)\\ &= \frac{12}{0.96} \text{ mg C h}^{-1} && (11.69)\\ &= \frac{12}{0.96 \cdot 10^6 \cdot 60^2} \text{ kg C s}^{-1} && (11.70) \end{aligned}$$

By converting to appropriate units and compensating for temperature changes using a Q_{10} temperature coefficient, we define the SMR of an individual fish in our model as

$$R_S^{[\kappa]} = \left(Q_{10}^{\text{SMR}}\right)^{\frac{\Delta T(t)}{10}} \frac{12}{5.43 \cdot 0.96 \cdot 10^6 \cdot 60^2} \left(10^3 M^{[\kappa]}\right)^{0.8}. \tag{11.71}$$

Videler (1993) deems $Q_{10}^{\text{SMR}} = 2$ to be appropriate in the context of resting metabolic rates of fish.

Regarding the respiratory net costs of swimming, Boisclair and Tang (1993) as well as Ohlberger et al. (2005) propose a model of the form

$$AMR = SMR + aM^b v^c, \tag{11.72}$$

where M is wet mass in g, v is swimming speed in cm s^{-1}, and AMR is in mg O_2 h^{-1}. A parametrization of Equation 11.72 for steady swimming in many fish is $a = 10^{-2.43}$, $b = 0.8$, and $c = 1.21$ (Boisclair and Tang 1993). For determining v, we only have to consider the velocity due to active movement

$$V_{\text{active}}^{[\kappa]} = V_{\text{reactive}}^{[\kappa]} + V_{\text{predictive}}^{[\kappa]}. \tag{11.73}$$

With regard to unit conversion, we use Equation 11.66 again to derive:

$$1 \text{ mg } O_2 \text{ h}^{-1} = 1 \text{ mg } O_2 \text{ h}^{-1} \cdot \left(\frac{1}{32} \frac{\text{mol } O_2}{\text{g } O_2} \right) \tag{11.74}$$

$$= \frac{1}{32} \text{ mmol } O_2 \text{ h}^{-1} \tag{11.75}$$

$$= \frac{1}{0.96 \cdot 32} \text{ mmol } CO_2 \text{ h}^{-1} \tag{11.76}$$

$$= \frac{12}{0.96 \cdot 32 \cdot 10^6 \cdot 60^2} \text{ kg C s}^{-1} \tag{11.77}$$

Applying appropriate unit conversions, we can define the net swimming costs per individual as

$$R_A^{[\kappa]} = \frac{12 \cdot 10^{-2.43}}{0.96 \cdot 32 \cdot 10^6 \cdot 60^2} \left(10^3 M^{[\kappa]} \right)^{0.8} \left(10^2 \| V_{\text{active}}^{[\kappa]} \| \right)^{1.21}. \tag{11.78}$$

Reproduction

During the spawning season $S^{[\kappa]} \subset [0,1]$, mass is transferred from mature female fish to (fertilized) eggs. We assume a constant 50 % : 50 % sex ratio of females to males in the population. The net wet mass fecundity $\Phi^{[\kappa]}$ describes how many fertilized eggs per kg wet mass a mature female produces during the spawning season.

In order to describe the mass transfer process due to reproduction, we define the carbon mass at maturity

$$r_{\text{mature}}^{[\kappa]} = c_{\text{C/dm}}^{[\kappa]} c_{\text{d/wm}}^{[\kappa]} M_{\text{mature}}^{[\kappa]}, \tag{11.79}$$

the carbon mass of an egg

$$r_{\text{egg}}^{[\kappa]} = c_{\text{C/dm}}^{[\kappa]} w_{\text{egg}}^{[\kappa]}, \tag{11.80}$$

and the duration of the spawning season of species κ in seconds

$$\theta_s^{[\kappa]} = 365 \cdot 24 \cdot 60^2 \cdot \int_0^1 \mathbb{1}_{S^{[\kappa]}}(\phi) \, d\phi, \tag{11.81}$$

where $\mathbb{1}_{S^{[\kappa]}}$ is the characteristic function on $S^{[\kappa]}$. This allows us to define the biomass spawning rate

$$B_s^{[\kappa]} = \begin{cases} \frac{1}{2} \frac{\Phi^{[\kappa]} r_{\text{egg}}^{[\kappa]}}{c_{\text{C/dm}}^{[\kappa]} c_{\text{d/wm}}^{[\kappa]} \theta_s^{[\kappa]}} m^{[\kappa]} & \text{if } \theta(t) \in S^{[\kappa]} \wedge r \geqslant r_{\text{mature}}^{[\kappa]} \\ 0 & \text{otherwise,} \end{cases} \tag{11.82}$$

which describes the amount of egg biomass that is spawned at each moment of the spawning season (spawning happens at a constant rate during the whole season).

In order to reintroduce the egg biomass into the distribution m at an appropriate "place" in the size dimension, we define an insertion distribution $\psi^{[\kappa]}(r)$ that only has a small support which is centered around $r_{\text{egg}}^{[\kappa]}$. We require $\psi^{[\kappa]} \colon [r_{\min}, r_{\max}] \to \mathbb{R}_{\geqslant 0}$ to be continuous and to fulfill

$$\arg\max_{\phi \in [r_{\min}, r_{\max}]} \left(\psi^{[\kappa]}(\phi) \right) = r_{\text{egg}}^{[\kappa]} \tag{11.83}$$

$$\wedge \int_{r_{\min}}^{r_{\max}} \psi^{[\kappa]}(\phi)\, d\phi = 1 \tag{11.84}$$

$$\wedge\ \psi^{[\kappa]}(r) = 0 \text{ for all } r \geqslant r_{\text{mature}}^{[\kappa]}. \tag{11.85}$$

The insertion distribution $\psi^{[\kappa]}(r)$ allows us to describe the redistribution of egg mass as

$$H_{\text{repr}}^{[\kappa]}(t,x,y,r) = \psi^{[\kappa]}(r) \int_{r_{\text{mature}}^{[\kappa]}}^{r_{\max}} B_s^{[\kappa]}(t,x,y,\phi)\, d\phi - B_s^{[\kappa]}(t,x,y,r). \tag{11.86}$$

In the following lemma, we show that $H_{\text{repr}}^{[\kappa]}$ does *not* interfere with the mass conservation property of the Sprat Model by introducing spurious biomass sources or sinks.

Lemma 11.1. *The term $H_{\text{repr}}^{[\kappa]}$, as defined in Equation 11.86, conserves biomass as it only redistributes mass (the net effect on the total biomass is zero):*

$$\int_\Omega H_{\text{repr}}^{[\kappa]}(t,x,y,r)\, d(x,y,r) = 0 \quad \text{for all } t \in [0, t_{\max}] \tag{11.87}$$

Proof. Let $t \in [0, t_{\max}]$ and $(x, y) \in \Omega_S$. Then, it holds:

$$\int_{r_{\min}}^{r_{\max}} H_{\text{repr}}^{[\kappa]}(t,x,y,r)\, dr \tag{11.88}$$

$$= \int_{r_{\min}}^{r_{\max}} \psi^{[\kappa]}(r) \int_{r_{\text{mature}}^{[\kappa]}}^{r_{\max}} B_s^{[\kappa]}(t,x,y,\phi)\, d\phi - B_s^{[\kappa]}(t,x,y,r)\, dr \tag{11.89}$$

$$= \int_{r_{\min}}^{r_{\max}} B_s^{[\kappa]}(t,x,y,\phi)\, d\phi \int_{r_{\min}}^{r_{\max}} \psi^{[\kappa]}(r)\, dr - \int_{r_{\min}}^{r_{\max}} B_s^{[\kappa]}(t,x,y,r)\, dr \tag{11.90}$$

$$= \int_{r_{\min}}^{r_{\max}} B_s^{[\kappa]}(t,x,y,\phi)\, d\phi - \int_{r_{\min}}^{r_{\max}} B_s^{[\kappa]}(t,x,y,r)\, dr = 0 \qquad \square$$

Predation

In this section, we describe the processes related to predation that determine the intake of biomass ($I^{[\kappa]}$), biomass losses due to predation ($L^{[\kappa]}$), and the grazing of zooplankton by fish (G). The latter links the fish model with the NPZ model as described in Equation 11.2.

The amount of prey of species κ that is available to a predator is given by

$$S_p^{[\kappa]}(t,x,y,r) = \int_{r_{\min}}^{r/\tau} m^{[\kappa]}(t,x,y,\phi)\, d\phi. \tag{11.91}$$

With it, we define the auxiliary term

$$F_m^{[\kappa]}(t,x,y,r) = \eta^{[\kappa]} \left(Q_{10}^P\right)^{\frac{\Delta T(t)}{10}} c_{\text{C/dm}}^{[\kappa]} c_{\text{d/wm}}^{[\kappa]} \left(M^{[\kappa]}\right)^{\alpha} u^{[\kappa]} \tag{11.92}$$

$$\cdot \frac{1}{F_0^{[\kappa]} + \sum_{i \neq \kappa} S_p^{[i]}} \tag{11.93}$$

to describe biomass intake as

$$I^{[\kappa]} = G^{[\kappa]} + \begin{cases} F_m^{[\kappa]} \sum_{i \neq \kappa} S_p^{[i]} & \text{if } r \geqslant C_{\text{C/dm}}^{[\kappa]} w_{\text{forage}}^{[\kappa]} \\ 0 & \text{otherwise.} \end{cases} \tag{11.94}$$

Biomass intake of fish that forage on their own consists of zooplankton grazing $G^{[\kappa]}$ (see below) and predation on fish from other species (as men-

tioned above, the model does not include cannibalism). Predation on fish is modeled via a saturation response with the half saturation constant $F_0^{[\kappa]}$. $\eta^{[\kappa]}$ is the constant predation rate of species κ that is adjusted for temperature fluctuations by the temperature coefficient $Q_{10}^P = 2$. The allometric exponent α scales the influence of body mass on the predation rate and can be estimated to be about 0.74 for pelagic predator fish (Ware 1978). If $\alpha = 1$ held, a fish twice the size of another could forage twice as much.

The biomass losses due to predation are given by

$$L^{[\kappa]}(t,x,y,r) = m^{[\kappa]} \sum_{i \neq \kappa} \int_{\min(\max(\tau r, r_{\text{forage}}^{[i]}), r_{\max})}^{r_{\max}} F_m^{[i]}(t,x,y,\phi)\, d\phi \tag{11.95}$$

with $r_{\text{forage}}^{[i]} = C_{\text{C/dm}}^{[i]} w_{\text{forage}}^{[i]}$. The complex expression for the lower integral boundary ensures that only predation by fish that are large enough (τr) and can forage on their own ($r_{\text{forage}}^{[i]}$) is considered. Additionally, the lower integral boundary must not exceed the overall maximum fish size $r_{\max}$.

Before defining the zooplankton grazing term, we prove in the following lemma that our formulation of fish intake and losses due to predation does not interfere with the mass conservation property of our model.

Lemma 11.2. *It holds that*

$$\int_{r_{\min}}^{r_{\max}} \sum_{\kappa=1}^{n} I^{[\kappa]} - G^{[\kappa]}\, dr = \int_{r_{\min}}^{r_{\max}} \sum_{\kappa=1}^{n} L^{[\kappa]}\, dr. \tag{11.96}$$

Proof. The statement to be proven is implied by the stronger statement that the fish intake of predator species κ from prey species j equals the losses of species j due to predation by species κ. This is described by

$$\int_{r_{\text{forage}}^{[\kappa]}}^{r_{\max}} F_m^{[\kappa]}(t,x,y,r)\, S_p^{[j]}(t,x,y,r)\, dr \tag{11.97}$$

$$= \int_{r_{\min}}^{r_{\max}} m^{[j]}(t,x,y,r) \int_{\min(\max(\tau r, r_{\text{forage}}^{[\kappa]}), r_{\max})}^{r_{\max}} F_m^{[\kappa]}(t,x,y,\phi)\, d\phi\, dr, \tag{11.98}$$

which is equivalent to

$$\int_{r_{\text{forage}}^{[\kappa]}}^{r_{\max}} F_m^{[\kappa]}(t,x,y,r) \int_{r_{\min}}^{r/\tau} m^{[j]}(t,x,y,\phi)\; d\phi\; dr \tag{11.99}$$

$$= \int_{r_{\min}}^{r_{\max}/\tau} m^{[j]}(t,x,y,r) \int_{\max(\tau r, r_{\text{forage}}^{[\kappa]})}^{r_{\max}} F_m^{[\kappa]}(t,x,y,\phi)\; d\phi\; dr \tag{11.100}$$

$$\iff \int_{r_{\text{forage}}^{[\kappa]}}^{r_{\max}} \int_{r_{\min}}^{r/\tau} F_m^{[\kappa]}(t,x,y,r)\, m^{[j]}(t,x,y,\phi)\; d\phi\; dr \tag{11.101}$$

$$= \int_{r_{\min}}^{r_{\max}/\tau} \int_{\max(\tau\phi, r_{\text{forage}}^{[\kappa]})}^{r_{\max}} F_m^{[\kappa]}(t,x,y,r)\, m^{[j]}(t,x,y,\phi)\; dr\; d\phi. \tag{11.102}$$

With the sets

$$U := \{(r,\phi) \mid r \in [r_{\text{forage}}^{[\kappa]}, r_{\max}] \wedge \phi \in [r_{\min}, r/\tau]\} \text{ and} \tag{11.103}$$

$$W := \{(r,\phi) \mid \phi \in [r_{\min}, r_{\max}/\tau] \wedge r \in [\max(\tau\phi, r_{\text{forage}}^{[\kappa]}), r_{\max}]\}, \tag{11.104}$$

our proposition is equivalent to

$$\int_U F_m^{[\kappa]}(r)\, m^{[j]}(\phi)\; d(r,\phi) = \int_W F_m^{[\kappa]}(r)\, m^{[j]}(\phi)\; d(r,\phi). \tag{11.105}$$

To prove this statement, we only have to show that $U = W$:

1. Let $u = (r,\phi) \in U$. Then, obviously, $\phi \in [r_{\min}, r_{\max}/\tau]$ holds. Because of

$$\phi \leqslant \frac{r}{\tau} \quad \Leftrightarrow \quad r \geqslant \tau\phi, \tag{11.106}$$

it is also $r \in [\max(\tau\phi, r_{\text{forage}}^{[\kappa]}), r_{\max}]$. Therefore, $u \in W$.

2. Now let $w = (r,\phi) \in W$. Then, it is

$$r \geqslant \tau\phi \quad \wedge \quad r \geqslant r_{\text{forage}}^{[\kappa]}. \tag{11.107}$$

This implies, on the one hand, that $r \in [r^{[\kappa]}_{\text{forage}}, r_{\max}]$ and, on the other hand, that

$$\phi \leqslant \frac{r}{\tau}, \tag{11.108}$$

which means $\phi \in [r_{\min}, {}^{r}/_{\tau}]$. Therefore, $w \in U$.

Since $U \subseteq W$ and $W \subseteq U$, it must be $U = W$. □

Regarding zooplankton grazing, we define an auxiliary term analogous to $F^{[\kappa]}_m$:

$$Z^{[\kappa]}_m(t,x,y,r) = \mu^{[\kappa]} \left(Q^P_{10}\right)^{\frac{\Delta T(t)}{10}} c^{[\kappa]}_{\text{C/dm}} c^{[\kappa]}_{\text{d/wm}} \left(M^{[\kappa]}\right)^{\alpha} u^{[\kappa]}, \tag{11.109}$$

where $\mu^{[\kappa]}$ is the zooplankton grazing rate of species κ. With it, we characterize zooplankton grazing by a saturating response

$$G^{[\kappa]} = \begin{cases} Z^{[\kappa]}_m \cdot K_Z(t,x,y) \cdot \frac{\rho Z(t,x,y)}{Z^{[\kappa]}_0 + \rho Z(t,x,y)} & \text{if } r \geqslant C^{[\kappa]}_{\text{C/dm}} w^{[\kappa]}_{\text{forage}} \wedge \\ & \quad M^{[\kappa]} \leqslant M^{[\kappa]}_{\text{plankton}} \\ 0 & \text{otherwise} \end{cases} \tag{11.110}$$

with K_Z being a limiter factor and $Z^{[\kappa]}_0$ the half saturation constant. The limiter factor K_Z ensures that zooplankton cannot be depleted at arbitrarily high rates and is defined as

$$K_Z(t,x,y) = \min\left(1, \frac{\rho \cdot j_f(Z(t,x,y)) \cdot Z(t,x,y)}{k(t,x,y)}\right), \tag{11.111}$$

where j_f is from Equation 11.9 and k is the (only theoretical) total unlimited zooplankton consumption by all species:

$$k(t,x,y) = \sum_{\kappa=1}^{n} \int_{r_{\min}}^{r_{\max}} G^{[\kappa]}_{UL}(t,x,y,\phi)\, d\phi \tag{11.112}$$

$$G_{UL}^{[\kappa]}(t,x,y,r) = \begin{cases} Z_m^{[\kappa]} \cdot \frac{\rho Z(t,x,y)}{Z_0^{[\kappa]}+\rho Z(t,x,y)} & \text{if } r \geqslant C_{\mathrm{C/dm}}^{[\kappa]} w_{\mathrm{forage}}^{[\kappa]} \wedge \\ & \quad M^{[\kappa]} \leqslant M_{\mathrm{plankton}}^{[\kappa]} \\ 0 & \text{otherwise} \end{cases} \tag{11.113}$$

A problem that could occur with the formulation of K_Z in Equation 11.111 is that—via Equations 11.112 and 11.113—$Z = 0$ implies $k = 0$. In practice, however, local zooplankton levels should never reach zero since we limit zooplankton uptake. Nonetheless, to be absolutely sure, the threat of dividing by zero can be eliminated by setting $K_Z = 0$ for very small zooplankton concentrations.

In order to feed back the zooplankton consumption to the NPZ model, we define the total zooplankton consumption G, which is used in Equation 11.2:

$$G(t,x,y) = \frac{1}{\rho} \sum_{\kappa=1}^{n} \int_{r_{\min}}^{r_{\max}} G^{[\kappa]}(t,x,y,\phi)\, d\phi \tag{11.114}$$

11.2.6 Growth

The growth speed in kg s^{-1} is given by

$$g^{[\kappa]} = \begin{cases} \left(Q_{10}^{G}\right)^{\frac{\Delta T(t)}{10}} \frac{1}{u^{[\kappa]}} \max\left(0, E^{[\kappa]} I^{[\kappa]} - R^{[\kappa]}\right) & \text{if } \frac{w_{\mathrm{forage}}^{[\kappa]}}{c_{\mathrm{d/wm}}^{[\kappa]}} \leqslant M^{[\kappa]} < M_{\max}^{[\kappa]} \\ \left(Q_{10}^{L}\right)^{\frac{\Delta T(t)}{10}} \frac{c_{\mathrm{C/dm}}^{[\kappa]} \left(w_{\mathrm{forage}}^{[\kappa]} - w_{\mathrm{egg}}^{[\kappa]}\right)}{\theta_{\mathrm{yolk}}^{[\kappa]}} & \text{if } M^{[\kappa]} < \frac{w_{\mathrm{forage}}^{[\kappa]}}{c_{\mathrm{d/wm}}^{[\kappa]}} \\ 0 & \text{otherwise.} \end{cases} \tag{11.115}$$

We distinguish three cases:

1. Fish that can forage on their own but have not yet reached their maximum weight.
2. Fish that cannot yet forage on their own.
3. Fish that have reached the maximum weight of their species.

In the first two cases, we compensate for temperature fluctuations by using Q_{10} temperature coefficients. For larvae/eggs, we employ $Q_{10}^{L} = 3$ and for further-developed individuals, $Q_{10}^{G} = 2$ (see Houde 2009).

In the first case of individuals that forage on their own, $g^{[\kappa]}$ describes the positive net biomass assimilation rate of all individuals at the corresponding coordinates divided by the number of individuals. Thereby, it expresses the biomass accumulation rate of a single individual. Since mass is the only attribute to track the developmental stage of individuals in our model, we can only allow for positive growth rates because, otherwise, adult fish could, for example, become larvae again (i. e., develop backwards in time).

In the second case of eggs/larvae that cannot (yet) forage on their own, we apply a constant growth speed (aside from the temperature dependency) to let them grow from $w_{\text{egg}}^{[\kappa]}$ to $w_{\text{forage}}^{[\kappa]}$ in time $\theta_{\text{yolk}}^{[\kappa]}$. A constant growth speed is assumed to be a reasonably good approximation of the exponential growth that fish experience during their very early life stages, in which they (exclusively) feed on their yolk sac.

One could argue, however, that eggs and larvae cannot accumulate mass as long as they do not take it up from their environment but from their yolk sac (the weight of which is already included in $w_{\text{egg}}^{[\kappa]}$). Hence, it should hold that $w_{\text{egg}}^{[\kappa]} = w_{\text{forage}}^{[\kappa]}$. Since in our model we have to represent the aging process of eggs/larvae via changes in their mass (the model does not have an age dimension), we solve this problem pragmatically by letting $w_{\text{egg}}^{[\kappa]}$ represent the lower end of the egg weight spectrum and $w_{\text{forage}}^{[\kappa]}$ the upper end.

By transporting a constant amount of mass from $w_{\text{egg}}^{[\kappa]}$ to $w_{\text{forage}}^{[\kappa]}$, we also introduce a hidden death term since the same mass at $w_{\text{egg}}^{[\kappa]}$ represents more individuals than at $w_{\text{forage}}^{[\kappa]} > w_{\text{egg}}^{[\kappa]}$. This, however, does not have an effect on the mass conservation property of the model.

Once the fish have reached their maximum mass in the third case of the definition of $g^{[\kappa]}$, growth stops.

Chapter 12

An Explicit Flux-Corrected Transport FEM Scheme

We apply a Finite Element Method (FEM) solver to numerically approximate the solution of the Sprat Marine Ecosystem Model, which is given by the PDE system in Equation 11.37 in the previous chapter. In contrast to other numerical methods for PDEs, such as finite differences, the method of finite elements has the advantage of guaranteeing good approximation results even without high regularity of the solution and it allows to use unstructured meshes (see Chapter 4). However, it is well-known that the standard Galerkin approach is unstable for (non-linear) advection equations and introduces spurious oscillations into the solution (Kuzmin and Turek 2002). These oscillations are the result of undershoots in the numerical solution (i. e., the discrete solution becomes negative although the analytical solution can be proven to be non-negative) that appear because too much of the transported quantity is advected away from a single point in one discrete time step. Since the Galerkin discretization conserves mass, these undershoots have to be compensated elsewhere, which leads to overshoots at other locations (the problem is visualized in Figure 12.1). The oscillations do not only diminish the accuracy of the numerical solution but also lead to biologically unrealistic results since fish biomass, in reality, cannot assume negative values.

A common approach to tackle the problem of spurious oscillations is to introduce a certain amount of artificial diffusion into the solution. This prevents undershoots, as depicted in Figure 12.2, but at the same time decreases the accuracy of the solution dramatically (e. g., the values of minima and maxima are not retained; minima increase and maxima decrease).

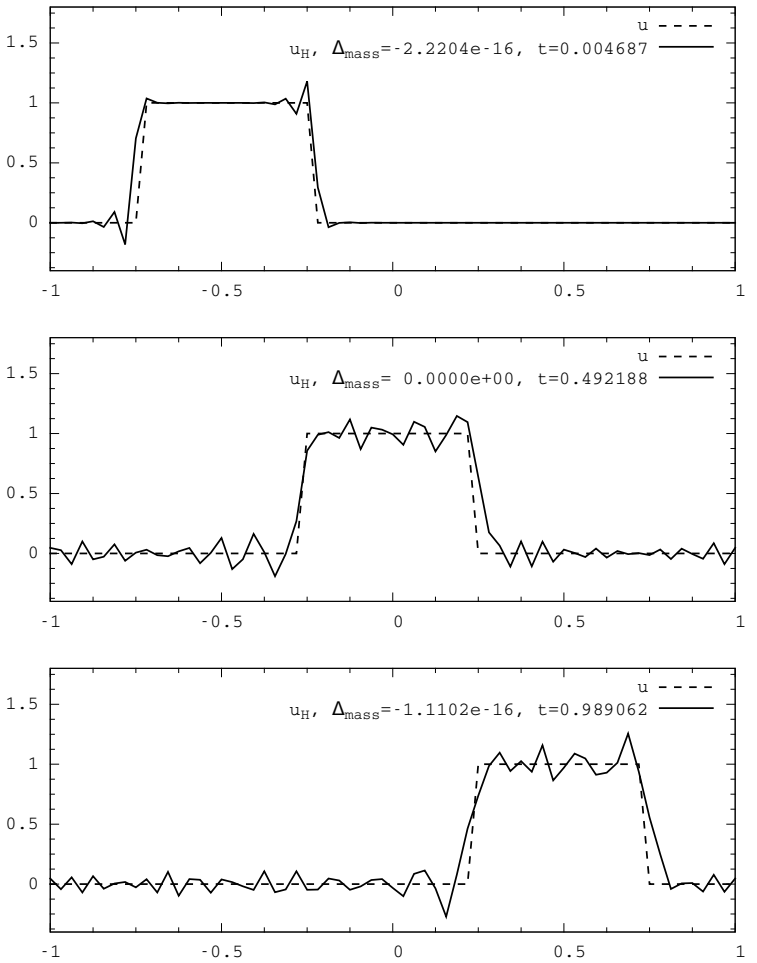


Figure 12.1. A profile is advected with unit velocity on the domain $[-1, 1]$ with periodic boundary conditions. u is the analytical solution and u_H is calculated using the Galerkin method with linear elements to discretize space and the three-step Adams-Bashforth method to integrate in time. Spurious oscillations appear after only a few time steps. Note that mass is conserved up to machine precision (Δ_{mass}).

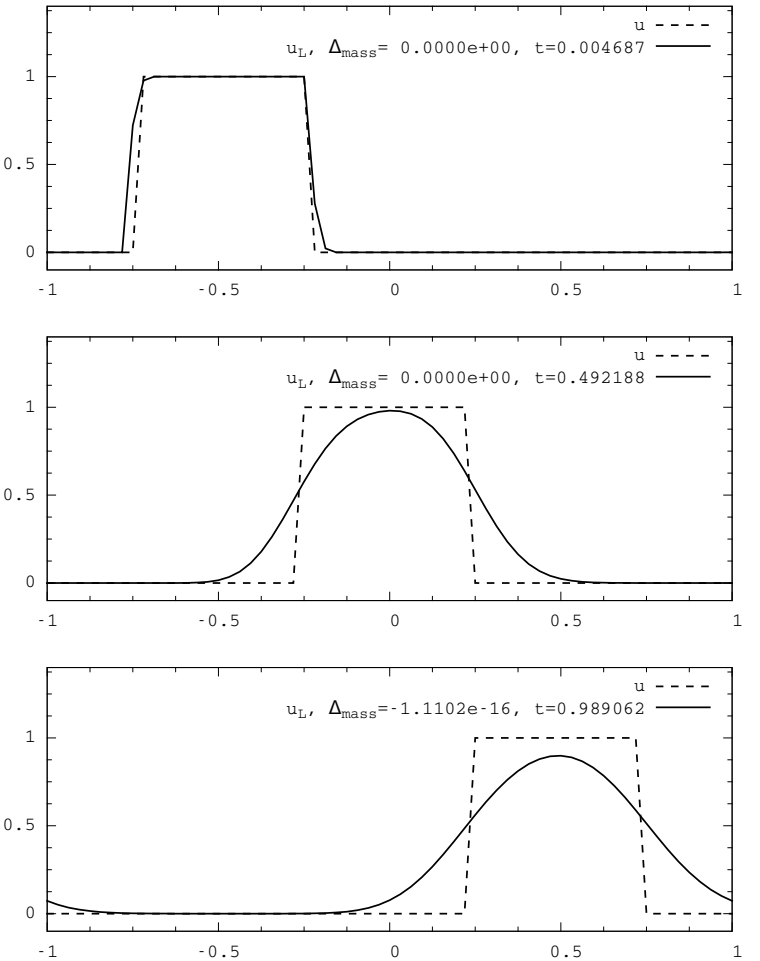


Figure 12.2. A profile is advected with unit velocity. u is the analytical solution and u_L is calculated using the explicit Euler method with artificial diffusion to integrate in time.

To circumvent both the spurious oscillations and the high diffusivity, Boris and Book (1973) introduced the concept of Flux-Corrected Transport (FCT), which was applied to the FEM by Löhner et al. (1987). The fundamental idea of the FCT FEM is to calculate in every time step both a high-order solution (possibly containing ripples) and a low-order solution (free from ripples but overly diffusive) and to blend these two together. This blending process is based on a reformulation of the finite element approach in terms of fluxes that are exchanged between nodes and that can be limited to prevent undershoots in the discrete solution. This limiting procedure ensures that the solution is high-order accurate in regions where no oscillations occur and prevents ripples by adding diffusion in areas where undershoots would develop.

In this chapter, we introduce an FCT FEM scheme for advection equations that uses explicit multi-step methods for integrating the high-order solution in time. The method is based on the publications of Kuzmin et al. (2003), Kuzmin and Turek (2002), and Kuzmin (2010) but extends their work, which focuses on implicit time stepping, to explicit multi-step methods.

The rest of this chapter is structured as follows: in Section 12.1, we present a model problem that is used to illustrate our FCT FEM scheme in Section 12.2. Section 12.3 gives a summary of the algorithmic steps that are involved in applying our scheme. This section also discusses the runtime and storage requirements of the algorithm. Results from numerical experiments with our FCT FEM method are analyzed in Section 12.4.

12.1 A Model Problem and the Group Finite Element Formulation

In order to describe our FCT FEM scheme with explicit time stepping, we introduce a two-dimensional model problem. On a polygon domain $\Omega \subset \mathbb{R}^2$, we seek to determine

$$u\colon [0, t_{\max}] \times \Omega \to \mathbb{R},\ (t, x, y) \mapsto u(t, x, y) \tag{12.1}$$

that is given as the solution of the non-linear advection equation

$$\frac{\partial}{\partial t}u + \frac{\partial}{\partial x}\left(q_x(u)\,u\right) + \frac{\partial}{\partial y}\left(q_y(u)\,u\right) = s, \tag{12.2}$$

where q_x and q_y are advection velocities and s is a source term. The solution u is required to meet the initial conditions

$$u(0, x, y) = u_0(x, y). \tag{12.3}$$

We treat all boundaries of the spatial domain as outflows and, hence, do not prescribe any boundary conditions.

Note that it is straightforward to generalize the model problem given by Equation 12.2 to the PDE system of the Sprat Model (Equation 11.37) as there is no mathematical difference between advection in a space and in a size dimension.

12.1.1 The Group Finite Element Formulation

If a finite element approach is applied to discretize the non-linear advection equation given by Equation 12.2, one encounters the problem that the transport velocities q_x and q_y depend on the unknown u. By applying the chain rule, it becomes clear that this dependency introduces more complicated derivatives which involve differentiating the transport velocities with respect to u. This is inconvenient as the advection velocities are often not given analytically but only discretely (e. g., the reactive movement component in the Sprat Model).

A standard approach to circumvent this problem is the *group finite element approach* by Fletcher (1983). In this approach, not only the unknown u is discretized in space but whole groups of variables are (hence the name). This means that we do not only approximate

$$u(t, x, y) \approx \sum_j u_j(t)\,\varphi_j(x, y) \tag{12.4}$$

but also, for example,

$$q_x(u(t,x,y))\,u(t,x,y) \approx \sum_j q_{x,j}(u_j(t))\,\varphi_j(x,y). \tag{12.5}$$

Since $q_{x,j}(u_j(t))$ in Equation 12.5 does not depend on any spatial coordinate, it is not affected by the spatial derivatives in Equation 12.2. Fletcher (1983) showed that this approach does not significantly influence the approximation accuracy while it tremendously simplifies the finite element discretization.

12.2 Flux-Corrected Transport With Explicit Multi-Step Methods

In order to describe our FCT FEM scheme, we first employ the standard FEM discretization in space to transform the model problem from Equation 12.2 into a semi-discrete form (Section 12.2.1). We then explain how to partition the discrete transport operators into element matrices which can be used to reconstruct element contributions to individual nodes that correspond to numerical (mass) fluxes and can later on be limited (Section 12.2.2). Based on the element partitions of our transport operators, we introduce a diffusive low-order scheme that is guaranteed to be free from oscillations (Section 12.2.3) and a high-order scheme using an explicit Adams-Bashforth multi-step method (Section 12.2.4). The high-order scheme can be expressed in terms of the low-order scheme and a sum of *anti-diffusive* element contributions/fluxes. This allows us to locally limit the anti-diffusive fluxes (which would otherwise reconstruct the high-order method everywhere) using a flux limiter (Section 12.2.5). In this way, we receive a hybrid solution that is high-order accurate everywhere except for areas in which oscillations would form. We show that our scheme conserves mass and that the resulting solution is non-negative (which implies that it is free from oscillations, as explained in the introduction of this chapter).

12.2.1 Spatial Discretization

As detailed in Chapter 4, we transform our model problem from Equation 12.2 into a variational form using the test function space $V = H^1(\Omega)$:

$$\int_\Omega \varphi \left(\frac{\partial}{\partial t} u + \frac{\partial}{\partial x} \left(q_x(u)\, u \right) + \frac{\partial}{\partial y} \left(q_y(u)\, u \right) - s \right) d(x,y) = 0 \quad \forall\, \varphi \in V \tag{12.6}$$

Equation 12.6 is discretized by replacing V with a finite-dimensional subspace $V_h \subseteq V$ with a basis $(\varphi_j)_{j=1,\dots,n}$, $n = \dim(V_h)$. We require that this basis fulfills the unit sum property

$$\sum_{j=1}^{n} \varphi_j \equiv 1 \tag{12.7}$$

and that

$$\int_\Omega \varphi_i \, d(x,y) > 0 \quad \forall\, i = 1, \dots, n. \tag{12.8}$$

These properties are, for example, fulfilled by tensor products of one-dimensional hat functions on quadrilateral elements.

In addition to restricting our test function space, we also employ the group finite element approach as described above to obtain

$$\int_\Omega \varphi_i \left(\frac{\partial}{\partial t} u_h + \frac{\partial}{\partial x} q_{x,h} + \frac{\partial}{\partial y} q_{y,h} - s_h \right) d(x,y) = 0 \quad \forall\, i = 1, \dots, n. \tag{12.9}$$

The discrete functions u_h, $q_{x,h}$, $q_{y,h}$, and s_h are given by the vectors $\mathbf{u}(t) = (u_j(t))_{j=1,\dots,n}$, $\mathbf{q}_x(\mathbf{u}) = (q_{x,j}(u_j))_j$, $\mathbf{q}_y(\mathbf{u}) = (q_{y,j}(u_j))_j$, and $\mathbf{s}(t,\mathbf{u}) = (s_j(t,\mathbf{u}))_j$ via:

$$u_h(t,x,y) = \sum_{j=1}^{n} u_j(t)\, \varphi_j(x,y) \tag{12.10}$$

$$q_x(u)\, u(t,x,y) \approx q_{x,h}(t,x,y,\mathbf{u}) = \sum_{j=1}^{n} q_{x,j}(u_j(t))\, \varphi_j(x,y) \tag{12.11}$$

$$q_y(u)\, u(t,x,y) \approx q_{y,h}(t,x,y,\mathbf{u}) = \sum_{j=1}^{n} q_{y,j}(u_j(t))\, \varphi_j(x,y) \tag{12.12}$$

$$s(t,x,y,u) \approx s_h(t,x,y,\mathbf{u}) = \sum_{j=1}^{n} s_j(t,\mathbf{u}(t))\,\varphi_j(x,y), \text{ where} \tag{12.13}$$

$$u_j(t) = u_i(t) \;\Rightarrow\; q_{x,j}(u_j(t)) = q_{x,i}(u_i(t)) \;\wedge\; q_{y,j}(u_j(t)) = q_{y,i}(u_i(t)). \tag{12.14}$$

With the definition of the discrete functions in Equations 12.10 to 12.13 and by applying Green's formula (Equation 4.5 in Chapter 4) to Equation 12.9, we obtain

$$\sum_{j=1}^{n} \frac{\partial}{\partial t} u_j \int_\Omega \varphi_i \varphi_j \, d(x,y) \tag{12.15}$$

$$= \sum_{j=1}^{n} q_{x,j} \left(\int_\Omega \varphi_j \frac{\partial}{\partial x} \varphi_i \, d(x,y) - \int_{\partial\Omega} \overrightarrow{n_x} \varphi_i \varphi_j \, d\sigma \right) \tag{12.16}$$

$$+ \sum_{j=1}^{n} q_{y,j} \left(\int_\Omega \varphi_j \frac{\partial}{\partial y} \varphi_i \, d(x,y) - \int_{\partial\Omega} \overrightarrow{n_y} \varphi_i \varphi_j \, d\sigma \right) \tag{12.17}$$

$$+ \sum_{j=1}^{n} s_j \int_\Omega \varphi_i \varphi_j \, d(x,y) \quad \forall\, i = 1, \ldots, n, \tag{12.18}$$

where $\overrightarrow{n_x}$ and $\overrightarrow{n_y}$ are the x and y components of the outward-pointing normal vector of the domain boundary $\partial\Omega$. This equation can be written in matrix form as

$$\mathbf{M}_C \frac{\partial}{\partial t} \mathbf{u} = (\mathbf{C}_x - \mathbf{C}_{B,x})\,\mathbf{q}_x + (\mathbf{C}_y - \mathbf{C}_{B,y})\,\mathbf{q}_y + \mathbf{M}_C \mathbf{s} \tag{12.19}$$

by defining the consistent mass matrix

$$\mathbf{M}_C = \left(\int_\Omega \varphi_i \varphi_j \, d(x,y) \right)_{i,j} \tag{12.20}$$

and the discrete transport operator in x-direction (and analogously in y-direction) with its boundary part

$$\mathbf{C}_x = \left(\int_\Omega \varphi_j \frac{\partial}{\partial x} \varphi_i \, d(x,y) \right)_{i,j} \quad \text{and} \quad \mathbf{C}_{B,x} = \left(\int_{\partial\Omega} \overrightarrow{n_x} \varphi_i \varphi_j \, d\sigma \right)_{i,j}. \tag{12.21}$$

Therefore, the discrete variational form from Equation 12.9 is equivalent to the system of ordinary differential equations

$$\frac{\partial}{\partial t}\mathbf{u} = \mathbf{M}_C^{-1}\left(\left(\mathbf{C}_x - \mathbf{C}_{B,x}\right)\mathbf{q}_x + \left(\mathbf{C}_y - \mathbf{C}_{B,y}\right)\mathbf{q}_y\right) + \mathbf{s}. \tag{12.22}$$

By partially integrating the integrals of the transport operators in Equations 12.16 and 12.17, we were able to express the transport operators as a combination of an interior part ($\mathbf{C}_x$) and a boundary part ($\mathbf{C}_{B,x}$). Since the interior part has zero column sums, as can be seen from

$$\sum_{i=1}^{n}(\mathbf{C}_x)_{ij} = \int_\Omega \varphi_j \frac{\partial}{\partial x}\Bigg(\underbrace{\sum_{i=1}^{n}\varphi_i}_{\equiv 1}\Bigg)\, d(x,y) = 0, \tag{12.23}$$

our semi-discrete approximation conserves mass, as is shown in the following lemma.

Lemma 12.1. *If u_h is given by*

$$\mathbf{M}_C\frac{\partial}{\partial t}\mathbf{u} = \mathbf{C}_x\mathbf{q}_x + \mathbf{C}_y\mathbf{q}_y, \tag{12.24}$$

then it holds that

$$\int_\Omega u_h(t,x,y)\, d(x,y) \equiv \text{const.} \tag{12.25}$$

Proof. Assuming Equation 12.24, it holds that:

$$\begin{aligned}
\frac{\partial}{\partial t}\int_\Omega u_h\, d(x,y) &= \sum_{j=1}^{n}\frac{\partial}{\partial t}u_j\int_\Omega \varphi_j\, d(x,y) && (12.26)\\
&\overset{(12.7)}{=} \sum_{j=1}^{n}\frac{\partial}{\partial t}u_j\int_\Omega \varphi_j\sum_{i=1}^{n}\varphi_i\, d(x,y) && (12.27)\\
&= \sum_{j=1}^{n}\sum_{i=1}^{n}(\mathbf{M}_C)_{ij}\frac{\partial}{\partial t}u_j && (12.28)
\end{aligned}$$

$$= \sum_{j=1}^{n} \left[\underbrace{\sum_{i=1}^{n} (\mathbf{C}_x)_{ij}}_{=0} q_{x,j} + \underbrace{\sum_{i=1}^{n} (\mathbf{C}_y)_{ij}}_{=0} q_{y,j} \right] = 0 \qquad \square$$

12.2.2 Element Partition of the Transport Operator

In order to locally blend together a high-order and a low-order solution, we have to reconstruct element contributions or numerical fluxes in the context of the FEM. This can be done by decomposing the operators of the matrix equation 12.19 into *element matrices*. For this purpose, we assume that the domain Ω is partitioned into a set of elements $\mathcal{T}$ (e. g., a triangulation of the domain). In the following, we demonstrate how to decompose our operators into element matrices using the operator $\mathbf{C}_x$ as a representative. For each $\tau \in \mathcal{T}$, we define the element matrix

$$\mathbf{C}_x^\tau := \left(\int_\tau \varphi_j \frac{\partial}{\partial x} \varphi_i \, d(x,y) \right)_{i,j}. \tag{12.29}$$

It holds that

$$\sum_{\tau \in \mathcal{T}} \mathbf{C}_x^\tau = \sum_{\tau \in \mathcal{T}} \left(\int_\tau \varphi_j \frac{\partial}{\partial x} \varphi_i \, d(x,y) \right)_{i,j} = \left(\sum_{\tau \in \mathcal{T}} \int_\tau \varphi_j \frac{\partial}{\partial x} \varphi_i \, d(x,y) \right)_{i,j} \tag{12.30}$$

$$= \left(\int_\Omega \varphi_j \frac{\partial}{\partial x} \varphi_i \, d(x,y) \right)_{i,j} = \mathbf{C}_x. \tag{12.31}$$

Analogously, all other operators discussed so far can be decomposed into corresponding element matrices.

Note that for typical finite elements $(\varphi_i)_i$ with a relatively small support, the element matrices $\mathbf{C}_x^\tau$ are very thinly populated. Usually, the number of non-zero entries of a $\mathbf{C}_x^\tau$ is k^2, where k is the number of basis functions associated with an element of $\mathcal{T}$ (assuming this to be constant).

12.2.3 Diffusive Low-Order Integration Scheme

In this section, we construct a low-order solution to the equation system given by Equation 12.22 that is guaranteed to be positive and, hence, free

from spurious oscillations. In order to do so, we introduce a so-called discrete diffusion operator that is added to our problem at hand and we integrate in time using the explicit Euler method. The concept of discrete diffusion operators is defined in the following.

Definition 12.2 (Discrete Diffusion Operator)**.** A symmetric matrix $D = (d_{ij})_{i,j} \in \mathbb{R}^{n \times n}$ is called *discrete diffusion operator* if it has zero row and column sums:

$$\sum_{k=1}^{n} d_{ik} = \sum_{k=1}^{n} d_{ki} = 0 \quad \forall\, i = 1, \ldots, n \tag{12.32}$$

Since a discrete diffusion operator has—by definition—zero column sums, such an operator conserves mass by virtue of the same arguments as in the proof of Lemma 12.1.

It would be simple to construct a suitable discrete diffusion operator $\mathbf{D} = \sum_{\tau \in \mathcal{T}} \mathbf{D}^\tau$ for our problem if we could reformulate Equation 12.19 (while ignoring the source term) as

$$\mathbf{M}_L \frac{\partial}{\partial t} \mathbf{u} = \left(\sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau + \sum_{\tau \in \mathcal{T}} \mathbf{D}^\tau \right) \mathbf{u}, \tag{12.33}$$

where $\sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau$ is a linearization of our non-linear advective fluxes and $\mathbf{M}_L$ is the so-called *lumped mass matrix*[1] defined by

$$(\mathbf{M}_L)_{ij} = \begin{cases} \sum_{k=1}^{n} (\mathbf{M}_C)_{ik} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{12.34}$$

It can be proven (and we will do so below in Lemma 12.3) that if the matrix

$$\sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau + \sum_{\tau \in \mathcal{T}} \mathbf{D}^\tau \tag{12.35}$$

has no negative off-diagonal entries, which can be achieved easily by choosing the $\mathbf{D}^\tau$ appropriately, the solution of Equation 12.33 is non-negative everywhere.

[1] "Lumping" the consistent mass matrix is a common approach to constructing computationally effective (low-order) FEM schemes (Fisher et al. 2005).

In the following, we construct $\mathbf{K}_x^\tau$ (and analogously $\mathbf{K}_y^\tau$) to fulfill

$$\sum_{j=1}^{n}(\mathbf{K}_x^\tau)_{ij}u_j = \sum_{j=1}^{n}(\mathbf{C}_{A,x}^\tau)_{ij}q_{x,j} \quad \forall\, i = 1,\dots,n, \tag{12.36}$$

where

$$\mathbf{C}_{A,x}^\tau := \mathbf{C}_x^\tau - \mathbf{C}_{B,x}^\tau = \left(-\int_\tau \varphi_i \frac{\partial}{\partial x}\varphi_j \, d(x,y)\right)_{i,j}. \tag{12.37}$$

Note that $\mathbf{C}_{A,x}^\tau$ has zero row sums:

$$\sum_{j=1}^{n}(\mathbf{C}_{A,x}^\tau)_{ij} = -\int_\tau \varphi_i \frac{\partial}{\partial x}\underbrace{\sum_{j=1}^{n}\varphi_j}_{\equiv 1} \, d(x,y) = 0 \tag{12.38}$$

This implies that we can express the diagonal entries through the sum of the off-diagonal ones:

$$(\mathbf{C}_{A,x}^\tau)_{ii} = -\sum_{j\neq i}(\mathbf{C}_{A,x}^\tau)_{ij} \tag{12.39}$$

With the linearized fluxes

$$v_{x,ij} := \begin{cases} \frac{q_{x,j}-q_{x,i}}{u_j-u_i} & \text{if } u_i \neq u_j \\ 0 & \text{otherwise,} \end{cases} \tag{12.40}$$

we define the linearized transport operator $\mathbf{K}_x$ per element $\tau \in \mathcal{T}$ by

$$(\mathbf{K}_x^\tau)_{ij} := (\mathbf{C}_{A,x}^\tau)_{ij}v_{x,ij} \quad \forall\, i \neq j \tag{12.41}$$

$$(\mathbf{K}_x^\tau)_{ii} := -\sum_{j\neq i}(\mathbf{K}_x^\tau)_{ij}. \tag{12.42}$$

These element matrices fulfill

$$\sum_{j=1}^{n}(\mathbf{K}_x^\tau)_{ij}u_j \;=\; \sum_{j\neq i}(\mathbf{K}_x^\tau)_{ij}u_j + (\mathbf{K}_x^\tau)_{ii}u_i \tag{12.43}$$

$$= \sum_{j \neq i} (\mathbf{K}_x^\tau)_{ij} (u_j - u_i) \tag{12.44}$$

$$= \sum_{j \neq i} (\mathbf{C}_{A,x}^\tau)_{ij} v_{x,ij} (u_j - u_i) \tag{12.45}$$

$$= \sum_{j \neq i} (\mathbf{C}_{A,x}^\tau)_{ij} (q_{x,j} - q_{x,i}) \tag{12.46}$$

$$\overset{(12.39)}{=} \sum_{j=1}^{n} (\mathbf{C}_{A,x}^\tau)_{ij} q_{x,j} \quad \forall\, i = 1, \ldots, n. \tag{12.47}$$

Equations 12.45 and 12.46 are indeed equal (even for $u_j = u_i$) because it holds that

$$u_j = u_i \;\Rightarrow\; q_{x,j} = q_{x,i}. \tag{12.48}$$

This means that with

$$\mathbf{K}^\tau := \mathbf{K}_x^\tau + \mathbf{K}_y^\tau \tag{12.49}$$

we can express our problem from Equation 12.22 with a lumped mass matrix as

$$\frac{\partial}{\partial t}\mathbf{u} = \mathbf{M}_L^{-1} \left(\left(\mathbf{C}_x - \mathbf{C}_{B,x} \right) \mathbf{q}_x + \left(\mathbf{C}_y - \mathbf{C}_{B,y} \right) \mathbf{q}_y \right) + \mathbf{s} \tag{12.50}$$

$$= \mathbf{M}_L^{-1} \sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau \mathbf{u} + \mathbf{s}. \tag{12.51}$$

The discrete diffusion operator $\mathbf{D}$ given by

$$\mathbf{D}_{ij}^\tau := \max\{0, -\mathbf{K}_{ij}^\tau, -\mathbf{K}_{ji}^\tau\} \quad \forall\, i \neq j \tag{12.52}$$

$$\mathbf{D}_{ii}^\tau := -\sum_{j \neq i} \mathbf{D}_{ij}^\tau \tag{12.53}$$

$$\mathbf{D} := \sum_{\tau \in \mathcal{T}} \mathbf{D}^\tau \tag{12.54}$$

is obviously suitable to eliminate all negative off-diagonal entries of $\sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau$. As shown in the following lemma, this property of $\mathbf{D}$ ensures that the solution to Equation 12.33 is non-negative everywhere.

Lemma 12.3. *If for all $i = 1, \dots, n$ it holds that $u_i(0) \geqslant 0$ and that the evolution of u_i is governed by*

$$\frac{\partial}{\partial t} u_i = \frac{1}{(\mathbf{M}_L)_{ii}} \sum_{j=1}^{n} \left(\sum_{\tau \in \mathcal{T}} \mathbf{K}_{ij}^{\tau} + \sum_{\tau \in \mathcal{T}} \mathbf{D}_{ij}^{\tau} \right) u_j, \tag{12.55}$$

then

$$u_i(t) \geqslant 0 \quad \forall\, t \in [0, t_{\max}]. \tag{12.56}$$

Proof. Because of the zero row sum property of the operators $\mathbf{K}_x^{\tau}$, $\mathbf{K}_y^{\tau}$, and $\mathbf{D}^{\tau}$, it holds (cf. Equation 12.45):

$$\frac{\partial}{\partial t} u_i = \frac{1}{(\mathbf{M}_L)_{ii}} \sum_{j \neq i} \sum_{\tau \in \mathcal{T}} \left((\mathbf{C}_{A,x}^{\tau})_{ij} v_{x,ij} + (\mathbf{C}_{A,y}^{\tau})_{ij} v_{y,ij} + \mathbf{D}_{ij}^{\tau} \right) (u_j - u_i) \tag{12.57}$$

From the definition of $\mathbf{D}_{ij}^{\tau}$, we can see that

$$\begin{aligned} (\mathbf{C}_{A,x}^{\tau})_{ij} v_{x,ij} + (\mathbf{C}_{A,y}^{\tau})_{ij} v_{y,ij} + \mathbf{D}_{ij}^{\tau} &= (\mathbf{K}_x^{\tau})_{ij} + (\mathbf{K}_y^{\tau})_{ij} + \mathbf{D}_{ij}^{\tau} && (12.58) \\ &= \mathbf{K}_{ij}^{\tau} + \mathbf{D}_{ij}^{\tau} \geqslant 0 \quad \forall\, i \neq j. && (12.59) \end{aligned}$$

Note that it also holds that

$$(\mathbf{M}_L)_{ii} = \sum_{j=1}^{n} (\mathbf{M}_C)_{ij} = \int_{\Omega} \varphi_i \sum_{j=1}^{n} \varphi_j \, d(x,y) \overset{(12.7)}{=} \int_{\Omega} \varphi_i \, d(x,y) \overset{(12.8)}{>} 0. \tag{12.60}$$

If u_i is a (local) minimum of u_h, then $u_j - u_i \geqslant 0$, which implies that

$$\frac{\partial}{\partial t} u_i \geqslant 0. \tag{12.61}$$

Hence, a (local) minimum cannot decrease. This implies that, since the initial values $\mathbf{u}(0)$ are non-negative, all values of $\mathbf{u}(t)$ must be non-negative for any $t \in [0, t_{\max}]$. □

With $\mathbf{D}$, we can define a low-order solution in time step $\eta + 1$ by applying the explicit Euler method:

$$\mathbf{u}_L^{(\eta+1)} = \mathbf{u}^{(\eta)} + \Delta t \mathbf{M}_L^{-1} \left(\mathbf{C}_{A,x} \mathbf{q}_x^{(\eta)} + \mathbf{C}_{A,y} \mathbf{q}_y^{(\eta)} + \mathbf{D}\mathbf{u}^{(\eta)} + \mathbf{M}_L \mathbf{s}^{(\eta)} \right) \quad (12.62)$$

Note that the low-order solution $\mathbf{u}_L^{(\eta+1)}$ only depends on the blended solution $\mathbf{u}^{(\eta)}$ from the previous time step (to be defined below) and *not* (directly) on the *low-order* solution from the previous step.

By decomposing all operators into element matrices, we can define the low-order element contributions or fluxes

$$F_L^{\tau,\eta} := \mathbf{C}_{A,x}^{\tau} \mathbf{q}_x^{(\eta)} + \mathbf{C}_{A,y}^{\tau} \mathbf{q}_y^{(\eta)} + \mathbf{D}^{\tau} \mathbf{u}^{(\eta)} + \mathbf{M}_L^{\tau} \mathbf{s}^{(\eta)}, \quad (12.63)$$

which allow us to rewrite Equation 12.62 as

$$\mathbf{u}_L^{(\eta+1)} = \mathbf{u}^{(\eta)} + \Delta t \mathbf{M}_L^{-1} \sum_{\tau \in \mathcal{T}} F_L^{\tau,\eta}. \quad (12.64)$$

12.2.4 High-Order Integration Scheme and Anti-Diffusive Element Contributions

To derive a high-order solution, we employ an explicit multi-step method (Adams-Bashforth) with r steps and use the consistent mass matrix. Analogously to Equation 12.63, we define high-order element contributions

$$f_H^{\tau,\eta} := \mathbf{C}_{A,x}^{\tau} \mathbf{q}_x^{(\eta)} + \mathbf{C}_{A,y}^{\tau} \mathbf{q}_y^{(\eta)} + \mathbf{M}_C^{\tau} \mathbf{s}^{(\eta)} \quad (12.65)$$

$$F_H^{\tau,\eta} := \sum_{k=0}^{r-1} \alpha_k f_H^{\tau,\eta-k} \quad (12.66)$$

with α_k being the weights of the multi-step method. With these element contributions, the high-order method is given by

$$\mathbf{u}_H^{(\eta+1)} = \mathbf{u}^{(\eta)} + \Delta t \mathbf{M}_C^{-1} \sum_{\tau \in \mathcal{T}} F_H^{\tau,\eta}. \quad (12.67)$$

Using element contributions in the formulation of the high-order method enables us to express the high-order solution in terms of the low-order solution:

$$\mathbf{M}_C\mathbf{u}_H^{(\eta+1)} = \mathbf{M}_C\mathbf{u}^{(\eta)} + \Delta t \sum_{\tau\in\mathcal{T}} F_H^{\tau,\eta} \tag{12.68}$$

$$\Leftrightarrow\ \mathbf{M}_L\mathbf{u}_H^{(\eta+1)} = (\mathbf{M}_L - \mathbf{M}_C)\mathbf{u}_H^{(\eta+1)} + (\mathbf{M}_C - \mathbf{M}_L)\mathbf{u}^{(\eta)} \tag{12.69}$$

$$+ \mathbf{M}_L\mathbf{u}^{(\eta)} + \Delta t \sum_{\tau\in\mathcal{T}} F_H^{\tau,\eta} \tag{12.70}$$

$$\Leftrightarrow\ \mathbf{M}_L\mathbf{u}_H^{(\eta+1)} = \mathbf{M}_L\mathbf{u}^{(\eta)} + \sum_{\tau\in\mathcal{T}} \Delta t\, F_H^{\tau,\eta} \tag{12.71}$$

$$+ (\mathbf{M}_L^\tau - \mathbf{M}_C^\tau)(\mathbf{u}_H^{(\eta+1)} - \mathbf{u}^{(\eta)}) \tag{12.72}$$

$$\Leftrightarrow\ \mathbf{M}_L\mathbf{u}_H^{(\eta+1)} = \mathbf{M}_L\mathbf{u}_L^{(\eta+1)} + \sum_{\tau\in\mathcal{T}} \Delta t(F_H^{\tau,\eta} - F_L^{\tau,\eta}) \tag{12.73}$$

$$+ (\mathbf{M}_L^\tau - \mathbf{M}_C^\tau)(\mathbf{u}_H^{(\eta+1)} - \mathbf{u}^{(\eta)}) \tag{12.74}$$

With the anti-diffusive element contributions or fluxes

$$F_A^{\tau,\eta} := \Delta t(F_H^{\tau,\eta} - F_L^{\tau,\eta}) + (\mathbf{M}_L^\tau - \mathbf{M}_C^\tau)(\mathbf{u}_H^{(\eta+1)} - \mathbf{u}^{(\eta)}) \tag{12.75}$$

we can, therefore, write the high-order solution as

$$\mathbf{M}_L\mathbf{u}_H^{(\eta+1)} = \mathbf{M}_L\mathbf{u}_L^{(\eta+1)} + \sum_{\tau\in\mathcal{T}} F_A^{\tau,\eta}. \tag{12.76}$$

By introducing parameters $\lambda_\tau^{(\eta)} \in [0,1]$, we can locally blend the high-order and the low-order solution together and define $\mathbf{u}^{(\eta+1)}$, the FCT solution for time step $\eta+1$, via

$$\mathbf{M}_L\mathbf{u}^{(\eta+1)} = \mathbf{M}_L\mathbf{u}_L^{(\eta+1)} + \sum_{\tau\in\mathcal{T}} \lambda_\tau^{(\eta)} F_A^{\tau,\eta}. \tag{12.77}$$

For $\lambda_\tau^{(\eta)} = 0$, we locally obtain the low-order solution, for $\lambda_\tau^{(\eta)} = 1$, the high-order solution, and for values in between, a combination of both.

In order to prove that the scheme given by Equation 12.77 conserves mass (if we ignore the boundary transport operators and the source term), we show that adding any amount of anti-diffusion to a semi-discrete low-order scheme (of which we already know that it conserves mass) does not affect the total mass in the system. Our semi-discrete low-order scheme without the source term is defined by

$$\mathbf{M}_L \frac{\partial}{\partial t}\mathbf{u} = \left(\sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau + \sum_{\tau \in \mathcal{T}} \mathbf{D}^\tau \right) \mathbf{u} \tag{12.78}$$

and the high-order scheme without the source term by

$$\mathbf{M}_C \frac{\partial}{\partial t}\mathbf{u} = \sum_{\tau \in \mathcal{T}} \mathbf{K}^\tau \mathbf{u}. \tag{12.79}$$

The difference between the residuals of both schemes (i. e., the anti-diffusion that transforms the low-order into the high-order scheme) is, by construction, given by

$$\mathbf{f} = \sum_{\tau \in \mathcal{T}} \mathbf{f}^\tau = \sum_{\tau \in \mathcal{T}} \left((\mathbf{M}_L^\tau - \mathbf{M}_C^\tau) \frac{\partial}{\partial t}\mathbf{u} - \mathbf{D}^\tau \mathbf{u} \right) \tag{12.80}$$

with

$$(\mathbf{M}_L^\tau)_{ij} = \begin{cases} \sum_{k=1}^{n} (\mathbf{M}_C^\tau)_{ik} & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{12.81}$$

In the following lemma, we show that each $\mathbf{f}^\tau$ can be decomposed into anti-symmetrical mass fluxes between nodes (the same amount of mass added to node i from node j is subtracted from node j). Therefore, adding *any* linear combination of the $\mathbf{f}^\tau$ to the right-hand side of Equation 12.78 does neither create nor destroy mass (the same amount of mass that is added to one node is always subtracted from another node; cf. Kuzmin 2010).

Lemma 12.4. *For all $\tau \in \mathcal{T}$, there exist $(f_{ij}^\tau)_{i,j}$ so that*

$$\mathbf{f}_i^\tau = \sum_{j \neq i} f_{ij}^\tau \quad \text{and} \quad f_{ji}^\tau = -f_{ij}^\tau. \tag{12.82}$$

Proof. Let $\tau \in \mathcal{T}$ and $i \in \{1, \dots, n\}$. Since $\mathbf{D}^\tau$ has zero row sums, it holds that

$$(\mathbf{D}^\tau \mathbf{u})_i = \sum_{j \neq i} \mathbf{D}_{ij}^\tau (u_j - u_i). \tag{12.83}$$

Similarly, it holds that

$$(\mathbf{M}_C^\tau \mathbf{u} - \mathbf{M}_L^\tau \mathbf{u})_i = \sum_{j=1}^n (\mathbf{M}_C^\tau)_{ij} u_j - \sum_{j=1}^n (\mathbf{M}_C^\tau)_{ij} u_i = \sum_{j \neq i} (\mathbf{M}_C^\tau)_{ij} (u_j - u_i). \tag{12.84}$$

Define

$$f_{ij}^\tau := \left((\mathbf{M}_C^\tau)_{ij} \frac{\partial}{\partial t} + \mathbf{D}_{ij}^\tau \right) (u_i - u_j), \tag{12.85}$$

which fulfill $f_{ji}^\tau = -f_{ij}^\tau$ (because $\mathbf{M}_C^\tau$ and $\mathbf{D}^\tau$ are symmetric) and where the derivative $\frac{\partial}{\partial t}$ is replaced by a finite difference after discretization. With these f_{ij}^τ, it holds that

$$\mathbf{f}_i^\tau = \sum_{j \neq i} f_{ij}^\tau. \tag{12.86}$$

□

12.2.5 Flux Limiter

We are interested in determining the $\lambda_\tau^{(\eta)} \in [0, 1]$ in a way that maximizes the amount of anti-diffusion which is added to the low-order solution (because this gives us high accuracy) without introducing spurious oscillations. For this purpose, we employ Zalesak's flux limiter (Zalesak 1979), which is described in the following. However, before explaining the algorithm itself, we need to introduce two sets that are related to the discretization of our domain. For all $\tau \in \mathcal{T}$, the set $\mathcal{D}(\tau)$ contains the indices of all basis functions that are associated with τ:

$$\mathcal{D}(\tau) := \left\{ i \in \mathbb{N} \,\middle|\, \int_\tau |\varphi_i| \, d(x, y) \neq 0 \right\} \tag{12.87}$$

For all $i = 1, \dots, n$, the set $\mathcal{N}(i)$ contains the indices of all basis functions that φ_i interacts with (i. e., its "neighbors"):

$$\mathcal{N}(i) := \{ j \in \mathbb{N} \mid \exists\, \tau \in \mathcal{T} : i \in \mathcal{D}(\tau) \wedge j \in \mathcal{D}(\tau) \} \tag{12.88}$$

For Zalesak's limiter, we first determine the sums of all outgoing and incoming fluxes for each node $i = 1, \ldots, n$ via

$$P_i^{+,\eta} = \sum_{\tau \in \mathcal{T}} \max\left(0, (F_A^{\tau,\eta})_i\right) \tag{12.89}$$

$$P_i^{-,\eta} = \sum_{\tau \in \mathcal{T}} \min\left(0, (F_A^{\tau,\eta})_i\right). \tag{12.90}$$

We then calculate the maximum and minimum increment of the low-order solution that does not create new local maxima or minima:

$$Q_i^{+,\eta} = U_i^{\max,\eta} - (\mathbf{u}_L^{(\eta+1)})_i \tag{12.91}$$

$$Q_i^{-,\eta} = U_i^{\min,\eta} - (\mathbf{u}_L^{(\eta+1)})_i \tag{12.92}$$

with

$$U_i^{\max,\eta} = \max_{j \in \mathcal{N}(i)} (\mathbf{u}_L^{(\eta+1)})_j \tag{12.93}$$

$$U_i^{\min,\eta} = \min_{j \in \mathcal{N}(i)} (\mathbf{u}_L^{(\eta+1)})_j \tag{12.94}$$

This enables us to define the fractions of outgoing or incoming fluxes that can be added to our solution without violating the positivity constraint as

$$R_i^{+,\eta} = \begin{cases} \min\left(1, Q_i^{+,\eta}/P_i^{+,\eta}\right) & \text{if } P_i^{+,\eta} > 0 \\ 0 & \text{if } P_i^{+,\eta} = 0 \end{cases} \tag{12.95}$$

$$R_i^{-,\eta} = \begin{cases} \min\left(1, Q_i^{-,\eta}/P_i^{-,\eta}\right) & \text{if } P_i^{-,\eta} < 0 \\ 0 & \text{if } P_i^{-,\eta} = 0. \end{cases} \tag{12.96}$$

The $\lambda_\tau^{(\eta)}$ are chosen in a way that guarantees that no individual anti-diffusive fluxes associated with a node of element τ violate the positivity constraint:

$$R_i^{\tau,\eta} := \begin{cases} R_i^{+,\eta} & \text{if } (F_A^{\tau,\eta})_i > 0 \\ R_i^{-,\eta} & \text{if } (F_A^{\tau,\eta})_i < 0 \\ 1 & \text{otherwise} \end{cases} \tag{12.97}$$

$$\lambda_\tau^{(\eta)} := \min_{i \in \mathcal{D}(\tau)} \left(R_i^{\tau,\eta} \right) \tag{12.98}$$

This choice of $\lambda_\tau^{(\eta)}$ ensures that the limited anti-diffusive fluxes do not introduce negative values in the solution $\mathbf{u}^{(\eta+1)}$ and that the solution is, hence, free from spurious oscillations.

Note that the choice of $\lambda_\tau^{(\eta)}$ in Zalesak's limiter is in general not optimal: the algorithm ignores that incoming and outgoing fluxes could cancel each other out, which would make it possible to add even more anti-diffusion to the solution without violating the positivity constraint. However, such a global perspective on the fluxes would require solving a global optimization problem including all nodes. Since this would be too computationally expensive, we opt for optimizing the $\lambda_\tau^{(\eta)}$ only locally and accept that the solution can contain a little more diffusion than necessary.

12.3 Summary of the FCT FEM Algorithm

In this section, we give a summary of the algorithmic steps involved in computing the FCT solution as described above. Additionally, we discuss the complexity of our FCT algorithm.

For the initialization of the algorithm, the different discrete operators (the consistent and lumped mass matrix as well as the transport operators) have to be computed. They must be stored as element matrices as described in Section 12.2.2 (note our comment on the sparsity of the element matrices in this section).

For each time step of the FCT FEM algorithm, the following operations have to be performed:

1. Set $\Delta \mathbf{u}_L \leftarrow 0$ and $\mathbf{b} \leftarrow 0$.

   Do for all $\tau \in \mathcal{T}$:

   1. For $i, j \in \mathcal{D}(\tau)$, compute the linearized fluxes according to

$$v_{x/y,ij} = \begin{cases} \frac{q_{x/y,j} - q_{x/y,i}}{u_j - u_i} & \text{if } u_i \neq u_j \\ 1 & \text{otherwise.} \end{cases} \tag{12.99}$$

2. Compute $\mathbf{D}^\tau$ according to:

$$\mathbf{D}_{ij}^\tau = \max\left\{0, -\left((\mathbf{C}_{A,x}^\tau)_{ij} v_{x,ij} + (\mathbf{C}_{A,y}^\tau)_{ij} v_{y,ij}\right), \right. \tag{12.100}$$

$$\left. -\left((\mathbf{C}_{A,x}^\tau)_{ji} v_{x,ji} + (\mathbf{C}_{A,y}^\tau)_{ji} v_{y,ji}\right)\right\} \tag{12.101}$$

$$\mathbf{D}_{ii}^\tau = -\sum_{j \neq i} \mathbf{D}_{ij}^\tau \tag{12.102}$$

3. Compute the element contributions according to

$$F_L^\tau = \mathbf{C}_{A,x}^\tau \mathbf{q}_x + \mathbf{C}_{A,y}^\tau \mathbf{q}_y + \mathbf{D}^\tau \mathbf{u} + \mathbf{M}_L^\tau \mathbf{s} \tag{12.103}$$

$$f_H^{\tau,\eta} = \mathbf{C}_{A,x}^\tau \mathbf{q}_x + \mathbf{C}_{A,y}^\tau \mathbf{q}_y + \mathbf{M}_C^\tau \mathbf{s} \tag{12.104}$$

$$F_H^\tau = \sum_{k=0}^{r-1} \alpha_k f_H^{\tau,\eta-k}. \tag{12.105}$$

Note that

$$F_L^\tau = f_H^{\tau,\eta} + \mathbf{D}^\tau \mathbf{u} - \mathbf{M}_C^\tau \mathbf{s} + \mathbf{M}_L^\tau \mathbf{s}. \tag{12.106}$$

4. Set $\Delta \mathbf{u}_L \leftarrow \Delta \mathbf{u}_L + F_L^\tau$ and $\mathbf{b} \leftarrow \mathbf{b} + F_H^\tau$.

2. Set $\Delta \mathbf{u}_L \leftarrow \Delta t \mathbf{M}_L^{-1} \Delta \mathbf{u}_L$.
3. Set $\mathbf{b} \leftarrow \Delta t \mathbf{b}$ and $\Delta \mathbf{u}_H \leftarrow \Delta \mathbf{u}_L$ and solve $\mathbf{M}_C \Delta \mathbf{u}_H = \mathbf{b}$ with an iterative solver such as the Conjugate Gradient (CG) method (use the element matrix representation of $\mathbf{M}_C$ for the necessary matrix-vector multiplications).
4. Set $\mathbf{u}_L \leftarrow \mathbf{u} + \Delta \mathbf{u}_L$ ($\mathbf{u}_L$ can be stored in the storage location of $\mathbf{u}$ since the value of the latter is not needed anymore).
5. Set $P^+ \leftarrow 0$, $P^- \leftarrow 0$, $U^{\max} \leftarrow 0$, and $U^{\min} \leftarrow 0$.
   Do for all $\tau \in \mathcal{T}$:
   1. Set $F_\tau^A \leftarrow \Delta t (F_H^\tau - F_L^\tau) + (\mathbf{M}_L^\tau - \mathbf{M}_C^\tau) \Delta \mathbf{u}_H$.
   2. Do for all $i \in \mathcal{D}(\tau)$:
      (a) Compute step-wise: $u_L^{\tau,\max} = \max\{(\mathbf{u}_L)_k \mid k \in \mathcal{D}(\tau)\}$ and $u_L^{\tau,\min} = \min\{(\mathbf{u}_L)_k \mid k \in \mathcal{D}(\tau)\}$.

(b) Set $P_i^+ \leftarrow P_i^+ + \max(0, (F_A^\tau)_i)$ and $P_i^- \leftarrow P_i^- + \min(0, (F_A^\tau)_i)$.

3. For all $i \in \mathcal{D}(\tau)$, set $U_i^{\max} \leftarrow \max(U_i^{\max}, u_L^{\tau,\max})$ and $U_i^{\min} \leftarrow \min(U_i^{\min}, u_L^{\tau,\min})$.

6. Do for all $i \in \{1, \ldots, n\}$:

$$R_i^+ \leftarrow \begin{cases} \min(1, (U_i^{\max} - (\mathbf{u}_L)_i)/P_i^+) & \text{if } P_i^+ > 0 \\ 0 & \text{if } P_i^+ = 0 \end{cases} \tag{12.107}$$

$$R_i^- \leftarrow \begin{cases} \min(1, (U_i^{\min} - (\mathbf{u}_L)_i)/P_i^-) & \text{if } P_i^- < 0 \\ 0 & \text{if } P_i^- = 0 \end{cases} \tag{12.108}$$

The values of $R^\pm$ can be stored in the storage location of $P^\pm$ since the values of the latter variables are not needed anymore.

7. Set $\mathbf{u} \leftarrow \mathbf{u}_L$.

8. Set $\Delta\mathbf{u}_A \leftarrow 0$ ($\Delta\mathbf{u}_A$ can be stored in the storage location of $\Delta\mathbf{u}_H$ since the value of the latter is not needed anymore).

   Do for all $\tau \in \mathcal{T}$:

   1. Calculate

$$\lambda_\tau = \min_{i \in \mathcal{D}(\tau)} \begin{cases} R_i^+ & \text{if } (F_A^\tau)_i > 0 \\ R_i^- & \text{if } (F_A^\tau)_i < 0 \\ 1 & \text{if } (F_A^\tau)_i = 0 \end{cases} \tag{12.109}$$

      by iterating over all $i \in \mathcal{D}(\tau)$.

   2. Set $\Delta\mathbf{u}_A \leftarrow \Delta\mathbf{u}_A + \lambda_\tau F_A^\tau$.

9. Set $\mathbf{u} \leftarrow \mathbf{u} + \mathbf{M}_L^{-1}\Delta\mathbf{u}_A$.

12.3.1 Algorithm Complexity

For typical finite elements $(\varphi_i)_i$ with a relatively small support, there exists a $k \in \mathbb{N}$ that is independent of n so that

$$|\mathcal{D}(\tau)| \leqslant k \quad \forall\, \tau \in \mathcal{T}. \tag{12.110}$$

Therefore, all element matrices that appear in our FCT algorithm are very thinly populated and possess at most k^2 non-zero entries.

The number of elements that a node $i \in \{1, \ldots, n\}$ can be associated with is also bound by a constant $l \in \mathbb{N}$ independent of n:

$$|\{\tau \in \mathcal{T} \mid i \in \mathcal{D}(\tau)\}| \leqslant l \quad \forall\, i \in \{1, \ldots, n\} \tag{12.111}$$

Thus, the number of elements $|\mathcal{T}|$ is bound by ln. This implies that the storage requirements of the algorithm for the element matrices and element vectors is of order $\mathcal{O}(k^2 ln) = \mathcal{O}(n)$.

Since in each top-level step of the algorithm, we iterate over all nodes or elements (the number of which is bound by n) and since all nested iterations are over sets with a cardinality limited by a constant independent of n, the computational effort for one time step is $\mathcal{O}(n)$. Note that for this to be true, we have to assume that inverting the consistent mass matrix with an iterative method is also of order $\mathcal{O}(n)$. This is a reasonable assumption as $\mathbf{M}_C$ is very well-conditioned. Accordingly, a good iterative solver only requires a small number of steps that is (virtually) independent of n to solve the system with acceptable accuracy.

12.4 Numerical Experiments

In this section, we present results from numerical experiments using our FCT FEM algorithm. The results show that our method possesses the following two desirable properties of a numerical solver for advection problems:

1. No spurious oscillations

2. The scheme is not overly diffusive; in particular maxima and minima do not decay noticeably for linear advection fields

Since we are only interested in a qualitative assessment of these properties, we refrain from reporting any quantitative error norms and rely solely on visual inspection of the results.

12.4.1 Linear Advection in One Dimension

We applied our FCT FEM scheme to the one-dimensional problem that we studied in the introduction of this chapter to show the shortcomings both of a standard Galerkin approach and of an overly diffusive method for advection equations. In this problem, a profile is advected with unit velocity on the domain $\Omega = [-1, 1]$ with periodic boundary conditions. If you look at the results in Figure 12.3, you will notice that with our FCT solver, there are no ripples as there are with the standard Galerkin approach (Figure 12.1) and that maxima and minima are well retained in comparison with a diffusive low-order scheme (Figure 12.2).

12.4.2 Slotted Cylinder Rotation Problem

In order to test our FCT method in two dimensions, we reuse a problem from John and Schmeyer (2008) that they employ to compare the quality of different numerical schemes for advection equations. In this problem, a slotted cylinder (see Figure 12.4) is rotated one full revolution and is afterwards compared to the initial data. In the beginning, the slotted cylinder is centered around $(0.5, 0)$ on the domain $\Omega = [-1, 1]^2$, which is discretized into 128×128 elements and is equipped with periodic boundary conditions. A counter-clockwise rotational advection field around the origin is applied which lets the cylinder perform a full revolution every 2π units of time. Ideally, after $t = 2\pi$, the numerical solution should be identical with the initial data depicted in Figure 12.4.

Figure 12.5 shows the slotted cylinder after one revolution with our FCT scheme. As in the one-dimensional experiment, there are no oscillations. The slot is still clearly visible and the height of the cylinder has decayed only minimally. In our experiment, mass is conserved up to machine precision.

All in all, we can conclude that our explicit multi-step FCT FEM solver succeeds in accurately approximating the solution of advection problems without producing spurious oscillations or overly diffusive results.

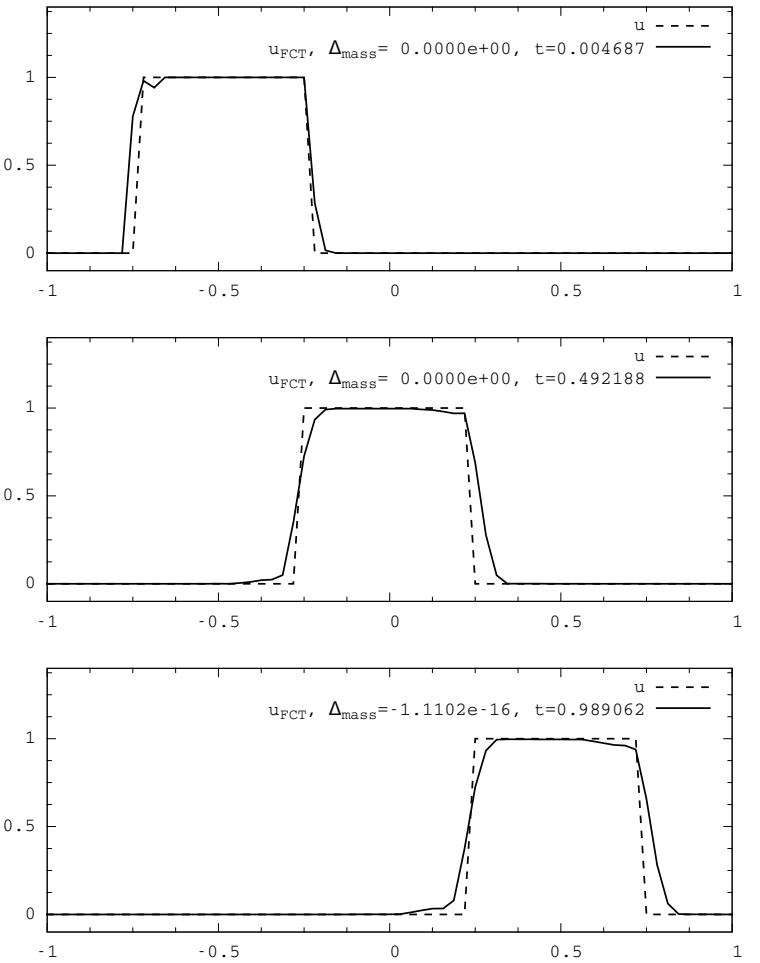


Figure 12.3. A profile is advected with unit velocity on the domain $[-1,1]$ with periodic boundary conditions. u is the analytical solution and u_{FCT} is calculated using our FCT FEM scheme with linear elements to discretize space and the three-step Adams-Bashforth method for the high-order solution. Note that mass is conserved up to machine precision (Δ_{mass}).

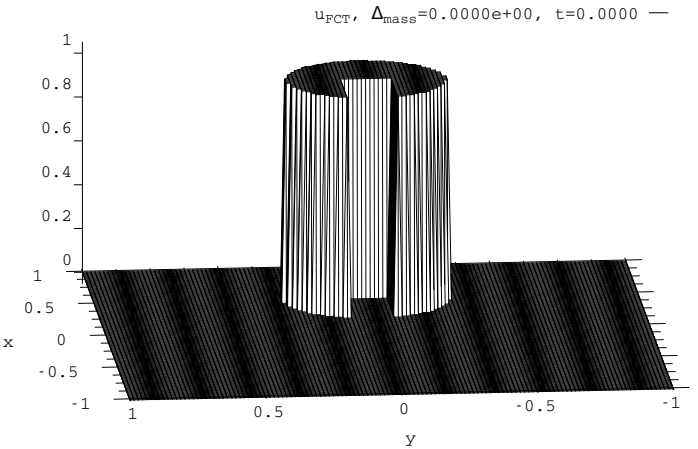


Figure 12.4. Initial data for the slotted cylinder problem.

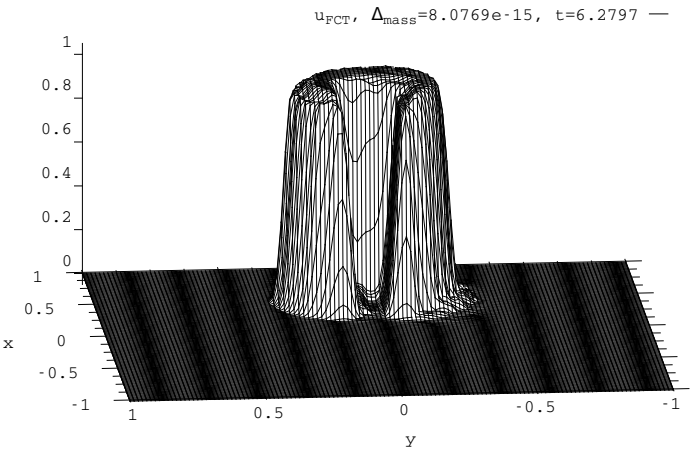


Figure 12.5. The slotted cylinder after being advected with our FCT FEM scheme for one full revolution.

Part IV

Evaluation

Chapter 13

Evaluation of the Sprat PDE DSL

In this chapter, we present results from evaluating the Sprat PDE DSL as part of the case study regarding the application of the Sprat Approach (see Section 5.2.4). For the evaluation of the DSL, we conducted expert interviews among a group of eight professionals which included both specialists from the application domain of the language and professional DSL designers. In addition to the interviews, we analyzed the performance of the Sprat PDE DSL using benchmark experiments. The benchmark results also served as a basis for the expert interviews because they allowed to demonstrate to the experts that the DSL does not compromise runtime performance compared to GPL solutions.

The main results of the interview study are:

1. Overall, the Sprat PDE DSL is perceived as a valuable addition to the field of applied mathematics: the language simplifies the implementation of well-maintainable (FEM) PDE solvers without performance drawbacks. It makes experimenting with algorithms and checking for their correctness easier.
2. The meta-model of the Sprat PDE DSL contains all necessary abstractions for building (FEM) PDE solvers (completeness) and structures these abstractions in a consistent way (orthogonality).
3. Its syntax is characterized as simple, natural, as well as easy to learn and read.
4. The technical implementation of the DSL itself is described as well-maintainable and the employed implementation technologies are generally accepted by the interviewed domain experts.

In addition to the evaluation of the Sprat PDE DSL, our interview study reveals differing stances towards good DSL design practices for computational

science. These insights can potentially help professional DSL developers to make sure their DSLs are readily accepted in the computational science community.

The rest of this chapter is divided into two main parts: Section 13.1 reports on benchmarking the Sprat PDE DSL and Section 13.2 discusses the results of our interview study.

13.1 Performance Analysis

As mentioned in Section 7.3.2, a DSL for computational science—and especially for HPC—should not compromise runtime performance. To demonstrate that the runtime performance of our implementation of the Sprat PDE DSL is not inferior to solutions implemented in GPLs, this section reports on results of several benchmark experiments. The benchmarks compare the runtime performance either of individual DSL meta-model elements (micro-benchmarks; Section 13.1.1) or of their integrated use (macro-benchmarks; Section 13.1.2) to structurally similar C or C++ implementations.

Our benchmarking strategy bears two threats to validity, which we discuss before going on to the actual experiments:

1. All reference implementations to compare the performance of the DSL to have been created by the author alone. As a consequence, it is possible (and also very likely) that these implementations are not optimal with regard to runtime performance. However, our aim is *not* to demonstrate that, for example, our DSL implementation of a certain algorithm is faster than any possible implementation using plain C or C++. Instead, our intention is just to study the potential costs introduced by using certain DSL abstractions in place of writing functionally equivalent standard C++ code (without the corresponding DSL features). Therefore, we only have to make sure that the reference implementations we construct and the respective DSL programs are *structurally similar* to each other. This leads to the second threat to validity.
2. There is no objective metric to measure whether two implementations are "structurally similar." We address this threat by giving intersubjectively traceable explanations for each benchmark that describe by which rules we constructed the reference implementations.

All benchmarks were compiled with gcc 4.8.1 (with compiler switches `-O3 -ffast-math -funroll-loops`) and run on an Intel Core-i5. For the experiments we disabled all automatic parallelization features of the Sprat PDE DSL and compared only single-thread performance. As the threshold for statistically significant results we used $\alpha = 0.05$. The complete benchmark suite including all analysis scripts can be obtained online (Johanson 2015a).

13.1.1 Performance of Individual Language Constructs

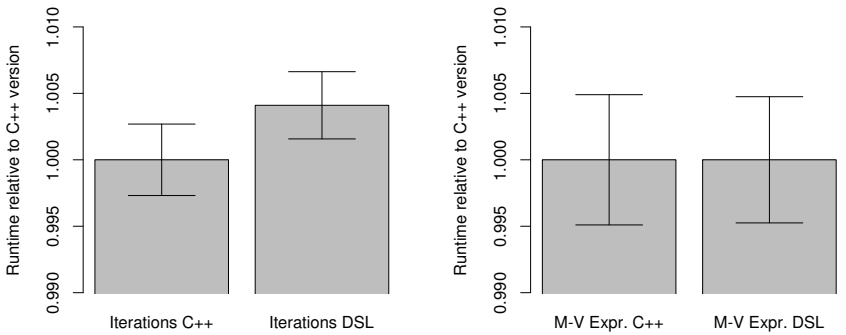
In this section, we assess the overhead that is introduced by individual high-level abstractions of the Sprat PDE DSL. A large amount of meta-model elements of the DSL are data types such as different mesh, matrix, and vector types. We do not test the performance of these data types themselves because we are only interested in how the embedding of the DSL affects the performance of the *interactions* between these types. Therefore, we focus on the runtime of the remaining categories of meta-model elements, which are *iterations over sets* and *matrix-vector expressions*.

In order to measure the performance impact of the iteration over sets concept, we record the runtime of initializing a vector depending on spatial coordinates and of assembling a discrete Laplace operator. We construct two versions of this program: one that uses the DSL abstractions of iterations over sets and another one that is structurally similar but does not use these concepts and instead relies purely on standard C index-based `for` loops. E. g., in the DSL version of the program, we initialize the vector named above with a `foreach` iteration over the set of all DoF of our mesh:

```
foreach(i, DoF(femMesh), {
    const real x = i.positionInDimension(0);
    u[i] = (x<0.5 ? 1.0 : 0.0);
})
```

In the second version of the program (which we call the "C++ version") the iteration is simply replaced by a `for` loop:

```
for(unsigned int i=0; i<vecSize; i++) {
    const real x = femMesh.dof(i).positionInDimension(0);
    u[i] = (x<0.5 ? 1.0 : 0.0);
}
```



(a) Iterations over sets **(b)** Matrix-vector expressions

Figure 13.1. Benchmark results from comparing the Sprat PDE DSL with equivalent C++ implementations (95 % confidence intervals, $n = 25$ samples).

Note that we still rely on the data structures of the Sprat PDE DSL in order not to skew timing results.

Figure 13.1a displays the runtime of the DSL version of this benchmark relative to the runtime of the C++ version. The difference in means between both versions is statistically significant ($p = 0.02629$ for Welch's t-test; Bortz and Schuster 2010) but very small: the DSL version is only about 0.4 % slower than the C++ version.

In order to assess the performance impact of embedding matrix-vector expressions into C++ via template meta-programming in the Sprat PDE DSL, we measure the time it takes to compute repeated assignments of matrix-vector expressions:

```
for(unsigned int i=0; i<1000; i++) {
    u = 0.33 * (u + v + w) + laplace * w;
}
```

In this snippet, the variables `u`, `v`, and `w` are vectors each consisting of 10^6 components and `laplace` is a discrete Laplace operator stored in a sparse matrix format.

We compare the DSL version of this benchmark for matrix-vector expressions with an implementation that uses the same data structures but employs function calls to apply the arithmetic operations. However, if this second version—which we call "C++ version" again—really used a single function call for each of the vector-vector operations like in the following snippet, it would necessarily perform much worse than the DSL version.

```
for(unsigned int i=0; i<1000; i++) {
    u.add(v);
    u.add(w);
    u.scale(0.33);
    laplace.multiplyAdd(u, w);
}
```

In fact, this version runs 13 % slower than the DSL implementation because each `add` and `scale` requires a complete iteration over all indices of the vector `u`, whereas in the DSL version, there is effectively only one such iteration to apply all the vector-vector operations at once. To be able to compete with the DSL version, we realize the vector-vector operations in the C++ version via an explicit `for` loop:

```
for(unsigned int i=0; i<1000; i++) {
    for(unsigned int j=0; j<nDoF; j++) {
        u[j] = 0.33 * (u[j] + v[j] + w[j]);
    }
    laplace.multiplyAdd(u, w);
}
```

From Figure 13.1b, we can see that there is virtually no difference in runtime for the DSL and the (second) C++ version (the difference is insignificant with $p \approx 1$). This result is to be expected because the C++ version is exactly what the DSL version is effectively transformed to by the template meta-programming mechanisms.

13.1.2 Performance of a Complex Algorithm Implementation

Besides individual language constructs, we are also interested in assessing the overhead of the Sprat PDE DSL in real world applications. For this

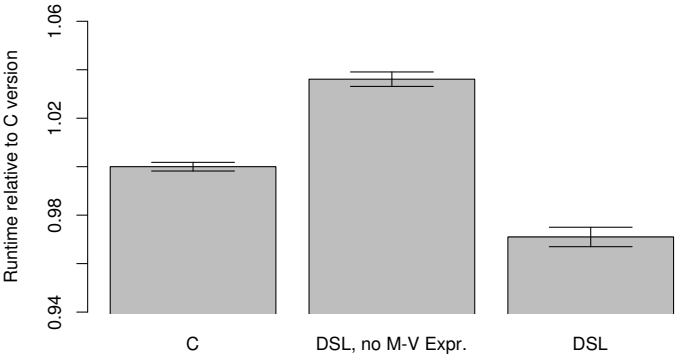


Figure 13.2. Runtime of FCT FEM solver implementations relative to the C version with 95 % confidence intervals for $n = 20$ samples.

purpose, we created three implementations of the complex FCT FEM algorithm presented in Chapter 12. First, a C version that uses a matrix-vector library which internally employs the same compute routines as the Sprat PDE DSL but exposes a traditional BLAS-like interface.[1] Second, a version that uses all abstractions provided by the Sprat PDE DSL with the exception of matrix-vector expressions (thus, no DSOs are applied in this version). Third, an implementation which utilizes all features of the DSL, including matrix-vector expressions and, therefore, applies DSOs.

Figure 13.2 shows the runtime of the three versions relative to that of the C implementation. The high-level abstractions introduced in the DSL version without DSOs incur a significant but comparably small performance loss (less than 4 %). This is probably due to the accumulated negative performance effects of multiple nested iterations over sets (see Section 13.1.1). However, the performance loss of the DSL version without matrix-vector expressions is more than compensated by the DSOs present in the version with matrix-vector expressions (approx. 7 % performance gain compared to the DSL version without DSOs and approx. 3 % gain in relation to the reference C version).

[1] http://www.netlib.org/blas/

13.1.3 Summary and Conclusions

From the results of our benchmark experiments we can conclude that the high-level abstractions of the Sprat PDE DSL do, all in all, *not* affect performance negatively in comparison to plain GPL implementations. The only language concepts that cause a small decrease in performance are iterations over sets (from less than 1 % in synthetic benchmarks up to 4 % in real world applications). The performance of matrix-vector expressions in the Sprat PDE DSL is on par even with hand-optimized C++ code. Furthermore, their performance can be far superior to using a matrix-vector library with a standard BLAS-like interface, which cannot execute multiple arithmetic operations in a single loop (13 % performance gain in our benchmark; this percentage should increase as the number of involved arithmetic operations increases). Additionally, DSOs for matrix-vector expressions enable the Sprat PDE DSL to more than compensate for the small performance losses due to iterations over sets in real world applications (about 7 % gross and 3 % net gain).

13.2 A Qualitative Assessment of the Sprat PDE DSL

In this section, we discuss the results of our interview study that aims at assessing how experts rate the functional and technical quality of the Sprat PDE DSL. We begin by explaining the employed research method in Section 13.2.1. The subsequent sections (13.2.2 to 13.2.5) discuss the main findings of the study grouped by our analysis dimensions. Section 13.2.6 examines threats to the validity of our results. Lastly, Section 13.2.7 provides a summary of and draws conclusions from these results.

13.2.1 Research Method

We use the method of expert interviews (Hove and Anda 2005; Kaiser 2014) for the evaluation of the Sprat PDE DSL because we are interested in acquiring a comprehensive qualitative assessment of the suitability of the language for its intended users. Semi-structured interviews with open-ended questions allow for a broad discussion of different aspects of the DSL with nuanced

feedback from the experts. A quantitative approach, such as controlled experiments combined with feedback questionnaires—as employed for the Sprat Ecosystem DSL in Chapter 14—is not suitable for the evaluation of the Sprat PDE DSL. The reason for this is that the domain of the latter DSL is much larger and more complex and, thus, requires a more open and differentiated discussion.

The group of experts we interviewed consisted of both specialists from the application domain (researchers working with PDE solvers) and professional DSL designers. Together, these experts—as representatives of the state of the art in their respective discipline—can not only judge the quality of the meta-model of the DSL but also our implementation of the language. Additionally, interviewing both groups allows to reveal different stances between them towards what makes a good DSL specifically in the domain of computational science.

Data Collection

We conducted semi-structured interviews with eight experts which each lasted between one and two hours. The interviews were mostly carried out in a one-on-one fashion (with the exception of one interview in which we talked to two experts) in the expert's workplace.

Half of the sample were domain experts for PDE solvers and the other half were professional DSL developers, all based in the area of Kiel, Germany. The domain experts were recruited by contacting professors of relevant research groups at Kiel University and Geomar Helmholtz Centre for Ocean Research Kiel. Candidates for the professional DSL developer group were identified via personal contacts of university employees with people working in local industry. The actual involvement of these persons in DSL development was verified prior to conducting the interviews.

The group of domain experts consisted of one postdoc and three professors from research areas such as numerical analysis, optimization, and ocean modeling (all from different research groups). All of them were proficient in C/C++ with the exception of one interviewee who mostly worked with Fortran but, nonetheless, was able to at least read C++ code. The professional DSL developers all had multiple years of experience with DSL design (one of them even authored a book on MDSE and DSLs) and were IT architects or consultants. Two of the DSL developers have worked with C++

professionally for several years, while the other two mainly used Java but, nevertheless, had intermediate level knowledge of C++.

Two interview guides (one for the domain experts and one for the DSL developers) were developed by the author based on four analysis dimensions deduced from our research question **SERQ4.2** from Chapter 5: *how do experts rate the functional and technical quality of the Sprat PDE DSL?* These analysis dimensions are:

1. Software development in computational science (only for the domain experts)

2. Learning material for DSLs

3. Meta-model and syntax of the Sprat PDE DSL

4. Technical implementation of the Sprat PDE DSL

The two interview guides, which can be found in Appendix D, contain both open-ended and more restricted questions in order to, on the one hand, encourage the interviewees to elaborate freely on different aspects of the DSL and, on the other hand, to acquire relatively succinct quality ratings concerning the language. We tested the guides with peers to make sure that the questions were phrased comprehensibly.

Prior to the interviews, all interviewees were supplied with the source code of, code examples for, and a short written introduction to the Sprat PDE DSL in order to familiarize themselves with the language. Since we expected that not all of the interviewees would find the time to have a closer look at the language beforehand, we started each interview by briefly discussing the fundamental concepts of the language and by going through example algorithms implemented with the DSL. To make sure that the experts themselves had actually worked with the language, each interviewee was asked to solve tasks related to the supplied code examples during the interview (e. g., changing the mesh of a solver or altering the parallelization scheme of an algorithm). In addition to that, the experts on DSL development received an introduction to the implementation of the Sprat PDE DSL, specifically to the embedding of matrix-vector expressions using Boost Proto (cf. Section 8.2).

Analysis Procedure

All interviews were taped and fully transcribed prior to the analysis. The transcripts were analyzed using the method of *qualitative content analysis* (Mayring 2015), for which we chose whole answers as our unit of analysis. Specifically, we employed the technique of *summarizing content analysis*, which consists of four steps:

S1: Paraphrasing/summarizing

S2: Generalization to the target level of abstraction

S3: First reduction (deletion of redundancies)

S4: Second reduction (aggregation)

First, the units of analysis (i. e., the answers) are paraphrased to reduce the volume of the material (S1). The paraphrases are then generalized to the level of abstraction of a set of analysis categories which are used to structure the material (S2). Our initial set of analysis categories stemmed from the analysis dimensions of our interviews and the associated sets of questions. In the third step, analysis units with redundant information are removed from the analysis (S3). The final reduction step (S4) consists in aggregating paraphrases and corresponding interview answers that are related to each other but are spread out in the material (for example in different interviews). Each of these aggregated sets of paraphrases is again summarized by identifying its key message in relation to the analysis categories.

As qualitative content analysis is an iterative process, it is likely that the initial set of analysis categories does not adequately represent the material to be analyzed. In this case, new analysis categories have to be derived from the interview material and the analysis process (S1–S4) has to be repeated using the updated set of categories until the material is found to be represented appropriately. Our final set of analysis categories can be found in Appendix D.

Kaiser (2014) notes that this standardized process ensures that qualitative content analysis is *open* (the analysis categories can be derived from the material), *systematic* (the analysis procedure follows intersubjectively traceable rules), and *theory-based* (the analysis categories are linked to the theoretical background of our research question).

13.2.2 Software Development in the Work of the Domain Experts

The domain experts who were interviewed by us develop software to gain scientific insight

- by numerically validating mathematical propositions about algorithms,
- by implementing models of real world systems (biogeochemical ocean models), or
- by analyzing how to optimize parameters for such models.

While the purpose of their software development efforts is not the software itself but the scientific insights that can be gained from it, three out of four interviewees develop and maintain a single relatively large (up to multiple ten thousand LOC) software library in their research group. Such a software library represents the accumulated research effort of a group insofar as all functionality that is used to explore a new algorithm or a new scientific problem is eventually added to this library. The interviewee whose group does not maintain such a library works in ocean modeling. The models that his team uses have already been implemented by other research groups and the interviewee's group only modifies or combines them in interesting ways but does not implement them from scratch. However, the group does maintain an internal repository in which proven model configurations are collected.

The interviewee from the discipline of ocean modeling furthermore reports that Fortran is the dominant programming language in his field. In his group, however, they use multiple programming languages for different purposes: for example, C++ for writing wrappers and R for data analysis. The libraries of the three other domain experts are all written either in C or C++. The interviewee working in the field of mathematical optimization additionally uses Matlab[2] because, according to him, the language already features useful optimization algorithms.

Most interviewees use popular and stable libraries such as LAPACK[3] and PETSc[4] for the development of software in their research group. However,

[2] http://mathworks.com/products/matlab/

[3] http://www.netlib.org/lapack/

[4] http://www.mcs.anl.gov/petsc/

one expert does not use any external dependencies for the software library of his group at all. He states that he has had bad experiences with external libraries with regard to their portability and, hence, wants to be dependent only on a C compiler and the C standard libraries.

All experts prefer free software with open source licenses in the academic context. Therefore, they consider it good and important that the Sprat PDE DSL uses such a license.

Implementing the Software

When inquired as to how they proceed methodologically when they implement a specific numerical algorithm, all domain experts answered in terms of mathematical concepts: they "write down formulas on paper" or think about mathematical concepts and then either "copy" those formulas "into the computer" or create constructs that "represent" those mathematical concepts in a programming language.

Probably because of this view of source code as a more or less direct representation of mathematical concepts, the experts mostly test their software solely by scientifically assessing its output (e. g., by checking for convergence rates). One interviewee explicitly states that they often do not perform any testing at all. This is because their optimization algorithms are fairly simple and they use Matlab for implementing them, which provides a notation that "practically is pseudocode." Therefore, they just rely on checking whether they have correctly "copied" the algorithm from its paper source.

Documentation material for the software development projects of the domain experts is relatively scarce. One interviewee states that in his group, they do not produce any documentation at all. In two projects, documentation material has been created when it was either mandated by a publisher of an article about the software or when there was a public release of a major version of the software library. However, both of these interviewees are unsure whether the material is still up-to-date for current versions of their library. The fourth expert says that they frequently produce documentation material once a certain development goal has been reached but that this material alone is not enough for new users to get an adequate understanding of the software.

All interviewees agree that new members of their research group cannot rely on documentation material alone but have to study source code and be coached by other people who are already knowledgeable about the code in order to familiarize themselves with the software. One expert notes that even he himself frequently has to read through old source code again to regain knowledge of how it functions.

All things considered, we can conclude that the way the interviewees develop scientific software is consistent with the findings of our literature review in Chapter 6. This confirms that the development of the Sprat PDE DSL was driven by correct assumptions concerning the characteristics of scientific software development.

13.2.3 Learning Material for DSLs

In this section, we discuss what kind of learning material the domain experts wish for in order to familiarize themselves with the Sprat PDE DSL and how their requests differ from the ideas of the professional DSL developers.

All the scientists view commented example programs as the key element for the introduction to a new DSL for three reasons:

1. Example programs allow to judge quickly whether the DSL is suitable for the intended application and whether the scientists can imagine to work with it ("Is the code compact? Does it seem intuitive? Do I understand it?").
2. Examples make it easy to learn how a typical DSL program is structured.
3. Examples can be used as a basis for own programs without investing much time in reading other documentation artifacts.

The example programs that are bundled with the Sprat PDE DSL are viewed as sufficient for this purpose by all domain experts.

Complementary to a set of commented examples, the interviewees would like to have a summary of the key concepts of the language which answers questions such as: What is the exact scope of the DSL? What is the performance of matrix-vector expressions? How is data managed with the language? Additionally, they would like to have a specification of the data structures and interfaces of the DSL in order to understand how (possibly

already existing) GPL code can be combined with the constructs of the language.

The DSL developers also suggest a combination of a summary and a complete language reference as learning material. However, they do not mention the importance of complete code examples (they rather seem to think of example snippets embedded into written text) and they generally put much more emphasis on the reference document than the scientists do: three out of four interviewed DSL developers name the reference first and talk about introductory material only when asked about it. For them, the basis of the reference document should be a formalized meta-model or the abstract syntax of the DSL, which is supposed to quickly give a top-down overview on the language. One of the interviewees mentions the reference of the Swift programming language as exemplary in this respect (Apple Incorporated 2015). This reference is structured around grammar rules that are grouped according to which aspect of the language they belong to (expressions, types, etc.).

From the interviews, it can be seen that the domain scientists seem to favor a more practical and pragmatic approach to learning a DSL for computational science than DSL designers might think. The scientists emphasize the importance of complete documented code examples and they are interested in a reference only as a second step when it comes to more technical aspects of the implementation. In consequence, the utility of a formalized meta-model and lengthy grammar rule descriptions seems questionable for such an audience. When developing DSLs for computational scientists, DSL designers should reflect on their generally more formal and systematic approach to introducing others to such a language.

13.2.4 Meta-Model and Syntax

This section reports how the domain experts rate the meta-model and the syntax of the Sprat PDE DSL as well as the overall usefulness of the language for their scientific discipline. Input from the experts on DSL development is included complementarily where appropriate, for example, in the context of the discussion about how much program structure the DSL should prescribe.

Conformity and Orthogonality of the Meta-Model

All domain experts assess the meta-model of the Sprat PDE DSL as covering all common aspects of (FEM) PDE solvers with a few exceptions, as discussed below. They also stress that the language organizes the meta-model elements in a logical and intuitive way with clear boundaries of the domain concepts.

Some suggestions for additional concepts to be included in the Sprat PDE DSL are made but only one of them actually affects the meta-model: this is the *Implicit_Matrix* meta-type, which is already part of the presentation of the meta-model in Section 8.2.1 of Chapter 8. One interviewee who regularly works with preconditioners for solving linear equations systems suggests the addition of such matrices, which are not stored explicitly (i. e., one cannot access individual entries but it is known how to apply the whole operator to a vector).

Two other suggestions for additional features for the Sprat PDE DSL aim at special-purpose mesh types. One interviewee is interested in moving meshes to, for example, model sea level changes. Another expert names hierarchical meshes for multigrid methods (Hackbusch 1985). Such special-purpose mesh types can be added by users of the DSL without affecting the meta-model of the language by simply filling out a code skeleton.

All in all, the experts describe the meta-model of the Sprat PDE DSL as conforming to the target domain and the modeled meta-classes as orthogonal.

Quality of the Syntax

Regarding the simplicity of the syntax of the Sprat PDE DSL, we receive the following feedback from the domain experts:

1. The syntax is "very accessible" and "if I had to do it, I would probably program it in a similar way. I think it is good."
2. It appears "clear and intuitive." Especially the assembly of operators is "very compact" and the iteration of degrees of freedom is "realized well" and "quite intuitive."
3. The syntax enables a clear correspondence between a mathematical algorithm description and the implementation.

4. The interviewee who did not have much experience with C/C++ nonetheless was able to solve all the programming tasks with the DSL (as were the other experts) and states that for him, the syntax is "clear enough" to work with.

The DSL is generally considered to be "relatively quick to learn" and, again, the importance of examples for this learning process is highlighted:

> "I think, one could work with it immediately. If you are provided with an example like this, I think, then, it is no problem to change it."

With regard to the naturalness of the syntax, the experts all agree that the syntax is "very close" to the one used in the discipline for expressing algorithms. One interviewee mentions that it "looks a bit like pseudocode." Additionally, all experts easily recognize the implementation of the Conjugate Gradient (CG) method in one of the source code examples provided with the DSL:

> "I mean, if you look at this, I would say that everybody who knows what a CG method is, recognizes that this is a CG method. Insofar, I would say that this is a great success."

The naturalness of the syntax is a crucial property of the Sprat PDE DSL especially because, as we have seen above, many computational scientists verify code solely by checking its correspondence with published algorithm descriptions (at least as long as "plausible results" are produced by the software). Of course, it becomes easier to check for this correspondence the more similar the syntax of the code and of the description are.

The abstractions provided by the Sprat PDE DSL to encourage defensive programming by making it easier to check assertions on vector-valued expressions are described as helpful and adequate by the interviewees.

One of the two experts on DSL development who have worked with C++ for several years does *not* perceive the Sprat PDE DSL as a "foreign body" in C/C++. He regards the syntax as appropriate because the DSL picks up familiar language constructs (such as expressions, assignments, for loops, etc.) and develops them further in a consistent way. Yet, the other of those two experts suggests to adapt the syntax of the DSL more to C++, for example, by using stream operators to "throw" degrees of freedom "into matrices" in order to represent the application of operators/matrices to

vectors. This suggestion, however, is not in line with the requirements for the language for two reasons. First, the DSL does not only target C++ but also C developers and there are, for example, no stream operators in C. Second, since algorithm developers are highly familiar with concepts such as matrix-vector multiplication to represent the application of operators, it does not seem useful to introduce unheard-of concepts to make the language feel more like "normal C++ code."

Altogether, we can conclude that the syntax of the Sprat PDE DSL receives positive feedback, particularly with respect to its simplicity, learnability, comprehensibility, and naturalness.

Prescribed vs. Flexible Program Structure

One of the experts in DSL development suggests to enforce a common block structure for all Sprat PDE DSL programs. His motivation is to make sure that "as little nonsense as possible happens." From the findings of our literature review in Chapter 6 (specifically Section 6.2.2 b)), however, we conclude that such an approach is likely to be met with reservation by computational scientists. When two of the domain experts are confronted with this idea in two subsequent interviews, they both express their fear that a prescribed code structure would take away too much control over their program and would make integration with existing code harder.

While software engineers working in the IT industry generally seem to focus their attention on consistency among solutions to facilitate reusability and maintainability, computational scientists favor loose structures that allow them to experiment and quickly obtain results. In order to be accepted by the scientists, a DSL for computational science (and especially for the HPC community) has to be pragmatic about the rigidity of prescribed structures and the level of abstraction it introduces. If one does not make concessions to the need of computational scientists to freely experiment with a DSL, the language simply will not be adopted.

Maintainability: Experimenting with Algorithms

One of the aims of the Sprat PDE DSL is to allow computational scientists to easily experiment with PDE solver algorithms, for example, by changing the dimensionality of the problem or by using another type of mesh without

having to change the algorithm itself. The domain experts report that this type of experimentation with algorithms is frequently part of their work and that code which is difficult to maintain can be a major obstacle to carrying out this task.

One domain expert mentions that the possibility of quickly modifying aspects of the spatial discretization in FEM solvers (such as the element type) without having to change the actual solver algorithm is of considerable value to him because it allows to efficiently test theoretical propositions numerically. According to him, the Sprat PDE DSL is "very dynamic" with respect to this. Another domain expert says that he frequently has to experiment with different mesh types and notes that the separation between the topology and the algorithm in the DSL is "easier than with our means." The other domain experts and—as far as they are able to assess this—also the DSL developers express that they, too, think that the separation between the topology and the algorithm is implemented successfully in the DSL, which leads to well-maintainable application code.

Usefulness of the DSL

In the end, we are interested in how the researchers rate the Sprat PDE DSL with regard to its potential benefits for the domain of (FEM) PDE solver development. Especially those interviewees who are specialized in analyzing numerical methods (and, thus, work with algorithms in a less "applied" way) emphasize that the DSL is a useful addition to the field of applied mathematics.

When asked whether they want to use the DSL themselves, the experts give the following replies:

1. The expert can well imagine to use the DSL for his own developments but he would not use it for the software library that is developed in his research group because he refuses to include any dependencies in its development (see above).
2. For his field of research (optimization of PDE-based models), the interviewee rather uses PDE solver DSLs with a higher level of abstraction that directly allow him to specify *which* PDEs to solve without having to describe *how* to solve them. Apart from that, he could, in principle, imagine using the Sprat PDE DSL but has some reservations concerning

its maintenance. He suggests to contribute with the DSL to projects such as deal.II[5] or FEniCS (Logg et al. 2012) by incorporating the language into their software libraries. In this way, the language would serve a large community which is able to support the DSL over a long period of time.

3. The expert wants to try out the DSL in his professional software development.
4. The interviewee is interested in using the DSL in the context of master and Ph.D. theses which deal with novel problems that require a lot of experimentation:

> "The gain here is that [the algorithm] can be modified with relative ease [...] With a novel problem, for example, the mesh has to be altered continuously [...] For this purpose, I think, it [(the DSL)] would be very very good."

Overall, the statements of the interviewees show that the Sprat PDE DSL succeeds in capturing the most relevant domain concepts and presents them in a natural concrete syntax that makes the language a useful tool for the domain of PDE solver development. In particular, it is highlighted by the experts that the language simplifies implementing algorithms in a well-maintainable way and makes it easy to experiment with them.

13.2.5 Technical Implementation

This section discusses the interview results with a focus on the technical implementation of the Sprat PDE DSL.

C++ as the Host Language

All domain experts agree that C++ is a good choice for the host language of the Sprat PDE DSL because, in their opinion, it will probably be the standard language in the HPC community for years, if not decades, to come. One interviewee mentions that future alternatives to C++ will likely be "high-level languages" that generate C/C++ code (i. e., external DSLs). Other approaches will probably not be accepted by the community.

[5] https://www.dealii.org

Dependencies of the DSL

Dependencies are often viewed sceptically by developers of scientific software (cf. Section 6.2.3 c)). One of the domain experts that we interviewed does not use any dependencies other than standard libraries for the software developed in his research group. Therefore, he perceives the dependency of the Sprat PDE DSL on the Boost libraries[6] as undesirable. The other experts do not view any of the dependencies of the DSL (apart from Boost, these are OpenMP and MPI) as something that would deter them from using the language. However, they stress the importance of free open source licenses for any dependencies.

Maintainability of the DSL Implementation

To embed lazily evaluated matrix-vector expressions into C++, the Sprat PDE DSL uses template meta-programming, as described in Section 8.2.2. Typically, this programming technique requires implementing large numbers of operator overloadings, which can be tedious and hard to maintain. Therefore, we use Boost Proto to handle these tasks for us.

None of the interviewees has worked with Boost Proto before. However, having inspected the implementation of the embedding of the Sprat PDE DSL, the experts on DSL development say that Boost Proto provides "suitable structures" that make the DSL implementation "elegant" and "intuitively readable." They do not think that it would be difficult for a trained software engineer to maintain the implementation.

One of the domain experts who also went through the code of the DSL embedding mentions that the "arithmetic approach" of the language towards matrix-vector expressions is what makes not only the DSL itself but also its programs well-maintainable: new types of operators or vectors can be added to the language without any changes to algorithms implemented with the DSL (except, of course, changes to the type of variables in order to use the new operator or vector types).

[6] http://www.boost.org

Internal vs. External

All except one of the domain experts favor an internal DSL over an external language for the level of abstraction that the Sprat PDE DSL aims for. They name several reasons for this:

1. An external DSL with a completely new syntax could be in conflict with concepts of programming languages such as Fortran and C that the experts have internalized. This could lead to confusion and an increased number of errors.
2. One expert already has negative experiences with external DSLs for implementing numerical algorithms. He says that such languages are often very good for the narrow domain they are designed for but commonly lack support for "everything else" in the large domain of computational science. In contrast to this, with an internal DSL, the user can seamlessly integrate DSL code with GPL code.
3. Another interviewee states that he does not believe that the increased flexibility of an external DSL offsets the added technical complexity of additional compiler runs and the need for other external tooling.

In spite of these reservations, none of the interviewees excludes the use of an external DSL with a code generator categorically, as long as this code generator is portable and available under an open source license.

The single domain expert who would actually prefer the Sprat PDE DSL to be an external language works with ocean models. He reports that in this scientific field, researchers are sometimes confronted with the problem of not being able to reproduce older simulation results once hardware platforms and compiler vendors/versions change. Therefore, he is interested in archiving source code that is as low-level as possible (e. g., already preprocessed Fortran code). With an internal language which uses template meta-programming techniques, such as the Sprat PDE DSL, this is not possible. Template meta-expressions are processed during compilation without yielding any intermediate low-level C++ code that could be archived. This lack of intermediate code also makes debugging of matrix-vector expressions hard because one cannot see what is actually executed during the evaluation of such an expression.

Since nobody excluded the use of an external DSL completely, a compromise would be possible: one could implement the Sprat PDE DSL as

a language extension to C or C++ using a framework such as JetBrains MPS (see Chapter 2). In this way, the requirements of the proponents of an internal solution and of those preferring an external solution could both be met. All the features of the C/C++ GPL would be present while readable intermediate C/C++ code could be generated. It would even be possible to incorporate *interactive model-based compilation* techniques into the tooling that allow to trace in detail the transformations applied to DSL models during code generation (Motika et al. 2014). However, further research has to be conducted in order to evaluate whether such an approach would actually be accepted in the community.

Altogether, the experts are content with the technical implementation of the Sprat PDE DSL. The choice of C++ as the host language and of Boost Proto as a means for implementing matrix-vector expressions is welcomed (except by the expert who refuses to use any dependencies at all). The implementation of the DSL using Boost Proto is described as well-maintainable. Potentially, the DSL would be even more useful for and better accepted by the domain experts if it was implemented as an external language extension using a framework such as MPS.

13.2.6 Validity of the Results

One threat to the validity of results from expert interviews is whether the insights gained from interviewing a relatively small sample can be generalized. Dorussen et al. (2005) find that the single most important factor for the validity of results derived from such interviews is the quality of the choice of experts: if they are very knowledgeable in their field, their statements will represent the state of the art in the discipline they represent and can, thus, be generalized. Our selection process of interview candidates (cf. Section 13.2.1) ensured that we only addressed persons in positions that require them to be experts in areas relevant in the light of our research question.

In order to make sure that the interviewees can speak freely, we guaranteed them anonymity. It is our impression that the experts indeed communicated openly.

Another risk to the validity of our findings is whether the time the experts had to familiarize themselves with the Sprat PDE DSL was long enough for them to assess the language adequately. Each interview included

an in-depth introduction to the DSL and, after that, all experts where asked whether they felt comfortable evaluating the language, which everybody confirmed.

In addition to the threats discussed so far, there are two critical aspects of validity that have to be considered for results based on interviews in general (Maxwell 1992):

1. *Descriptive validity*, which refers to the factual accuracy of the material that is used as the basis for the analysis of the interviews. To minimize the risk of distorting the actual content of the interviews, we taped all of them instead of taking minutes and fully transcribed each conversation.
2. *Interpretative validity*, which concerns whether the statements of the interviewees are interpreted correctly during the content analysis. In our case, the interviewees spoke very objectively, which reduced the room for interpretation. Additionally, the iterative nature of the method of qualitative content analysis ensured that in each iteration, we tested our current understanding and coding of the material again on the whole text corpus.

13.2.7 Summary and Conclusions

Overall, the interviewees evaluate the Sprat PDE DSL as being a valuable addition to the field of applied mathematics because it simplifies implementing algorithms in a well-maintainable fashion. The DSL makes it easier to experiment with these algorithms as well as to check whether they have been copied correctly from a publication or handwritten notes (which is often the only form of verification that takes place). According to the experts, the meta-model of the language contains all necessary abstractions for building (FEM) PDE solvers in a logically structured way. Furthermore, the experts characterize the syntax of the DSL as simple, natural, as well as easy to learn and read.

Also the technical implementation of the DSL receives generally positive feedback: it is considered to be well-maintainable and its host language as well as its other dependencies are perceived as good choices. Regarding whether to implement the language as an internal or external DSL, all domain experts agree that it is important to have all features of a GPL accessible to the user. However, the lack of generated intermediate code

with an internal DSL could be problematic for some users. Implementing the Sprat PDE DSL as an external language extension to C or C++ and generating readable C/C++ code could prove to be a viable alternative to embedding it into C++. This approach, however, requires further investigation especially with regard to its actual acceptance among computational scientists.

Beyond the evaluation of the Sprat PDE DSL, our interview study allows us to highlight some differing stances towards DSL design for computational science between DSL developers and computational scientists working in HPC:

1. For learning a new DSL, the domain experts wish for a practical approach and view commented example programs as the key resource. In contrast, the professional DSL developers tend to focus more on exhaustive language references constructed around a formalized meta-model.
2. Regarding the rigidity of DSL program structure, some DSL developers express the opinion that the users should be prevented from "doing nonsense" by prescribing the structure of DSL programs as much as possible. In opposition to that, the domain experts want to have full control over the layout of their programs and fear that such rigidity could lead to unexpected problems in the future.

We argue that it would be beneficial for the acceptance of DSLs in computational science (and specifically in HPC) if DSL developers were more aware of these different perspectives and were more open towards a development culture that favors loose structures and experimentation over consistency. For this purpose, more research is needed in order to identify further areas in which DSL developers might have misconceptions about the requirements of computational scientists.

Chapter 14

Evaluation of the Sprat Ecosystem DSL

This chapter presents the results of an online survey that includes controlled experiments to evaluate the suitability of the Sprat Ecosystem DSL for ecologists. The main outcomes of our study are:

1. Accuracy and efficiency are 62 % and 182 % higher for typical tasks in the implementation and maintenance of ecosystem simulations if you compare the Ecosystem DSL to a GPL-based solution.
2. The Ecosystem DSL receives high ratings from its users with regard to quality, which indicates that it successfully captures the essential domain concepts in a concise and accessible syntax.
3. Most users with no or very moderate Java skills are themselves able to carry out basic maintenance tasks concerning the Ecosystem DSL infrastructure.

These results are in line with findings from experiments conducted by Kosar et al. (2012), who report that textual DSLs—when compared to GPLs—increase the accuracy and efficiency in program comprehension for "technical" domains (i. e., those domains which are more affine to programming). With our experiments, we extend these findings to the non-technical domain of ecosystem modeling and provide additional empirical evidence for the long-standing hypothesis that:

> "Because of appropriate abstractions, notations and declarative formulations, a DSL program is more concise and readable than its GPL counterpart. Hence, development time is shortened and maintenance is improved." (Consel and Marlet 1998)

For the structure of this chapter, we follow the guidelines for reporting experimental results in software engineering from Jedlitschka et al. (2008). Thus, Section 14.1 discusses the design of our study and introduces the detailed hypotheses to be tested. In Section 14.2, we describe the survey results and the statistical procedures used in their analysis. The discussion of results in Section 14.3, which includes an analysis of threats to validity, is followed by concluding remarks in Section 14.4.

The full material used for the survey (including all task descriptions and questionnaires) can be found in Appendix C. The anonymized raw data and analysis scripts are available online (Johanson 2015c).

14.1 Study Design

In this section, we describe the design of our study that addresses the software engineering research questions **SERQ4.3** and **SERQ4.4** from Chapter 5. The questions are answered by relying on data from both controlled experiments and user feedback.

14.1.1 Goals and Research Questions

To evaluate the Sprat Ecosystem DSL, we investigate to which degree the DSL meets the needs of and is appropriate for ecologists. We formalize this main evaluation goal (EG) utilizing the goal definition template from the Goal/Question/Metric (GQM) method (Basili et al. 1994; Solingen and Berghout 1999) as follows:

EG: Analyze the Sprat Ecosystem DSL for the purpose of evaluation with respect to its suitability from the viewpoint of ecologists in the context of the implementation and maintenance of marine ecosystem simulations.

In order to derive an experiment procedure from this main goal, we explicate it by defining three evaluation research questions (ERQs).

- **ERQ1:** How accurately and how efficiently do ecologists carry out typical tasks in the implementation and maintenance of marine ecosystem simulations using the Sprat Ecosystem DSL compared to using a GPL?

- **ERQ2:** How do ecologists judge the quality of the Sprat Ecosystem DSL compared to a GPL with regard to commonly accepted quality standards for programming languages and APIs for domain-specific tasks?
- **ERQ3:** Are most users with no more than very moderate Java skills able to carry out basic maintenance tasks concerning the Ecosystem DSL infrastructure?

These research questions and the theoretical constructs involved are operationalized in Section 14.1.4, where we formulate our hypotheses to be tested. In this section, we also complete the GQM tree by listing the metrics we employ. Note that we use the GQM method only in order to achieve a clear top-down presentation and, therefore, adopt only some aspects of the method.

14.1.2 Participants

According to our evaluation goal, we intend to analyze the Sprat Ecosystem DSL from the perspective of ecologists. In order to recruit suitable ecologists for our survey, we sampled web pages of research groups working in the field of ecology and specifically marine ecology. From each research group we usually contacted a single person, requesting to forward our call for participation in the survey within their group. Additionally, we posted the link to the survey on a message board of the International Council for the Exploration of the Sea (ICES).[1]

In total, we approached 16 research groups within a sampling period extending over about five weeks. The groups were located in Germany, Canada, the USA, and France (with descending frequency).

This sampling strategy ensured that we would only contact domain practitioners with at least a graduate degree or being in the process of finishing such a degree (which we additionally verified with a questionnaire). In order to check whether the participants were actually working in the field of (marine) ecology, we collected the discipline of their highest academic qualification as well as their current occupation.

In total, 59 subjects took part in the survey, out of which 40 completed it (32 % drop-out rate). Our sampling design implies that we cannot give

[1] http://www.ices.dk

reasonable estimates of the answer rate. This is because we cannot be sure whether the persons we contacted in each research group actually forwarded the message to their colleagues.

The participants were informed about the purpose of the study and were guaranteed complete anonymity. They received no monetary compensation and participation was completely voluntary. The only incentive we gave was the reminder that their participation in the survey would support the development of DSLs for their field.

14.1.3 Experimental Material and Tasks

In the following, we describe the data collection instruments and the tasks provided to the subjects. All materials used in the survey are included in Appendix C.

Background Questionnaire The survey begins with a questionnaire that gathers information about the subjects' academic background and their prior experience with programming. The questions (13 in total) are a mix of open-ended questions (e. g., which programming languages they have experience with) and closed questions utilizing Likert scales (Likert 1932). Throughout the survey, we mostly use uneven scales with six possible answers to force choice between dis-/agreement. This helps to overcome the well-known response bias called "tendency towards the center" (Korman 1971). The tendency towards neutral answers would be problematic for this survey as we are interested in evaluating—and thus in rating—the Sprat Ecosystem DSL.

Comprehension Tests The survey contains two tasks to test program comprehension with either the DSL or a GPL. Like the tests employed by Kosar et al. (2012), our tasks focus on the implementation and maintenance phases of the software development life cycle as these phases are most demanding with regard to program comprehension activities (Hevner et al. 2005; Webb Collins et al. 2008). In addition, the exercises are designed to cover all features of the Sprat Ecosystem DSL.

For the GPL, we chose C++ for two reasons. First, C++ is prevalent in computational science. Second, the Sprat Ecosystem DSL generates C++

code, which we could use as a natural starting point for an alternative implementation (for further details see Section 14.3.4).

The first task to be solved by the subjects is concerned with specifying parameters for a fictitious ecosystem simulation. The participants are given tables of parameters that consist of name/value pairs with measurement units and a short description of how to implement such a name/value pair with the DSL or the GPL, respectively. When solving the C++ version of this task, the subjects have to perform simple unit conversions for some parameters (and are warned about this) as C++ does not feature the concept of units and their conversion.

The second task deals with the recording and aggregation of data during a simulation run. The participants have to implement "recorders" for certain aggregated quantities (such as the total wet biomass of the system). For the GPL, we describe an API that allows to define such recorders (see Section 14.3.4 and Appendix C).

For each task, the subjects receive a short introduction with an example program that covers how to solve the task with the respective programming language. To start an exercise, the participants have to click on a button after which they are shown a text editor component.[2] This component, in which the subjects enter their solution, initially contains the example program snippet given to them in the introduction to the task. The task description and the introductory text remain visible the whole time.

Below the text editor component, the participants find a small timer that displays a suggested maximum time to be spent on each exercise (four minutes for every task). The subjects can, however, choose to ignore this countdown and continue working after it expires. We included this counter to prevent participants who struggle with the tasks from dropping out of the survey due to frustration about how much time they have to spend on it.

In order to form a well-founded judgment about their experience with the DSL and the GPL, the participants have to be able to know whether their solutions are correct or not. Therefore, while solving the exercises, the subjects can, at any time, click on a button to check their current attempt to a solution for errors.

[2]For this purpose, we use CodeMirror (`http://codemirror.net`), for which we implemented a Sprat Ecosystem DSL syntax coloring plug-in.

For each exercise we measure *correctness* and *time spent*. Here, time spent means the time from starting the exercise (by clicking the button) to submitting the solution *while the web page with the editor component is active/in focus*. With the latter restriction, we correct for some of the possible interruptions of the subjects (such as receiving and reading an e-mail while solving the task).

Comprehension Feedback Questionnaire After completing both tasks with both the DSL and the GPL, the subjects are asked to rate the programming languages with respect to certain quality characteristics:

1. Level of abstraction
2. Simplicity of use
3. Ease of comprehension
4. Absence of technicalities from the syntax
5. Maintainability of solutions

These categories are derived from commonly accepted quality attributes for programming languages and APIs for domain-specific tasks (Kolovos et al. 2006). Again, the questionnaire uses forced-choice six-point Likert scales to measure the responses.

In addition to the closed questions discussed above, the feedback questionnaire contains two open-ended questions allowing the subjects to elaborate on difficulties with the Sprat Ecosystem DSL and suggestions for improving it.

Infrastructure Maintenance Test In this last test, the participants are supposed to carry out the relatively simple maintenance task of adding a new attribute for fish species to the Sprat Ecosystem DSL. For this, they are supplied with the central configuration file of the Ecosystem DSL, which is a Java source code file with about 130 LOC. This Java code intentionally does not adhere to standard object oriented design principles. In contrast, it employs a style that is similar to forming a small embedded DSL that targets programming novices who are not familiar with Java. To solve the task, the subjects have to scan through the code to find and adapt lines that look like:

```
SPECIES_ATTRIBUTES.add(new SpratAttribute("ParameterName", MASS));
```

The structure of the test page is the same as the one of the comprehension tests described above.

Maintenance Feedback Questionnaire This feedback questionnaire is concerned with assessing how the participants perceive the simplicity of the Java code during the test described above. Additionally, it measures the level of Java expertise.

14.1.4 Variables and Hypotheses

To answer **ERQ1** and **ERQ2**, we have to manipulate whether the subjects use the DSL or the GPL for the tests and questionnaires mentioned above while ensuring that the participants fit the intended user profile of the Sprat Ecosystem DSL. Additionally, we control for the experience with programming in general and with certain programming languages in particular.

For **ERQ3** to be answered, we have to control for the users' Java expertise.

We therefore introduce the following in-/dependent variables:

- Independent variables (controlled):
  - Academic discipline
  - Academic degree
  - Academic occupation
  - Programming experience in general
  - C++ experience
  - Java experience
- Independent variables (manipulated):
  - Programming language (DSL/GPL treatment)—within-subjects variable
  - DSL/GPL order
- Dependent variables:
  - Correctness (program comprehension)
  - Time spent (program comprehension)
  - Efficiency (program comprehension, correctness divided by time spent)

Table 14.1. GQM tree for the evaluation of the Sprat Ecosystem DSL.

Goal	Question	Metric
EG	**ERQ1**	Correctness in percent ($\mathbf{H1}_1$)
		Efficiency ($\mathbf{H2}_1$)
	ERQ2	Rating score ($\mathbf{H3}_1$)
	ERQ3	Correctness ($\mathbf{H4}_1$)

- Quality attributes: level of abstraction, simplicity of use, ease of comprehension, absence of technicalities, maintainability of solutions (see above)
- Correctness of DSL infrastructure extension

To allow for statistical hypothesis testing, we formalize **ERQ1-3** into the following set of four directed hypothesis pairs (for the complete GQM tree, see Table 14.1):

Correctness

$\mathbf{H1}_0$: The mean correctness for program comprehension tasks is less or equal for the DSL than for the GPL treatment.

$\mathbf{H1}_1$: The mean correctness for program comprehension tasks is greater for the DSL than for the GPL treatment.

Efficiency

$\mathbf{H2}_0$: The mean efficiency for program comprehension tasks is less or equal for the DSL than for the GPL treatment.

$\mathbf{H2}_1$: The mean efficiency for program comprehension tasks is greater for the DSL than for the GPL treatment.

Quality

$\mathbf{H3}_0$: There exists a quality attribute for which the DSL mean rating is less than or equal to the GPL mean rating.

$\mathbf{H3}_1$: For all quality attributes the DSL mean rating is greater than the GPL mean rating.

Maintainability by non-professionals

H4$_0$: Less than 90 % of the users with Java skills less than or equal to "beginner" are able to carry out basic maintenance tasks of the Sprat Ecosystem DSL.

H4$_1$: At least 90 % of the users with Java skills less than or equal to "beginner" are able to carry out basic maintenance tasks of the Sprat Ecosystem DSL.

Our operationalization of "most users" (**ERQ3**) for **H4** as 90 % is, of course, contingent. However, we believe this to be in accordance with common sense.

The reason for choosing directed hypotheses for **H1**-**H3** is motivated by the rationale expressed throughout the literature that the abstract, domain-specific notation of a DSL should be easier to work with than a GPL. Therefore, for a programmer who is familiar with the concepts of the respective domain, program comprehension should be higher with a DSL than with a GPL (Kosar et al. 2012).

14.1.5 Experimental Design

Our main experiment regarding program comprehension and quality (**H1**-**H3**) features a single-factor (DSL/GPL treatment) within-subject design. The main reason for choosing a within-subject over a between-subject design is that in this way, each participant acts as their own control with regard to programming experience and domain expertise. This advantage, however, entails the drawback of possible intra-subject learning effects. Since we give each participant the same tasks to be solved with both the DSL and the GPL, it is likely that they acquire language-independent knowledge about the task that allows them to solve it better on their second encounter with it. This potential bias can be mitigated effectively by randomizing the order of treatments (Jones and Kenward 2014). Thus, we determined the order of the programming languages for the tasks by chance for each participant.

The DSL infrastructure maintenance experiment to test **H4** does not employ a standard modern experiment design. Thus, we use the term "exploratory" experiment. Although this experiment does not have an independent variable that we manipulate, it follows a clear experimental set-up and allows for valid hypothesis testing. With this design we will, of

course, not be able to compare our DSL extension interface implementation to any other method nor establish a sound cause-effect relationship between our implementation approach and the participants' ability to carry out simple maintenance tasks. The experiment is meant to be a first step to explore whether experts from non-technical domains are able to extend DSLs developed for them.

14.1.6 Procedure

The data collection phase of our online experiment lasted for about five weeks and was conducted in January and February 2015. The order in which the data collection instruments were presented during the experiment corresponds to the order in which they are discussed in Section 14.1.3.

The survey does *not* include any dedicated learning phases for the subjects to familiarize with the programming languages to be used to solve the tasks. Instead, each task is preceded by a very concise description of how to employ the corresponding language for the given task. These descriptions heavily rely on code examples in order to account for the fact that the targeted users of the Sprat Ecosystem DSL potentially only have relatively moderate programming skills. Additionally, the very short introductions allowed us to make sure that their informational content is equal for both the DSL and the GPL.

Because of using directed hypotheses for the comparison of the DSL and the GPL, we do not have to take measures to counteract different levels of expertise with the languages: it is highly unlikely that any of the participants have previously been exposed to the Sprat Ecosystem DSL; if they have prior knowledge in C++, this would only increase the cogency of our alternative hypotheses.

A summary of the whole experimental setup is depicted in Figure 14.1.

14.2 Analysis and Results

This section discusses the analysis procedure for the survey results and reports on descriptive and inferential statistics. As the threshold for statistically significant results we used $\alpha = 0.05$. All analyses were performed

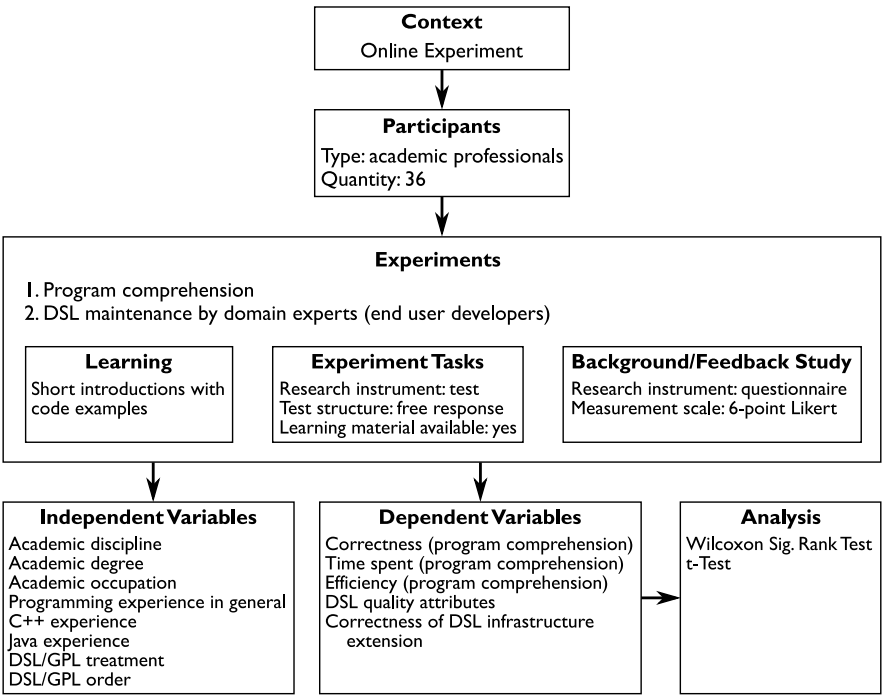


Figure 14.1. Overview on experimental setup.

using *R 3.1.1*[3] and our R analysis scripts as well as the raw data can be obtained online (Johanson 2015c).

In total, there were 59 participants, out of which 40 subjects completed the entire survey. This corresponds to a drop-out rate of approximately 32 %. We discarded all incomplete responses and further filtered the data set by applying the following two criteria:

1. Click-through: we discarded result sets from subjects spending less than 30 seconds on any task of the experiment as those were assumed to not really have tried to solve the task. Number of discards: 3

[3] `http://www.r-project.org`

Table 14.2. Academic disciplines of the participants.

Discipline	Number	Fraction (%)
(Marine) Biology	14	38.9
(Marine) Ecology	8	22.2
Fisheries Science	4	11.1
Biological Oceanography	4	11.1
Marine Environmental Sciences	3	8.3
(Applied) Mathematics	3	8.3

Table 14.3. Occupations of the participants.

Occupation	Number	Fraction (%)
Ph.D. Student	24	66.7
Postdoctoral Researcher	8	22.2
Master Student	3	8.3
Professor	1	2.8

2. Wrong discipline: we discarded the answers from participants whose academic discipline was not related to ecological modeling. Number of discards: 1

This filtering resulted in $n = 36$ result sets that were further analyzed. About 52.8 % of these participants were randomly selected to solve each task first using the DSL.

The academic disciplines and current occupations of the subjects are summarized in Tables 14.2 and 14.3. Most (61.1 %) reported to be working in (marine) biology or, more specifically, (marine) ecology. The remaining participants pursue careers either in a multidisciplinary subject which involves ecology such as fisheries science or they approach ecological modeling with a background in mathematics.

As can be seen from Figure 14.2, most participants have medium-level experience with programming—though there are two subjects who state that they have never written a program before. Average experience is lower with C++ and Java. The most-named programming languages that the users have used so far are: R (named 25 times), Matlab (19), C or C++ (12),

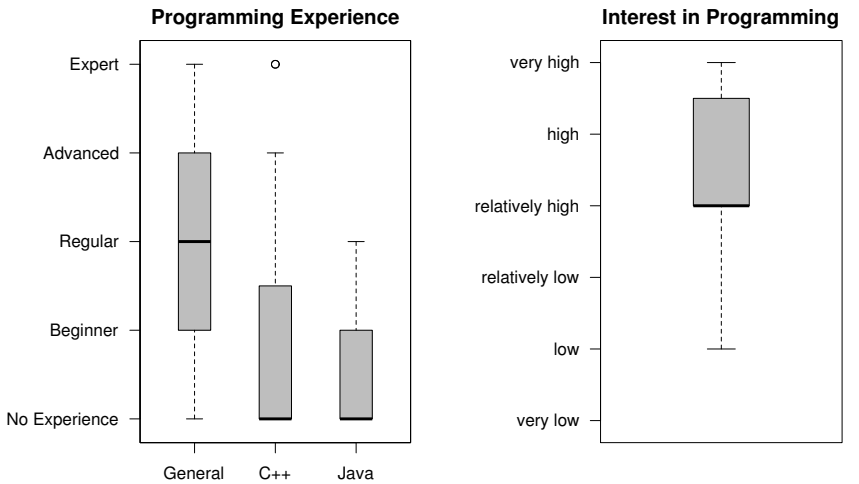


Figure 14.2. Programming experience and interest in programming of the participants. Experience levels: 1. No experience (never written a program, difficulty with reading), 2. Beginner (problems writing easy programs, can read), 3. Regular (can write basic programs, no problems reading), 4. Advanced (can write complex programs), 5. Expert (years of experience).

Python (9), and Fortran (8). In this context it is noteworthy, that the interest in programming among the subjects is on average much higher than their expertise in this matter.

The following sections are concerned with testing the hypotheses **H1**-**H4** statistically. For this purpose, we report the results of the paired difference t-test (Student 1908), of Wilcoxon's signed rank test (Wilcoxon 1945), and of the Shapiro-Wilk test (Shapiro et al. 1968) for the sample in question. Note that we perform one-tailed tests (for t-test and Wilcoxon) because our hypotheses are directed. Also note that we can assume homogeneous variances for the t-test because of our paired study design.

It is common practice to employ a "normality test"[4] like Shapiro-Wilk to decide whether to use a parametric (t-test) or a non-parametric (Wilcoxon)

[4]These tests are all formulated to test *against* normality (the alternative hypothesis is that the sample is *not* normal). This means that the test, strictly speaking, can never show that a

test. One could argue, however, that the central limit theorem ensures that for samples large enough (typically $n > 30$ is assumed), the mean of the samples is guaranteed to be approximately normal (Bortz and Schuster 2010). This means that in our case the assumptions for the t-test would always be met. Luckily, we do not have to be concerned with this debate, as we will see that both tests always agree.

Whether or not an observed difference is significant does not say anything about its practical relevance: even the tiniest differences can become significant if the sample size is large enough. Therefore, for each significant difference in means μ_X and μ_Y between samples X_i and Y_i, we report the *effect size*

$$\delta = \frac{|\mu_X - \mu_Y|}{\sigma_D}, \tag{14.1}$$

where σ_D is the standard deviation of the sample difference $D_i = X_i - Y_i$. Defined in this way, the effect size δ corrects the absolute difference of the means for the variance that is present in the sample. $\delta \geqslant 0.2$ corresponds to *small*, $\delta \geqslant 0.5$ to *medium*, and $\delta \geqslant 0.8$ to *large* effects (Bortz and Döring 2006).

14.2.1 Correctness and Efficiency

To measure correctness, for each task we identified a number of program features (i. e., ecosystem parameters and recording specifications) that need to be implemented for a solution to be correct. Thus, correctness $C_{i,j}^{\kappa}$ for subject i and task j (1 = Parameters, 2 = Recording) of treatment κ (1 = GPL, 2 = DSL) is given as the quotient of the number of correctly implemented features $\eta_{i,j}^{\kappa}$ divided by the total amount of features to be implemented ν_j:

$$C_{i,j}^{\kappa} = \frac{\eta_{i,j}^{\kappa}}{\nu_j} \tag{14.2}$$

If syntax errors prohibit a solution from compiling, we set $\eta_{i,j}^{\kappa} = 0$. Overall correctness C_i^{κ} for subject i with treatment κ is defined as the arithmetic

sample is likely to be normal because, as Bortz and Döring (2006) put it, "a non-significant result says nothing."

mean of the correctness for each task:

$$C_i^\kappa = \frac{C_{i,1}^\kappa + C_{i,2}^\kappa}{2} \tag{14.3}$$

In order to assess the efficiency of the subjects, one could determine the time spent for each task. In this way, however, a participant who does not provide correct answers but does so in a very fast way would be deemed efficient. To couple task ($E_{i,j}^\kappa$) and overall efficiency (E_i^κ) to both time spent $T_{i,j}^\kappa$ and correctness $C_{i,j}^\kappa$, we define them as

$$E_{i,j}^\kappa = \frac{C_{i,j}^\kappa}{T_{i,j}^\kappa} \quad \text{and} \quad E_i^\kappa = \frac{C_i^\kappa}{T_{i,1}^\kappa + T_{i,2}^\kappa}. \tag{14.4}$$

Figures 14.3 and 14.4 give an overview of the participants' performance with regard to correctness and efficiency as defined above. For each task, more than 75 % of the subjects were able to solve it without errors using the Sprat Ecosystem DSL, which results in degenerated box plots for these cases in the correctness plot. Furthermore, no participant completely failed the parameter specification task with the DSL while there are 0 % correct solutions for both tasks with the GPL. The median efficiency for the DSL is greater than the median of the GPL among all categories. The largest difference in efficiency can be observed for the record task.

Descriptive and inferential statistics for the program comprehension tasks are summarized in Tables 14.4 and 14.5. It can be seen that the DSL solutions are on average 62 % more correct and on average 182 % more efficient than the GPL. The Shapiro-Wilk test succeeds for all correctness categories (implying that the sample is most likely not drawn from a normal distribution) and fails for all efficiency categories. As Wilcoxon's signed rank test and the t-test indicate a significant difference in means in the direction of our alternative hypotheses $\mathbf{H1}_1$ and $\mathbf{H2}_1$, we reject $\mathbf{H1}_0$ and $\mathbf{H2}_0$. The effect size δ is "large" for all categories except for the difference in correctness for the recording task ("medium" effect size).

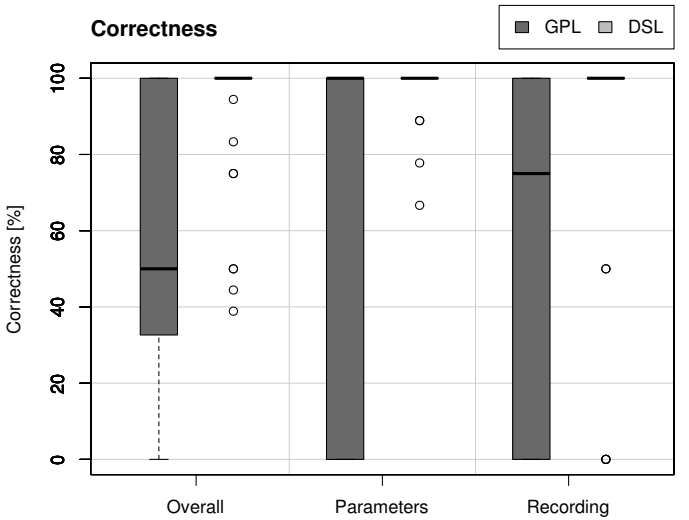


Figure 14.3. Correctness of solutions for all, the parametrization, and the recording tasks. The box plots for the DSL are degenerated because of more than 75 % fully correct solutions.

Table 14.4. Comparison of correctness for comprehension tasks. Significant p-values printed in bold.

	Correctness [%]					
	Overall		Parameters		Recording	
	GPL	DSL	GPL	DSL	GPL	DSL
Mean	56.8	92.0	60.8	97.8	52.8	86.1
Difference		+62.0 %		+60.9 %		+63.2 %
SD	39.0	17.8	47.3	6.9	45.0	33.0
Median	50.0	100.0	100.0	100.0	75.0	100.0
Shapiro-Wilk p		$\mathbf{5.1 \cdot 10^{-3}}$		$\mathbf{5.7 \cdot 10^{-7}}$		$\mathbf{6.2 \cdot 10^{-5}}$
t-Test p		$\mathbf{4.9 \cdot 10^{-6}}$		$\mathbf{1.2 \cdot 10^{-5}}$		$\mathbf{2.0 \cdot 10^{-4}}$
Wilcoxon p		$\mathbf{4.1 \cdot 10^{-5}}$		$\mathbf{1.0 \cdot 10^{-4}}$		$\mathbf{5.5 \cdot 10^{-4}}$
Effect Size δ		0.86 (large)		0.81 (large)		0.65 (medium)

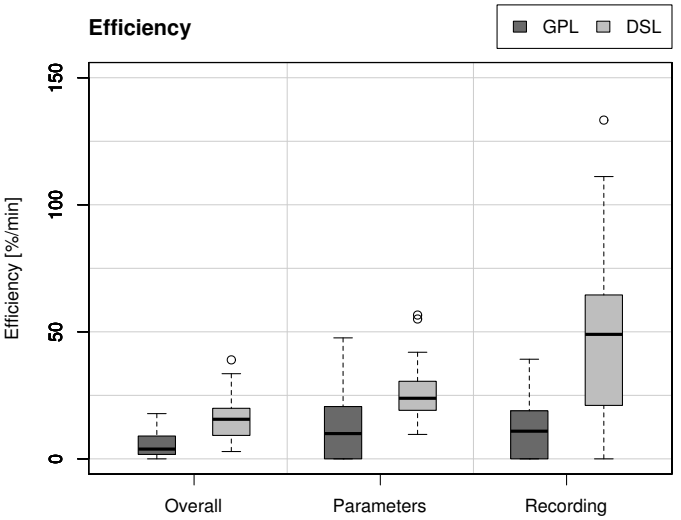
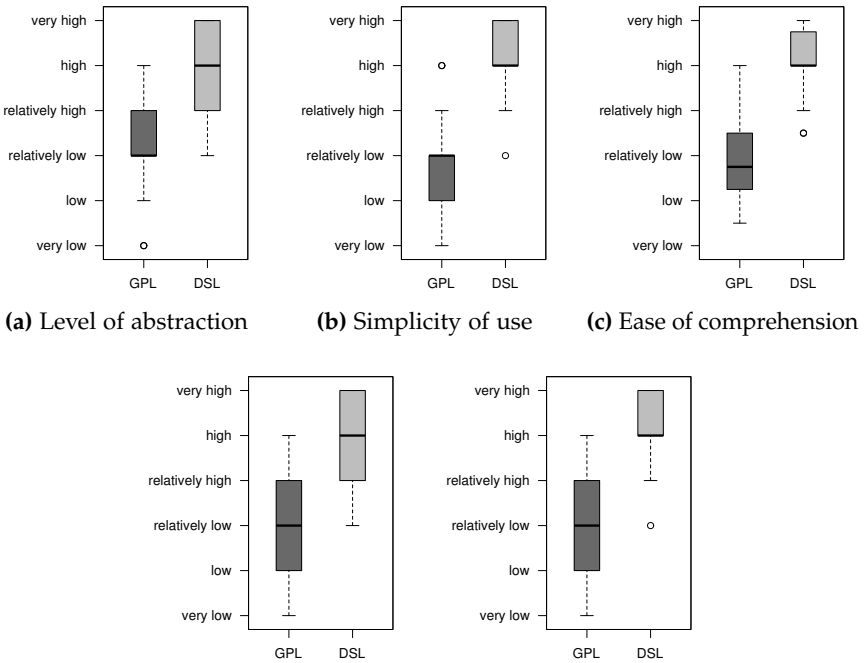


Figure 14.4. Efficiency of solving all, the parametrization, and the recording tasks.

Table 14.5. Comparison of efficiency for comprehension tasks. Significant p-values printed in bold.

	Efficiency [%/min]					
	Overall		Parameters		Recording	
	GPL	DSL	GPL	DSL	GPL	DSL
Mean	5.6	15.8	12.2	25.8	10.9	47.2
Difference		+181.9 %		+111.6 %		+302.1 %
SD	4.9	8.1	12.1	10.9	12.4	31.8
Median	3.9	15.6	9.9	23.8	10.9	49.0
Shapiro-Wilk p		$8.4 \cdot 10^{-1}$		$8.6 \cdot 10^{-2}$		$7.1 \cdot 10^{-1}$
t-Test p		$\mathbf{1.0 \cdot 10^{-10}}$		$\mathbf{1.8 \cdot 10^{-8}}$		$\mathbf{1.3 \cdot 10^{-8}}$
Wilcoxon p		$\mathbf{3.7 \cdot 10^{-9}}$		$\mathbf{1.9 \cdot 10^{-7}}$		$\mathbf{1.0 \cdot 10^{-6}}$
Effect Size δ		1.47 (large)		1.17 (large)		1.19 (large)



(a) Level of abstraction **(b)** Simplicity of use **(c)** Ease of comprehension

(d) Absence of technicalities **(e)** Maintainability of solutions

Figure 14.5. Perceived quality of the GPL and the DSL.

14.2.2 User Perceptions

The participants' perceptions of the DSL and the GPL with regard to the five quality categories explained in Section 14.1.3 are visualized in Figure 14.5. The medians of the quality ratings are all higher for the DSL relative to the GPL. Also considered independently, the absolute ratings of the Sprat Ecosystem DSL speak in its favor: the medians of the DSL are "high" for all categories and not even a single outlier is lower than "relatively low" with respect to any quality attribute.

Table 14.6. Comparison of the perceived quality of the GPL and the DSL. Significant p-values printed in bold.

	Rating [1 ("very low") to 6 ("very high")]					
	Abstraction		Simplicity		Comprehension	
	GPL	DSL	GPL	DSL	GPL	DSL
Mean	3.28	4.97	2.69	5.03	2.83	5.15
SD	1.06	0.84	1.06	0.77	0.95	0.68
Median	3.00	5.00	3.00	5.00	2.75	5.00
Shapiro-Wilk p		$\mathbf{1.9 \cdot 10^{-3}}$		$\mathbf{4.8 \cdot 10^{-2}}$		$1.3 \cdot 10^{-1}$
t-Test p		$\mathbf{1.4 \cdot 10^{-10}}$		$\mathbf{1.7 \cdot 10^{-12}}$		$\mathbf{4.9 \cdot 10^{-15}}$
Wilcoxon p		$\mathbf{2.8 \cdot 10^{-7}}$		$\mathbf{1.5 \cdot 10^{-7}}$		$\mathbf{1.2 \cdot 10^{-7}}$
Effect Size δ		1.45 (large)		1.73 (large)		2.13 (large)

	Rating [1 ("very low") to 6 ("very high")]			
	Abs. of technic.		Maintainability	
	GPL	DSL	GPL	DSL
Mean	3.06	4.97	3.17	5.06
SD	1.17	0.81	1.21	0.79
Median	3.00	5.00	3.00	5.00
Shapiro-Wilk p		$\mathbf{3.3 \cdot 10^{-3}}$		$\mathbf{3.7 \cdot 10^{-2}}$
t-Test p		$\mathbf{4.0 \cdot 10^{-10}}$		$\mathbf{3.2 \cdot 10^{-10}}$
Wilcoxon p		$\mathbf{1.1 \cdot 10^{-6}}$		$\mathbf{3.9 \cdot 10^{-7}}$
Effect Size δ		1.39 (large)		1.40 (large)

These observations are reinforced by the results of the statistical analysis given in Table 14.6. Both Wilcoxon's signed rank test and the t-test indicate a significant difference in the subjects' ratings in favor of the DSL. Therefore, we reject the null hypothesis $\mathbf{H3}_0$. For all categories we find a "large" effect size.

The answers to the open-ended questions about problems with and suggestions for the improvement of the Sprat Ecosystem DSL do not paint a clear picture because almost all suggestions made are very specific and

have been expressed only by a single subject. The only exception from this is positive feedback expressed in multiple answers along the line of: "From the limited amount I saw, [the DSL] was very good, useful, and I would use it." In the following, we list some of the given suggestions:

- One participant observed that the names of the species parameters appear redundantly in all of the species sections. Another one noted that the parameter names are long and thus time-consuming to type. Both problems should be sufficiently addressed by the autocomplete feature of the IDE for the Ecosystem DSL, which was not available to the subjects during the experiments. A different solution to this would be to use tables in a semi-graphical implementation of the Ecosystem DSL (see Chapter 8).
- A subject suggested to extend the support of the language to terrestrial simulations and IBMs. This is already supported by the Ecosystem DSL by exchanging the generator backend that generates code for a specific simulation.
- One user reported to find "the ':' sign compared to '=' in other languages rather counterintuitive" and suggested the use of "indent-sensitive code" inspired by Python. The design of the Ecosystem DSL is oriented on C-style languages, which are still dominant in the domain (see the most named programming languages by the participants). Looking at the high user ratings for the DSL with regard to ease of comprehension (Figure 14.5c) and absence of technicalities (Figure 14.5d), it can be assumed that most of the subjects did not have particular problems with the syntax of the DSL.

14.2.3 DSL Maintenance by the Users

To test the hypothesis pair $\mathbf{H4}_0/\mathbf{H4}_1$ about the maintainability of the Sprat Ecosystem DSL infrastructure by non-professionals, we focus only on participants with Java skills less than or equal to "beginner." Among our sample with $n = 36$, there are 33 subjects that match this criterion. Of these 33, only one participant was *not* able to extend the DSL correctly with an additional attribute, which corresponds to a mean success rate of $\mu = 97.0$ %. To determine whether μ is significantly greater than or equal to 90 %, we perform a one sample, one-tailed t-test. This test results in a significant p-value of 0.0141, which leads us to reject the null hypothesis $\mathbf{H4}_0$.

14.3 Discussion

In the following, we discuss the results of the statistical analysis presented above and examine possible threats to validity.

14.3.1 Correctness and Efficiency

Before considering the relevance of our findings for the evaluation of the Sprat Ecosystem DSL in particular and for the study of DSLs in general, we would like to highlight an unexpected finding from the analysis of the program comprehension tasks. If you compare the effect size for correctness between the recording task and the parametrization task, you find that it is much lower for the former than for the latter (0.65 ("medium") vs. 0.81 ("large")). This means that the DSL is less effective in reducing complexity for the programmer for the recording task than it is for the parametrization task. This is surprising because one could argue that the Sprat Ecosystem DSL features more powerful domain-specific abstractions for the recording task (recorders, recording expressions, recording intervals) than for the parametrization task (here, the GPL lacks only the concept of units). Hence, using the DSL compared to the GPL should have a larger impact on the recording than on the parametrization task. The fact that we find the exact opposite could be explained by the observation that while a DSL can simplify the description of a solution, it cannot simplify the solution itself: the task of formalizing data aggregation is complicated for domain experts irrespective of which language they use to express this formalization. Or in other words: a DSL can support domain experts in expressing solutions but it cannot create the solutions for them.

This interpretation would also explain that, while the difference in efficiency between DSL and GPL is much larger for the recording than for the parametrization task, the effect sizes are approximately equal. Those subjects who were good at solving the tasks on the domain level could express their solution much more efficiently with the Sprat Ecosystem DSL than with C++. Those subjects who already struggled with the task on the domain level were inefficient in solving it, no matter in what language they expressed their solution (resulting in a large sample variance and thus a reduced δ-value).

Our experiments provide further evidence for the hypothesis that DSLs increase program comprehension, which was first investigated empirically by Kosar et al. (2012). By using a DSL from a scientific context and participants with mostly only moderate programming experience, we extend the findings of Kosar et al. (2012) substantially. Our results show that DSLs are an effective tool to empower scientists to write well-maintainable programs (see discussion of user perceptions below) by themselves without extensive training in software engineering methods. The large effect sizes indicate that these findings are relevant for practice and they imply that the development costs of a DSL should pay off relatively quickly.

14.3.2 User Perceptions

The ratings of the Sprat Ecosystem DSL indicate that the domain-specific abstractions of the language are well-implemented (Figure 14.5a) and can be used with an accessible, concise syntax (Figure 14.5b) without introducing unwanted technicalities (Figure 14.5d). This implies that the domain meta-model of the DSL successfully captures the essential concepts of the ecosystem simulation domain and that the DSL employs a syntax that matches the jargon of the discipline.

The positive feedback of the participants with regard to the ease of program comprehension (Figure 14.5c) and maintainability of solutions (Figure 14.5e) shows that the Sprat Ecosystem DSL enables scientists to implement well-readable and well-maintainable software without a need for elaborate software engineering training. Therefore, the Sprat Ecosystem DSL contributes to the main goal of the Sprat Approach to overcome both the productivity and the credibility crisis of computational science as outlined in Section 6.3.2 of Chapter 6.

14.3.3 DSL Maintenance by the Users

DSLs make no exception to the rule that for most non-trivial software projects, maintenance accounts for a large amount of effort in the software life cycle (April and Abran 2012). As the Sprat Ecosystem DSL is supposed to enable scientists to implement software by themselves, it appears to be desirable to also enable them to carry out basic maintenance of the DSL infrastructure themselves. The results of the infrastructure maintenance test indicate that

the use of another embedded DSL for crucial parts in the implementation of the main DSL can be an effective means to achieve that.

14.3.4 Threats to Validity

This section discusses limitations of our study design that could potentially affect the validity of our findings. We distinguish between threats to internal and to external validity (Jedlitschka et al. 2008):

Threats to internal validity To what degree are the independent variables we measured or manipulated responsible for the effects observed in the dependent variables? Could instead unobserved variables be responsible for the effects?

Threats to external validity To what extent can our findings be generalized to other settings and populations (esp. to real world scenarios)?

In the following, we list only those threats to validity that are specifically relevant to our study. For threats to validity that apply to the results of experiments in general and to those in software engineering in particular, see Wohlin et al. (2012).

Internal Validity

Selection Unevenly distributed general performance levels of the subjects between the DSL and the GPL treatment group could influence the results. We circumvented this risk by choosing a within-subject experimental design. In this way, each participant acts as their own control with regard to general human performance levels.

Maturation The within-subject design, however, creates the threat of subjects becoming acquainted with the tasks and, thus, performing better with the second treatment. To mitigate this threat, we randomized the order of the DSL and the GPL treatment for each participant, which is considered to be an effective method to counterbalance the potential learning bias (Jones and Kenward 2014).

Slightly *more* than half of our sample (52.8 %) received the DSL treatment first. This does not pose a threat to our findings as it results in a bias towards the GPL treatment and our hypotheses are directed in favor of the DSL treatment.

Another threat related to the maturation of subjects is that they could become bored with the tasks, which can result in worse performance in tasks that appear later in the survey. With regard to the program comprehension tasks this threat is again mitigated by the randomization of treatment order. In addition to that, a decrease in performance could not be observed, as only one subject did not correctly complete the DSL maintenance test, which is the last test of the survey.

Mortality If the subjects who do not fully complete the survey are not representative of the total sample, the findings could be distorted. For example, if there was a significant amount of participants dropping out because they found programming with the Sprat Ecosystem DSL too hard, we would overestimate the effect of the DSL treatment. The dropout rate of 32 % we experienced is roughly equal to the average dropout rate of 30 % for online surveys (Galesic 2006). A closer examination of the dropouts reveals that most of them did not edit the task solution fields and did not spend much time on them. Additionally, the programming skill distribution of the drop-outs is very similar to the one depicted in Figure 14.2 on page 249. We therefore conclude that the drop-outs are mostly click-through participants who wanted to assess whether or not to take part in the survey and decided against it.

Instrumentation There are multiple threats related to the test instruments used in the survey. First, the type of tasks used for the program comprehension test could influence the result. We designed the tasks to be in accordance with our evaluation goal **EG** and, thus, to cover the typical activities involved in parametrizing and maintaining an ecosystem simulation from the viewpoint of ecological modelers. Additionally, the tasks cover all language constructs of the Sprat Ecosystem DSL.

Another concern is the comparability of the introductions, task descriptions, and the difficulty levels of the tasks for DSL and GPL treatment as well as the quality of the C++ API design. To mitigate this threat, we kept all descriptions given to the subjects as short and similar as possible and had them reviewed by peers/testers beforehand. For the API design, we took the code generated by the Sprat Ecosystem DSL as a starting point and derived—again with the guidance of peers—an API in accordance with good design principles for embedded DSLs (Kolovos et al. 2006). This approach of designing a C++ API that closely resembles the features

of the Ecosystem DSL allowed us to make the tasks for the DSL and the GPL treatment almost identical. Making the tasks almost identical also ensures that the complexity of the solutions is similar (although the C++ solutions have more LOC).

Another threat to internal validity is posed by the choice of C++ as the GPL. We selected C++ because it is prevalent in computational science and because we could use the code generated by the Ecosystem DSL as a starting point for the API design to keep them as comparable as possible. However, while C/C++ is one of the most named programming languages that participants had experience with (named 12 times), it would be interesting to investigate the effect of choosing a language that is even more popular among the studied population, such as R (named 25 times).

With regard to the DSL infrastructure maintenance test, the task chosen does not cover all the functionality of the embedded DSL-like API that is used for the configuration of the Sprat Ecosystem DSL. We believe, however, that the task is representative of the typical maintenance activities related to the configuration of the Ecosystem language.

A last general threat that applies to all timed experiments conducted in an online survey environment is that we cannot be sure whether the subjects solved the tasks without greater interruption or distraction. We tried to mitigate this threat at least partially by measuring only the time that the web page with the survey was in focus. This, however, allows us only to correct for interruptions that occurred on the participant's computer—telephone calls, for instance, cannot be observed in this way.

External Validity

Selection To ensure that our sample is representative of the whole studied population (scientists working with ecological models from an ecological perspective), we selectively contacted institutions (i. e., research groups) consisting of relevant practitioners. There is no indication that all institutions being located in Europe or North America has an impact on the representativeness of the sample. The use of volunteers as participants could have skewed our sample as volunteers are typically more motivated than the whole population (Wohlin et al. 2012). However, as all

participants receive both the DSL and the GPL treatment, they act as their own control also with regard to motivation. Furthermore, the relatively even distribution of programming experience among the sample, with most subjects having "regular" experience (Figure 14.2), agrees with what one would expect from the whole population according to our literature review in Chapter 6.

Setting The generalizability of our results could be impaired by simplified tasks and a relatively short usage of the programming languages. But even though the tasks are simplified (relatively few parameters to specify etc.), they are realistic for the studied domain and they cover all functionality of the Sprat Ecosystem DSL. Nonetheless, our experiments need to be complemented by further studies that focus on the use of the Ecosystem DSL in the field. This is particularly relevant to assess the impact of the associated language tools, such as the IDE, which is replaced with a web browser-enabled editor component with less functionality (esp. lacking word completion) for this online survey.

14.4 Conclusions

In this chapter, we presented the results of program comprehension and infrastructure maintenance experiments embedded into an online survey. The purpose of the survey was to evaluate the Sprat Ecosystem DSL with respect to how suitable it is for the implementation and maintenance of marine ecosystem simulations from the viewpoint of ecologists. We were able to show that users achieve significantly higher accuracy (+62 %) and efficiency (+182 %) when using the DSL instead of a comparable GPL-based solution. Over 90 % of the users with moderate Java skills were able to carry out basic maintenance tasks of the DSL infrastructure. Furthermore, the overall high user ratings of the quality of the Sprat Ecosystem DSL indicate that it successfully captures the essential domain concepts for the parametrization of marine ecosystem simulations in a concise and accessible syntax.

Our program comprehension experiments extend the empirical comparison of DSLs and GPLs initiated by Kosar et al. (2012) to scientific—i. e., non-technical—domains with users often not specifically trained in pro-

gramming and software engineering techniques. While Kosar et al. (2012) use students for their experiments, our subjects were expert practitioners from the domain. However, since our experiments—like the ones by Kosar et al. (2012)—did not utilize full DSL tool support and employed simplified tasks, our study will in the future have to be complemented by field experiments with both domain experts and full tool support using tasks based on the actual work of the domain practitioners.

All in all, the results of this survey show that the Sprat Ecosystem DSL succeeds in introducing high-level concepts for the domain of marine ecosystem simulations and, thereby, enables practitioners with on average only moderate programming experience to implement and maintain such simulations by themselves. This leads us to the conclusion that the DSL discussed here effectively contributes to the main goal of the Sprat Approach to overcome both the productivity and the credibility crisis of computational science as outlined in Section 6.3.2 of Chapter 6. The large gap between the participants' interest and their actual skill in programming as well as their preference for simpler scripting languages indicates that the development of further DSLs for the specific needs of scientific communities should be a worthwhile investment.

Chapter 15

Applying the Sprat Model to the Eastern Scotian Shelf

In this chapter, we apply the Sprat Marine Ecosystem Model to mechanistically explain the fish stock dynamics on the eastern Scotian Shelf in the time from 1970 to 2010. During this period, the Scotian Shelf ecosystem was restructured from being dominated by benthic predatory fish to being dominated by planktivorous forage fish species. The results from our modeling study suggest that environmental influences, such as changes in bottom water temperature, are likely to be a main driver of this shift in addition to fishing, which was previously identified as being the single most important driver. Furthermore, we evaluate possible management strategies that could have been employed to prevent the collapse of the predator fish stocks. Of the management strategies we assessed, a Marine Protected Area (MPA) that allows no fishing proves to be most effective but only if the protected area is sufficiently large and if it is established early enough. Our results show that the Sprat Model is able to reproduce real-world fish stock dynamics resulting from the interaction of environmental and anthropogenic influences and that the model could be developed as an exploratory tool for the evaluation of fisheries management strategies.

The chapter is structured as follows: in Section 15.1, we introduce the eastern Scotian Shelf ecosystem and the fish stock dynamics we wish to study with the Sprat Model. Our parametrization of the model is described in Section 15.2. Simulation results that aim at identifying the main drivers of the eastern Scotian Shelf fisheries dynamics and at assessing different hypothetical management strategies are presented in Section 15.3. This section also includes a sensitivity analysis of the most relevant model parameters. The presentation of results is followed by their discussion in Section 15.4 as well as concluding remarks in Section 15.5.

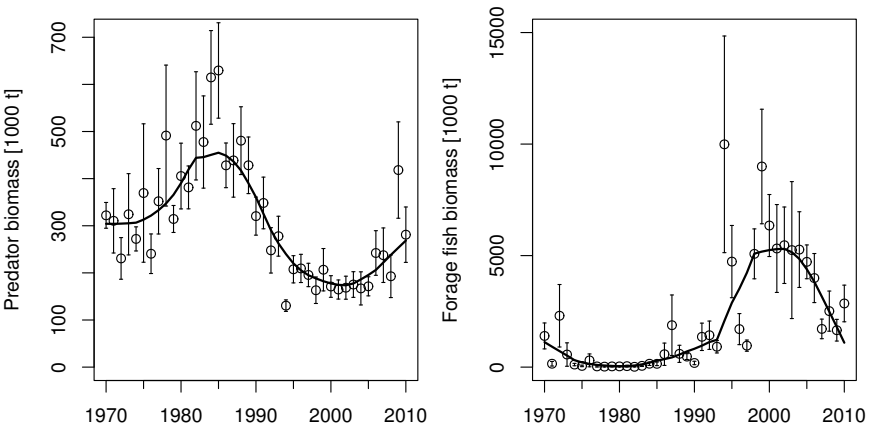


Figure 15.1. Observed biomass of demersal predator and forage fish complexes as reported by Frank et al. (2011). The bold lines result from applying a 25 % LOWESS filter.

The output of the Sprat Marine Ecosystem Model and the analysis scripts used to prepare the results we report on in this chapter can be obtained online (Johanson 2015b). Our implementation of the Spat Model itself is available online as well (Johanson 2015d).

15.1 Restructuring of the Eastern Scotian Shelf Ecosystem

In the early 1990s, the eastern Scotian Shelf ecosystem underwent a regime shift from the dominance of benthic predatory fish to the dominance of planktivorous forage fish species (Frank et al. 2011). While benthic predators (mainly cod) declined, forage fish biomass (mainly herring) increased by up to 900 % compared to levels prior to the shift (see Figure 15.1). Frank et al. (2005) found evidence that this regime shift was associated with a trophic cascade affecting not only piscivorous and planktivorous fish but also zooplankton and phytoplankton further down the food web. Due to reduced predation from the benthic predator fish complex, the forage fish complex

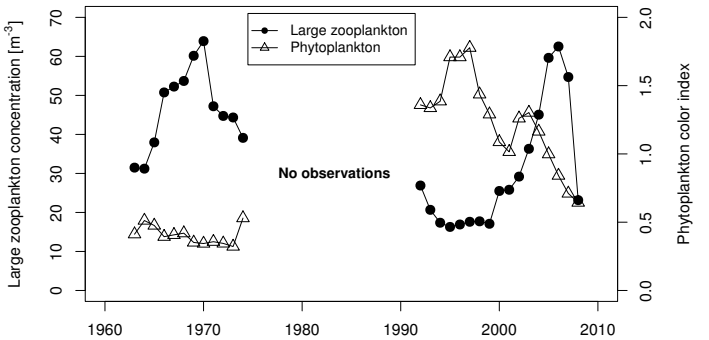


Figure 15.2. Large-bodied zooplankton abundance and phytoplankton color index as reported by Frank et al. (2011).

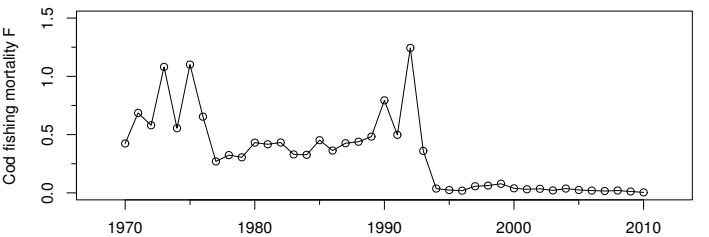


Figure 15.3. Cod fishing mortality as reported by Frank et al. (2011).

could thrive, which increased its zooplankton consumption, which in turn reduced phytoplankton mortality associated with zooplankton grazing. As can be seen from Figure 15.2, large-bodied zooplankton declined during the 1990s, while phytoplankton levels increased at the same time compared to levels before the regime shift.

Even though a fishing moratorium for cod and haddock was implemented in 1993 (see cod fishing mortality in Figure 15.3), the benthic predator stocks did not show signs of recovery for over ten years. Only from 2006 on, the regime shift seems to have started reversing slowly on all four levels of the trophic cascade.

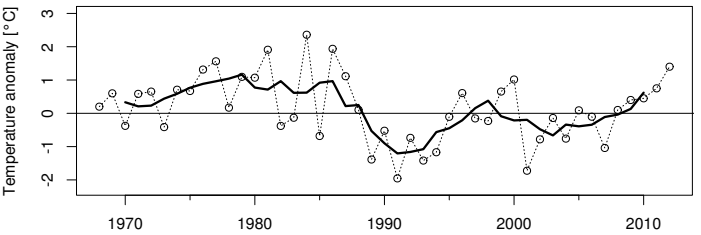


Figure 15.4. Bottom water temperature anomaly. The bold line is a five-year running mean. For details on how the data was derived, see Section 15.2.

The occurrence of the shift itself is explained by Frank et al. (2011) as being due to the intense fishing pressure on predator stocks prior to their collapse. Their findings suggest that environmental influences, such as the cooling of bottom waters (see Figure 15.4), play only a comparably minor role for inducing the shift. While they recognize that the cooling is likely to have had a pronounced negative effect on benthic predator recruitment, their analysis indicates that this effect is "dwarfed" by the impact of intense fishing.

The prolonged duration of the collapse of the predator fish complex despite the closure of fisheries is mainly attributed to predator-prey reversal (Collie et al. 2013; Minto and Worm 2012). When the predator stocks declined, the forage fish were released from predation and their biomass level increased rapidly. These large amounts of forage fish then directly competed with or preyed upon the early life stages of the benthic predator stocks, thus, severely reducing the number of new predator recruits.

An attempt at creating a model that can capture the fish stock dynamics described above was undertaken by Bundy (2005). She employed as mass-balance modeling approach using Ecopath with Ecosim (Christensen et al. 2005) to examine the trophic structure of the ecosystem prior and subsequent to the regime shift on the Scotian Shelf. Her model was able to reproduce the changes in the ecosystem structure during the regime shift but it offered little guidance for mechanistically *explaining* which are the main drivers that lead to the shift. Therefore, our focus lies on applying the Sprat Model to investigate hypothesized drivers for the collapse of the benthic predator

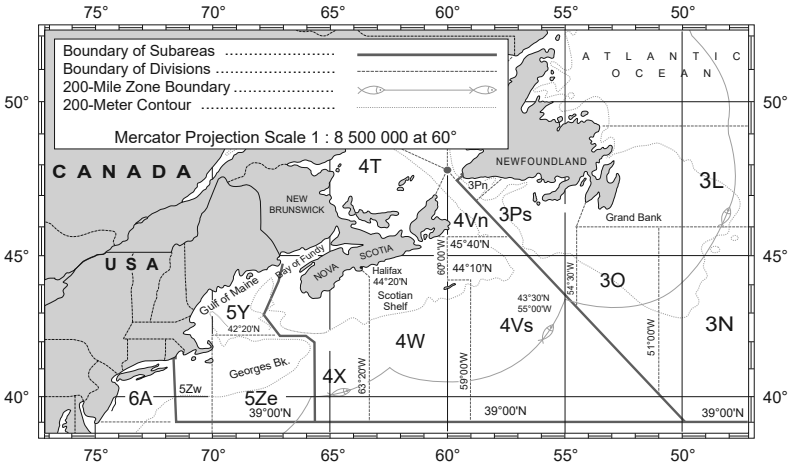


Figure 15.5. Map of the study region (NAFO divisions 4Vn,s and 4W). Figure adapted from `http://www.nafo.int/about/frames/area.html`.

stocks as well as on exploring the effectiveness of management strategies that could have been employed to prevent the collapse of the predator fish complex.

15.2 Parametrization of the Model

We use a vertically integrated version of the Sprat Marine Ecosystem Model to simulate the fish stock dynamics on the eastern Scotian Shelf from 1970 till 2010. This results in a 2D+1D version of the model as discussed in Chapter 11 with two fish species. One of these species represents the benthic predator complex and the other the forage fish complex. Initially, the fish in the model are distributed evenly in the space-size domain to achieve the biomass levels observed in 1970 (see Figure 15.1).

Space is assumed to be a homogeneous square with a size that is roughly equivalent to the continental shelf parts of the NAFO divisions 4Vn,s and 4W (see Figure 15.5). The spatial domain is equipped with periodic boundary conditions and is discretized into 48 by 48 equally sized rectangular cells.

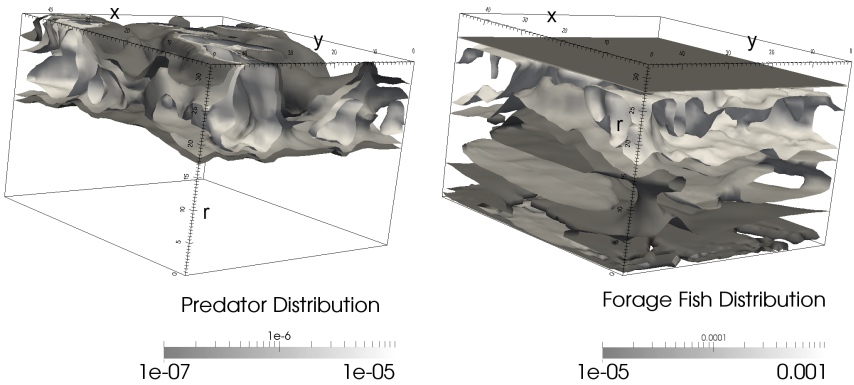


Figure 15.6. Visualization of predator and forage fish carbon biomass distributions from model output using isosurfaces. Note that for clarity all mesh cells are depicted to be of equal size in this figure.

The size dimension is divided into 32 cells using logarithmically distributed division points. On this regular mesh, we employ piecewise linear finite elements (P1 elements) to approximate the solution of the model with our FCT FEM solver from Chapter 12. We determined the resolution of the mesh to be sufficient by running a series of control simulations with increasingly finer discretizations. The refinement process was stopped once the difference in simulated aggregated stock biomasses ceased to change noticeably. For a visualization of the three-dimensional output produced by our model, see Figure 15.6.

For the simulations we report on in this chapter, the Sprat Model is *not* coupled with a fully-developed biogeochemical ocean model but instead uses the NPZ model introduced in Chapter 11 to represent the lower trophic levels of the ecosystem. Since we neglect currents in our simulations, the state variables of the NPZ model do not have to be transported in space (cf. Section 11.1.3).

The first step in the parametrization of the Sprat Marine Ecosystem Model focuses on the parameters of the NPZ model. The aim is to achieve a periodically stable evolution of the NPZ state variables while the model is *not* coupled with the fish model and the mortality term j_f (see Equation 11.9)

is included into the evolution of Z and N as in

$$\frac{dZ}{dt} = \ldots - j_f(Z) \tag{15.1}$$

$$\frac{dN}{dt} = \ldots + j_f(Z). \tag{15.2}$$

Under these circumstances, the evolution of N, P, and Z should match observations that can, for example, be obtained from the NOAA World Ocean Database (Boyer et al. 2013). All concentrations were integrated vertically, assuming the mixed ocean layer to be 40 m deep. To fit the model to these observations, we first extracted an average value for cloud coverage C_c from the NASA Earth Observations Database[1] for our model region. The other parameters of the NPZ model were initially set to values obtained from Franks and Chen (2001) and were then manually adjusted to the values given in Table 15.1 in order to match the observations from the NOAA World Ocean Database. The NPZ model proved to be most sensitive to the parameters associated with nutrient uptake (V_m, k_s) and with grazing of phytoplankton (R_m, λ_s).

In Table 15.2, we list the values of the global parameters of the fish model. The minimal predator-prey mass ratio τ was estimated from FishBase (Froese and Pauly 2015). The other two parameters, r_{view} and ζ_0, were arbitrarily chosen by us since their values did not exhibit a strong influence on modeling results in our experiments. A quite large radius of perfect information of $r_{\text{view}} = 100$ km together with $\zeta_0 = 1\ \text{kg}^{-1}$ leads to a balanced use of reactive and predictive movement strategies by the fish.

Most species parameters of the fish model are directly observable and can be obtained from individual publications or from FishBase (Froese and Pauly 2015), as can be seen in Table 15.3. When looking up parameter values, we used the species *Gadus morhua* (Atlantic cod) and *Clupea harengus* (Atlantic herring) to represent the predator and forage fish complexes, respectively. Some parameters, however, were used to manually fit the model to the observed fish biomasses described in Section 15.1. These parameters, which are mostly related to foraging, are marked with an asterisk in the "Source" column in Table 15.3.

[1] http://neo.sci.gsfc.nasa.gov/view.php?datasetId=MODAL2_M_CLD_FR

Table 15.1. Parametrization of the NPZ model for the Sprat Marine Ecosystem Model.

Symbol	Description	Value
γ	Zooplankton assimilation efficiency	0.7
ι_s	Half saturation irradiance	5 MJ m^{-2}
V_m	Uptake rate of nutrients	0.15 d^{-1}
k_s	Half saturation nutrients	30 mmol N m^{-2}
R_m	Uptake rate of phytoplankton	0.5 d^{-1}
λ_s	Ivlev constant for grazing	$1/600$ $(\text{mmol N})^{-1}$ m^2
ε_P	Mortality rate phytoplankton	$3.75 \cdot 10^{-4}$ $(\text{mmol N})^{-1}$ m^2 d^{-1}
ε_Z	Mortality rate zooplankton	$5 \cdot 10^{-4}$ $(\text{mmol N})^{-1}$ m^2 d^{-1}
ε_f	Zooplankton mortality due to grazing	0.075 d^{-1}
η_r	Remineralization rate	24 y^{-1}
N_m	Maximum nutrient concentration	170 mmol N m^{-2}
C_c	Cloud coverage	0.7

Table 15.2. Parametrization of the Sprat Marine Ecosystem Model (global parameters).

Symbol	Description	Value
τ	Minimal predator-prey mass ratio	8
r_{view}	Perfect information radius (Equation 11.51)	10^5 m
ζ_0	Scaling for feeding migration threshold (Equation 11.56)	1 kg^{-1}

Some species parameters, such as the cruise speed or the beginning and end of the mating season, can only be measured with some uncertainty. To investigate how sensitive the model is to slight variations in these parameters, we ran several test simulations in which we altered the values of each of these variables in isolation and examined the effect on model output. For cruise speed ς, slight variations (up to ± 20 %) had only neglectable influence on model results. The same is true for the times of the mating seasons: as long as the mating seasons for the two fish complexes do not overlap, their placement throughout the year does not have a large influence on model results.

Table 15.3. Parametrization of the Sprat Marine Ecosystem Model (species parameters).

Symbol	Description	Unit	Pred.	Forage	Source
$C_{\mathrm{C/dm}}$	Carbon to dry mass ratio	d.u.	0.5	0.5	Watson et al. 2014
$C_{\mathrm{d/wm}}$	Dry to wet mass ratio	d.u.	0.25	0.25	ibid.
w_{egg}	Egg dry mass	kg	$7 \cdot 10^{-8}$	$1.4 \cdot 10^{-7}$	Ouellet et al. 2001, Hempel and Blaxter 1967
w_{forage}	Min. forage dry mass	kg	$1.2 \cdot 10^{-7}$	$2.43 \cdot 10^{-7}$	ibid.
M_{plankton}	Max. Z consumption wet mass	kg	0.05	$r_{\max}$	Jobling 1981
M_{mature}	Wet mass at maturity	kg	1.2	0.173	Froese and Pauly 2015
$M_{\max}$	Maximum wet mass	kg	14.5	0.732	ibid.
a	a for L-W relationship	d.u.	$7.9 \cdot 10^{-3}$	$6.9 \cdot 10^{-3}$	ibid.
b	b for L-W relationship	d.u.	3.05	3.04	ibid.
ς	Cruise speed	BL s^{-1}	0.6	0.8	Videler and Wardle 1991
S	Mating season	n.a.	$[\frac{2}{12}, \frac{5}{12}]$	$[\frac{7}{12}, \frac{8.3}{12}]$	Froese and Pauly 2015
θ_{yolk}	Duration till w_{forage}	s	$2.85 \cdot 10^{6}$	$2.85 \cdot 10^{6}$	ibid.
E	Assimilation efficiency	d.u.	0.8	0.8	Simenstad and Cailliet 1986
ε_B	Background mortality rate	$\mathrm{m}^2\ \mathrm{s}^{-1}$	$1.47 \cdot 10^{-4}$	$7.35 \cdot 10^{-5}$	*
ζ_B	Scaling of temperature mortality	$\mathrm{s}^{-1}\ {}^{\circ}\mathrm{C}^{-1}$	5	0	*
Φ	Net wet mass fecundity	kg^{-1}	$1.3 \cdot 10^{6}$	$4 \cdot 10^{5}$	Froese and Pauly 2015
η	Predation rate	s^{-1}	10^{-5}	10^{-5}	*
F_0	Predation half sat.	$\mathrm{kg\ m}^{-2}$	0.05	0.01	*
μ	Zooplankton grazing rate	s^{-1}	$5.1 \cdot 10^{-8}$	$5.5 \cdot 10^{-8}$	*
Z_0	Zooplankton grazing half sat.	$\mathrm{kg\ m}^{-2}$	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	*

For the fishing mortality of the predator fish complex, we apply a linear interpolation of the observed fishing mortality reported by Frank et al. (2011) as shown in Figure 15.3. We assume zero fishing mortality for the forage fish complex.

We found some disagreement in the literature between different published bottom water temperature anomaly series (especially between Frank et al. (2011) and Hebert and Pettipas (2014) as well as Zwanenburg et al. (2002)). Therefore, we reanalyzed temperature data from the Canadian Department of Fisheries and Oceans Oceanographic Database[2] to obtain the most comprehensive and up-to-date data for our bottom water temperature anomaly series $\Delta T(t)$. We filtered the raw data from the Oceanographic Database to acquire only temperature measurements that have been taken in the study area on the continental shelf east of Halifax, Nova Scotia in depths between 150 m and 250 m. To obtain yearly mean temperatures for this subset of the data, we fitted an Analysis of Variance (ANOVA) model with temperature as the response variable and year and month as the predictor variables to the data (for the modeling approach, cf. Bortz and Schuster 2010; Worm and Myers 2003). The resulting yearly mean temperatures minus the overall mean temperature are plotted as open circles in Figure 15.4 on page 268. $\Delta T(t)$ itself is defined as the linear interpolation of the five-year running means of these yearly temperature anomalies (this corresponds to the bold graph in Figure 15.4). The resulting temperature anomaly is in agreement with data from individual observation stations on the eastern Scotian Shelf as reported by Hebert and Pettipas (2014) and Zwanenburg et al. (2002).

15.3 Simulation Results

We report results from simulations that investigate the main drivers of the regime shift from benthic predators to forage fish (Section 15.3.1), the sensitivity of certain parameters of the Sprat Model (Section 15.3.2), and counterfactual fisheries management strategies (Section 15.3.3).

[2] http://www.bio.gc.ca/science/data-donnees/base/index-en.php

15.3.1 Drivers of the Fish Stock Dynamics

As described in Section 15.1, three potential main drivers of the regime shift on the Scotian Shelf and its prolonged duration have been identified in the literature:

1. Intense fishing of benthic predators
2. Pronounced cooling of bottom waters
3. Predator-prey reversal

With our model, we examine the effect of each of these potential drivers by disabling the corresponding functionality in the model (e. g., by setting $\Delta T \equiv 0$) and comparing the resulting model output to a reference simulation. This reference simulation contains the full model functionality and uses the parametrization given in Section 15.2. In order to compare the effect size of the different drivers, we report L_2 norm errors relative to the predator complex biomass of the reference simulation. For the aggregated predator complex biomass of the i-th simulation

$$B_i^{[\mathrm{pred}]}(t) = \int_\Omega m^{[\mathrm{pred}]}(t,x,y,r)\, d(x,y,r) \tag{15.3}$$

and that of the reference simulation $B_r^{[\mathrm{pred}]}(t)$, the relative L_2 error is given by

$$\varepsilon_{L_2}\left(B_i^{[\mathrm{pred}]}\right) = \frac{\left\| B_i^{[\mathrm{pred}]} - B_r^{[\mathrm{pred}]} \right\|_{L_2,[0,t_{\max}]}}{\left\| B_r^{[\mathrm{pred}]} \right\|_{L_2,[0,t_{\max}]}} \tag{15.4}$$

with the L_2 norm being defined as

$$\|f\|_{L_2,[a,b]} = \sqrt{\int_a^b f(\phi)^2\, d\phi}. \tag{15.5}$$

Reference Simulation

In Figure 15.7, we show the aggregated biomasses of the predator and forage fish complexes of the reference simulation. Applying a 25 % LOWESS filter to our results removes inner-annual variations in fish biomass and allows

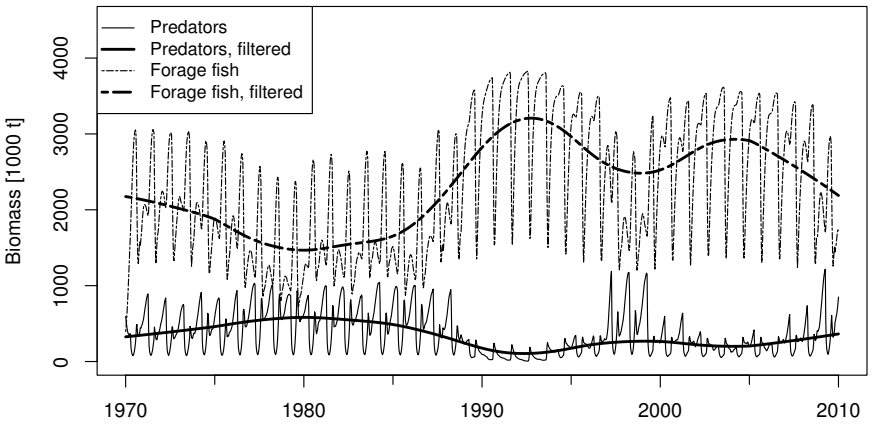


Figure 15.7. Aggregated biomasses for the predator and the forage fish complex of our reference simulation. The bold lines are the result of applying a 25 % LOWESS filter to the data.

us to compare the results from our reference simulation with observational data in Figure 15.8.

Our model is able to qualitatively reproduce the shape of the observed biomass curves. However, predator biomass is overestimated prior to its collapse (although within the margin of error of the raw data) and underestimated at the beginning of the collapse. Additionally, the benthic predators biomass exhibits a small local maximum in the late 1990s that is not present in the Scotian Shelf observations but, interestingly, has been documented for nearby benthic predator stocks (e. g., for cod in the southern Gulf of St. Lawrence as reported by Swain and Chouinard (2008)). The collapse of the benthic predators and the associated increase of the forage fish biomass occurs a few years too early in our simulation. Forage fish biomass levels are significantly overestimated prior to the regime shift and underestimated during the shift (the model can only reproduce a doubling of forage fish biomass).

The observed shift in the plankton community from large-bodied zooplankton to a more phytoplankton-dominated regime is also present in the

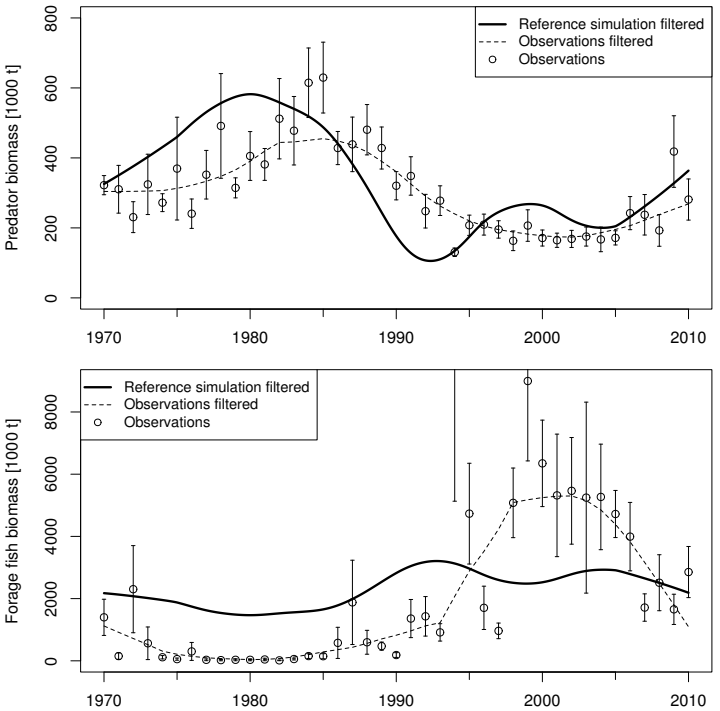
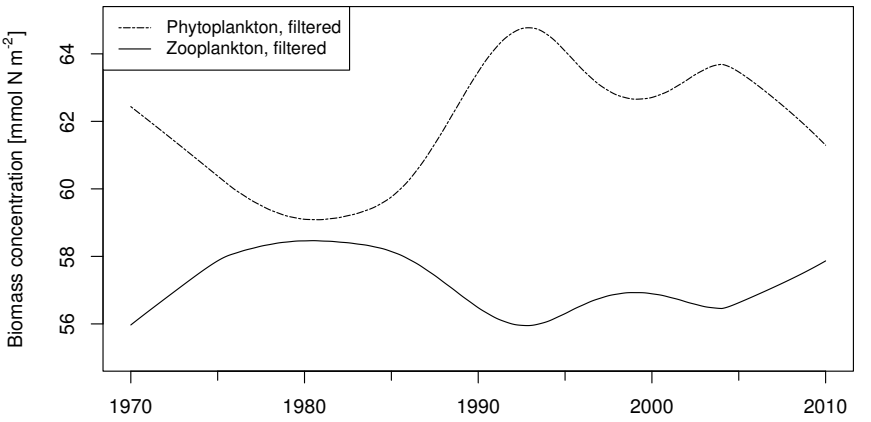


Figure 15.8. Filtered biomasses from our reference simulation in comparison with observations.

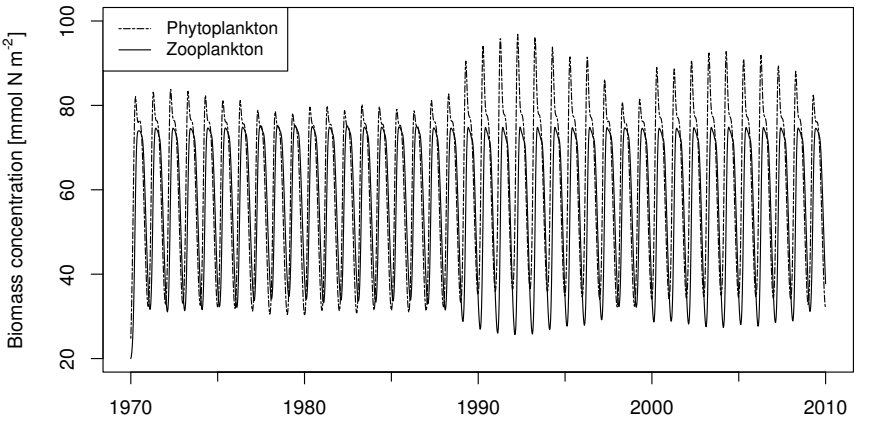
simulation data as can be seen in Figure 15.9a. A quantitative comparison of our results with the observations shown in Figure 15.3 is not possible because our NPZ model does not resolve different zooplankton size classes.

The Influence of Resolving Space

In order to test whether the explicit resolution of space in the Sprat Model is important for its predictive capabilities, we ran a simulation identical to the reference simulation but with a degenerated mesh (only two division points in each spatial dimension but still 32 in the size dimension). As can be seen



(a) 30 % LOWESS filtered



(b) Raw data

Figure 15.9. Aggregated zooplankton and phytoplankton abundance in our reference simulation.

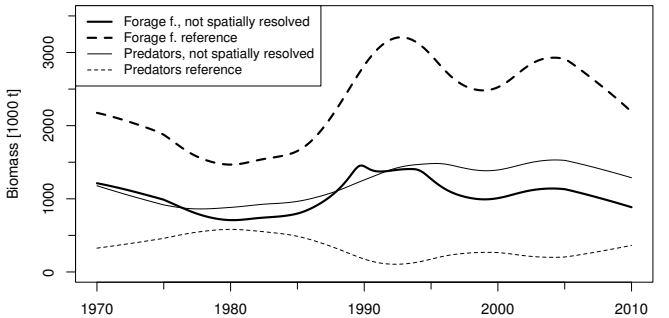


Figure 15.10. Comparison of a spatially non-resolved simulation with the reference simulation.

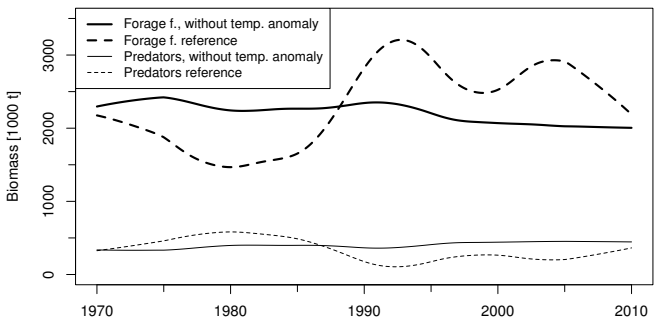


Figure 15.11. Simulation with $\Delta T \equiv 0$ in comparison with the reference simulation.

from Figure 15.10, the collapse of the predator complex is not reproduced if space is not considered. Instead, the predator stocks even dominate the forage fish stocks during the time in which the regime shift should take place.

The Influence of Temperature

To analyze the influence of bottom water temperature anomaly as a driver of the regime shift, we set $\Delta T \equiv 0$. This results in the biomass curves depicted

Table 15.4. Relative L_2 errors for the predator complex biomass and relative influence of the main drivers of the regime shift.

Scenario	ε_{L_2}	$\frac{\varepsilon_{L_2}}{\varepsilon_{L_2}(\Delta T \equiv 0)}$
$\Delta T \equiv 0$	0.478	1
$H_{\text{fishing}} \equiv 0$	0.228	0.477
No pred.-prey reversal	0.183	0.383

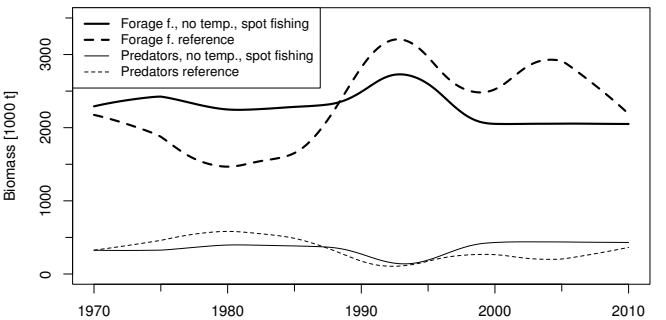


Figure 15.12. Simulation with $\Delta T \equiv 0$ and $F = 20$ in 1992 in comparison with the reference simulation.

in Figure 15.11 and a relative L_2 error of $\varepsilon_{L_2} = 0.478$ as can be seen from Table 15.4. If we neglect the influence of the bottom water temperature anomaly, the regime shift is mostly non-present.

To test, whether we can induce the regime shift by fishing alone without considering the temperature anomaly, we set the fishing intensity for benthic predators to $F = 20$ in 1992 (see Figure 15.12). Even such an unrealistically high fishing pressure cannot introduce a persisting regime shift if we do not include the effects of bottom water temperature in our model.

The Influence of Fishing

Setting $H_{\text{fishing}} \equiv 0$ in our model results in overall higher predator biomass levels and lower forage fish levels (see Figure 15.13). However, the difference

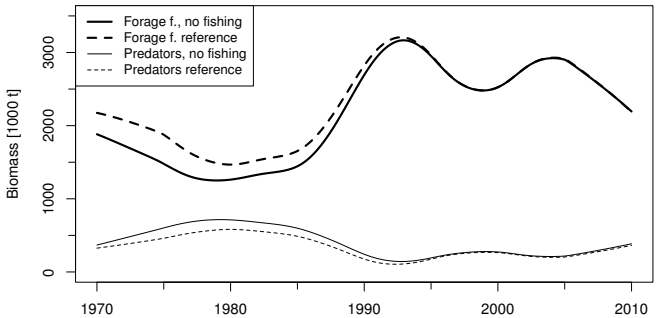


Figure 15.13. Simulation with $H_{\text{fishing}} \equiv 0$ in comparison with the reference simulation.

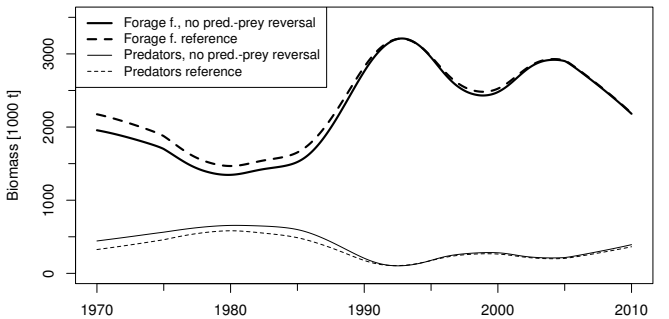


Figure 15.14. Simulation without predator-prey reversal in comparison with the reference simulation.

compared to the reference simulation becomes neglectable after about 1993 (the year of the fishing moratorium) and the regime shift is still clearly expressed. The relative L_2 error for this simulation is $\varepsilon_{L_2} = 0.228$, which is less than half of that of the scenario with $\Delta T \equiv 0$ (see Table 15.4).

The Influence of Predator-Prey Reversal

To disable the last potential driver of the regime shift (predator-prey reversal) in our model, we prevented the forage fish from preying on predator juveniles with a wet mass of less than 120 g. As is apparent from Figure 15.14, this leads to a situation similar to the no-fishing scenario discussed above. The relative L_2 error of $\varepsilon_{L_2} = 0.183$ for the exclusion of predator-prey reversal is even smaller than for the no-fishing scenario and is only about 38 % of the relative L_2 error of the $\Delta T \equiv 0$ scenario.

15.3.2 Sensitivity Analysis of the Model

This section determines which model parameters have the greatest influence on model results by conducting a coarse sweep of the parameter space. Based on the reference simulation from Section 15.3.1, we run simulations in which we halve and double the values of selected parameters (those used for fitting the model; see Table 15.3). For each of these simulations, we calculate the total misfit from fish biomass observations ε_{obs} via

$$\varepsilon_{\text{obs}} = \sqrt{\left(\varepsilon_{\text{obs}}^{[\text{pred}]}\right)^2 + \left(\varepsilon_{\text{obs}}^{[\text{forage}]}\right)^2} \qquad \text{with} \tag{15.6}$$

$$\varepsilon_{\text{obs}}^{[\text{pred}]} = \frac{\left\| B_i^{[\text{pred}]} - B_{\text{obs}}^{[\text{pred}]} \right\|_{L_2,[0,t_{\max}]}}{\left\| B_{\text{obs}}^{[\text{pred}]} \right\|_{L_2,[0,t_{\max}]}} \qquad \text{and} \tag{15.7}$$

$$\varepsilon_{\text{obs}}^{[\text{forage}]} = \frac{\left\| B_i^{[\text{forage}]} - B_{\text{obs}}^{[\text{forage}]} \right\|_{L_2,[0,t_{\max}]}}{\left\| B_{\text{obs}}^{[\text{forage}]} \right\|_{L_2,[0,t_{\max}]}}, \tag{15.8}$$

where $B_{\text{obs}}^{[\kappa]}$ are the LOWESS filtered observed fish biomasses from Figure 15.1.

In Figure 15.15, we plot ε_{obs} for the simulations grouped by model parameter. The steeper the lines connecting the misfit values, the more sensitive the model is to the respective parameter. Some model configurations exhibit a smaller error with respect to observations than our reference simulation. This poses the question of why we did not choose them as our reference instead. The answer is that, while some non-reference con-

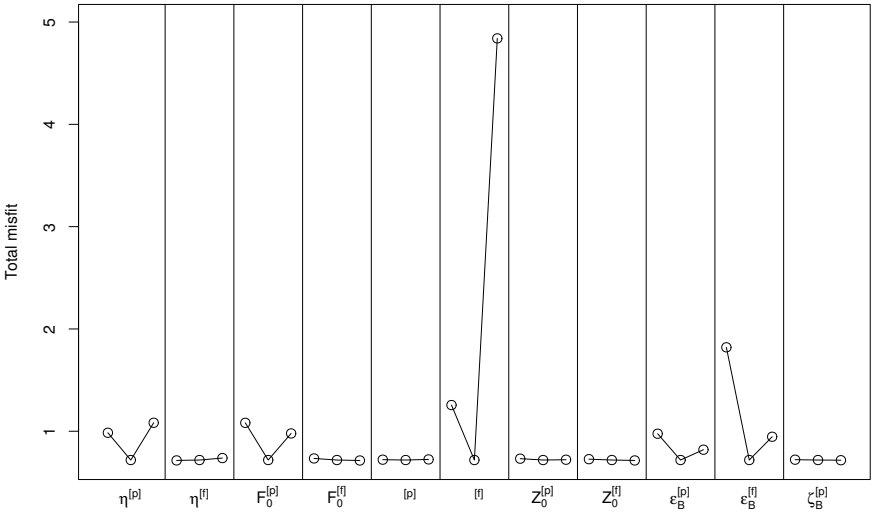


Figure 15.15. Sensitivity of the total model misfit to changes in several parameters. For each group of three values, the one in the center represents the reference simulation, the left one a simulation in which the respective parameter value is halved, and the right one a simulation in which the parameter value is doubled. The exponent [p] is for predator complex and [f] is for forage fish complex.

figurations may produce a smaller total misfit to observations, they do not capture all qualitative features of the observed biomass curves as well as our reference simulation does (cf. Section 15.3.1).

Of the eleven parameters we ran sensitivity experiments for, the Sprat Model is most sensitive to $\mu^{[\mathrm{f}]}$, which controls the zooplankton grazing rate of the forage fish complex. This is followed by the parameters for the background mortality rate $\varepsilon_B^{[\mathrm{f}]}$ and $\varepsilon_B^{[\mathrm{p}]}$, to which the model is second and fifth most sensitive, respectively. In between these two parameters, on the third and fourth place, lie $\eta^{[\mathrm{p}]}$ and $F_0^{[\mathrm{p}]}$, which govern the predation process of the predator complex. To the remaining parameters that were tested, the model is comparably insensitive.

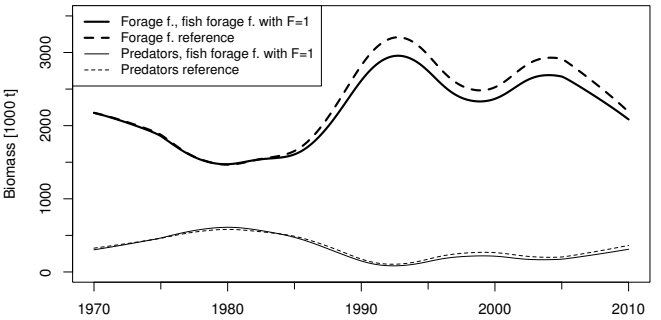


Figure 15.16. Simulation with $F = 1$ for forage fish from 1985 in comparison with the reference simulation.

15.3.3 Evaluation of Counterfactual Management Strategies

In this section, we investigate whether certain management strategies would have been effective at preventing the collapse of the benthic predator stocks. To evaluate the effectiveness of the i-th management strategy, we calculate its integrated effect on the aggregated predator stock biomass in comparison with the reference simulation from Section 15.3.1:

$$\Lambda\left(B_i^{[\text{pred}]}\right) = \frac{\int_0^{t_{\max}} B_i^{[\text{pred}]}(t) - B_r^{[\text{pred}]}(t)\, dt}{\int_0^{t_{\max}} B_r^{[\text{pred}]}(t)\, dt} \tag{15.9}$$

$$= \frac{\int_0^{t_{\max}} B_i^{[\text{pred}]}(t)\, dt}{\int_0^{t_{\max}} B_r^{[\text{pred}]}(t)\, dt} - 1 \tag{15.10}$$

Fish the Forage Fish Complex

A possible management strategy is to fish the forage fish complex in order to relieve predator fish juveniles from predation. If we do so with the relatively high instantaneous fishing mortality $F = 1$ from 1985 on, our model suggests that this would decimate the benthic predators even further ($\Lambda = -0.057$; see Figure 15.16). When increasing the fishing pressure on the

Table 15.5. Effects of different MPA designs on the aggregated predator stock biomass.

MPA coverage	MPA established in	Λ
50 %	1970	0.0328
50 %	1986	0.0212
50 %	1990	0.0271
90 %	1970	0.1926
90 %	1986	0.0398
90 %	1990	0.0403

forage fish complex, the predator stocks perform even worse ($\Lambda = -0.286$ for $F = 5$). Probably, the positive effect of lessened predation on juveniles is offset by the negative effect of less forage fish for the predators to prey on.

Establish an MPA for the Benthic Predators

Another management strategy is to establish a Marine Protected Area (MPA) in which no fishing of the predator fish complex takes place. As shown in Table 15.5, we ran simulations that included such an MPA with different temporal and spatial coverage (the breeding grounds of the benthic predators were always contained in the protected area).

The effect of all MPA configurations on the predator stocks is positive but mostly small. Establishing a no-take zone of 50 % area coverage, even as early as from 1970 on, only leads to an integrated 3.28 % increase in predator biomass. A pronounced effect of 19.26 % increase in predator biomass is only achieved with a very high area coverage of 90 % right from the beginning of simulation time.

Introduce Predator Complex Juveniles

The last management measure we look at, is to hatch juvenile fish and subsequently reintroduce them into the wild as it is, for example, practiced in the Baltic Sea (Gessner et al. 2006). As shown in Figure 15.17, this strategy can be effective at preventing the regime shift ($\Lambda = 0.252$). However, one

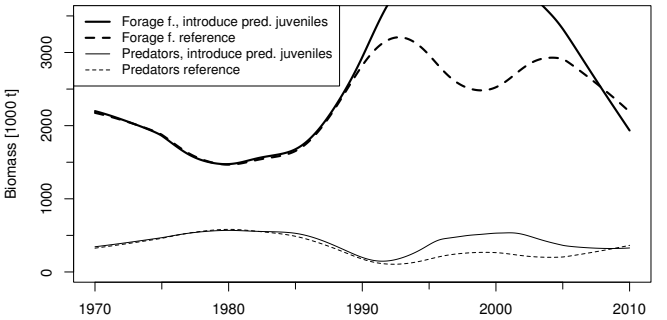


Figure 15.17. Simulation in which predator juveniles are introduced from 1993 till 2004 in comparison with the reference simulation.

would have to introduce several thousand kilo tons of juvenile fish each year to achieve this, which seems hardly feasible.

15.4 Discussion

The fish stock dynamics of the eastern Scotian Shelf ecosystem are complex, especially because of the indirect link between predator and forage fish stocks that is induced by forage fish preying on the early life stages of benthic predators (predator-prey reversal). Given this complexity, the accuracy of our reference simulation in Section 15.3.1 with respect to observed biomasses can be viewed as more than satisfactory. The Sprat Model is able to reproduce all dynamic features that are present in the observational data—especially the regime shift—and the model fits the observed predator complex biomass mostly within the margin of observational error. Since we are only interested in examining the causes of the regime shift and how it could have been prevented, a more exact match of the observed forage fish biomass curve is not relevant as long as its basic shape is reproduced (provided that the sensitivities of the modeled processes are simulated correctly).

15.4.1 Drivers of the Fish Stock Dynamics

Our modeling results indicate that it is important for an ecosystem model to explicitly consider space in order to reproduce the dynamics of the study region. Moreover, we can see that modeling environmental parameters, such as bottom water temperature anomaly, plays an important role in being able to explain the regime shift and the biomass fluctuations associated with it. In our simulations, the influence of ambient temperature on the benthic predator biomass dynamics was even a bit higher than the influence of fishing and predator-prey reversal combined, which were previously thought to be the main drivers behind these dynamics (Frank et al. 2011). Once the bottom water temperature anomaly is ignored ($\Delta T \equiv 0$), the regime shift is not reproduced in our model. However, if either fishing or predator-prey reversal is disabled, the regime shift is still clearly visible.

The comparably low impact of fishing on the biomass levels in our model is in disagreement with the observation that fish stocks generally react quite sensitively to fishing (Jennings and Kaiser 1998). We therefore have to assume that the model overestimates the influence of temperature in comparison with fishing. A possible explanation for this is that ΔT directly affects the predation (and zooplankton grazing) rates (see Equations 11.92 and 11.109), which are among the most sensitive model parameters (Section 15.3.2). The sensitivity of the model to the predation rates can, in turn, be explained by the fact that the modeled biomass levels of the fish stocks decline quite rapidly during winter (Figure 15.7). In the time following the winter, the stocks have to regenerate by taking up the right amount of biomass. If predation and grazing rates are even only marginally too low, the stocks cannot regenerate (leading to a large model misfit). If predation and grazing rates are marginally too high, stock levels overshoot (again, leading to a large model misfit). The rapid decline of modeled fish biomass in winter can be attributed to low levels of primary productivity (Figure 15.9b), which in effect deprive the fish of their food sources. In reality, stock levels would not decrease that strongly because the fish would migrate further away from the shelf region to find subsidiary food sources (Sinclair and Iles 1985). In our model, however, we (wrongly) assume a homogeneous space with no possibility of migrating out of the study area.

In conclusion, the dominant influence of temperature in comparison with fishing in our model can be traced back to a trade-off between model

complexity (ignoring migration) and accuracy. In order to correct the relative influence of temperature in the model, one would either have to parametrize migration out of and back into the study area or explicitly include areas with subsidiary food sources for the winter months into the modeled region.

Regardless of the possibly overemphasized role of ΔT, our modeling results indicate that fishing alone is probably not the sole main driver of the regime shift on the Scotian Shelf. More generally, we could say that it is unlikely that the complete restructuring of a marine ecosystem in the sense of a regime shift is induced exclusively by fishing. To cause a collapse of the predator stocks in our model, we have to assume a very high fishing mortality and even then—without the influence of temperature—the stocks recover within a few years without any signs of a lasting reorganization of the ecosystem. It appears that, while fishing may play a key role in decimating stocks, these stocks have to be weakened by other factors to cause a long-lasting collapse and a reorganization of the whole trophic structure in the ecosystem.

As discussed in Section 15.2, we recalculated the bottom water temperature anomaly ΔT from raw data because of conflicting temperature series reported in the literature. The results of our recalculation (Figure 15.4) agree with data for individual observation stations on the eastern Scotian Shelf as documented by Hebert and Pettipas (2014) and Zwanenburg et al. (2002). However, our results differ from the bottom water time series used by Frank et al. (2011) in that we observe a more pronounced and longer-lasting cooling effect in the early 1990s. If we force our model with the bottom water temperature data of Frank et al. (2011), the collapse of the predator fish complex is not reproduced in the model results. Therefore, the described differences between our temperature time series and the one of Frank et al. (2011) could explain why we find a much stronger correlation of the bottom water temperature with benthic predator biomass than they do.

15.4.2 Sensitivity Analysis of the Model

Above, we already discussed the sensitivity of the Sprat Model to predation and zooplankton grazing, specifically in the context of our analysis of the comparably large influence of ΔT. Regarding the results of the sensitivity experiments reported in Section 15.3.2 in general, we can observe that they seem to reflect the main trophic interactions one would expect to see in

the eastern Scotian Shelf ecosystem. The main control mechanisms in the ecosystem are the bottom-up forcing of the availability of zooplankton to the planktivorous forage fish and the top-down control of predatory fish preying on the forage fish complex. Therefore, it seems plausible that, as we found in Section 15.3.2, parameters associated with these processes ($\mu^{[\mathrm{f}]}$, $\eta^{[\mathrm{p}]}$, and $F_0^{[\mathrm{p}]}$) have a large effect on model results. We observe the strongest sensitivity for $\mu^{[\mathrm{f}]}$ because it is essentially the only parameter that regulates the import of zooplankton biomass into the fish model (therefore, increasing $\mu^{[\mathrm{f}]}$, dramatically increases overall fish biomass levels). The parameters $\mu^{[\mathrm{p}]}$, $Z_0^{[\mathrm{p}]}$, $\eta^{[\mathrm{f}]}$, and $F_0^{[\mathrm{f}]}$ exhibit only low sensitivity because the corresponding trophic interactions in the ecosystem are relatively weak. The only exception to this rule is $Z_0^{[\mathrm{f}]}$ (zooplankton grazing half saturation for forage fish), which belongs to a strong trophic link but to which the model is not sensitive. A possible explanation is that zooplankton levels are in saturation with respect to $Z_0^{[\mathrm{f}]}$ throughout most parts of the year (we choose $Z_0^{[\mathrm{f}]}$ relatively small) and that the same is still true for half and twice the value of the parameter.

The high sensitivity of the model with respect to $\varepsilon_B^{[\mathrm{f}]}$ and $\varepsilon_B^{[\mathrm{p}]}$ is also related to their influence on (implicit) trophic links: they parametrize predation by birds and marine mammals, which is quite strong in the study area (Frank et al. 2011).

While we saw that the bottom water temperature anomaly ΔT has a large impact on model results, the Sprat Model is insensitive to $\zeta_B^{[\mathrm{p}]}$—the scaling constant of the ΔT-dependent predator juvenile mortality. This small impact of direct mortality due to ΔT supports our argument from above that the large overall effect size of the temperature anomaly is due to its influences on foraging processes.

15.4.3 Evaluation of Counterfactual Management Strategies

Our simulation results regarding possible strategies for preventing the regime shift on the eastern Scotian Shelf show that the Sprat Model can be an effective tool for exploring such management measures. Being able to

predict the consequences of possible management strategies is important because the actual effects of management decisions may deviate from what has been intended in counterintuitive ways. For example, the model shows that fishing the forage fish complex, which could be thought to lower predation pressure on predator stocks, actually has a negative effect on predator biomass. Since such "ecological surprises" are hard to foresee and since an experimental approach to fisheries management is impractical, complex ecosystem models like Sprat that can dynamically represent the trophic links in an ecosystem are necessary in this context (cf. Yodzis 2001).

With regard to the other management strategies we explored, we find that for MPAs to be an effective management tool in our case, they have to be large (otherwise fish are still exposed to fishing due to migratory movements) and have to be established early (and not only when the stocks already show signs of collapsing). Management strategies that are based on releasing hatched juveniles into the wild prove, in our simulations, to be not feasible for marine ecosystems as large as the one in question (impracticably many juveniles would have to be released).

15.4.4 Possible Model Improvements

To achieve better prediction results in the future, the Sprat Model for the eastern Scotian Shelf could—as already discussed above—consider migration of fish to areas other than the original study region. This would allow a greater amount of forage fish biomass to survive winter and, thereby, enable the model to reproduce the increase in forage fish biomass during the regime shift much better (tremendously reducing the overall model misfit). This improvement should also decrease the overly high sensitivity of the model with respect to ΔT.

In addition to that, it would be interesting to explicitly model all the different fish species that constitute the predator and forage fish complexes to investigate the effect of the main drivers of the Scotian Shelf regime shift on the individual species.

Further suggestions for possible model improvements and future work related to the application of the Sprat Model to the Scotian Shelf ecosystem can be found in Section 19.2 of Chapter 19.

15.5 Conclusions

In this chapter, we have shown that the Sprat Model is able to reproduce the direct and indirect dynamics of the two major fish complexes on the eastern Scotian Shelf. The model has proven to be a viable tool for mechanistically explaining what drives the restructuring of a marine ecosystem and which management strategies can be effective at preventing such changes.

Furthermore, our simulation results provide new insights into the main drivers of regime shifts in marine ecosystems and what to consider in modeling them. With regard to modeling such regime shifts, our results show that it is advantageous to explicitly resolve space and to include changing environmental parameters, such as ambient temperature, into models of fish. Concerning the empirical understanding of regime shifts in marine ecosystems, our model suggests that the prolonged collapse of fish stocks is likely not to be caused by the effects of fishing alone. The stocks rather have to be already affected by other environmental pressures which make them more vulnerable to fishing. Therefore, it is important to take into consideration such environmental pressures and their predicted future change in the management of fish stocks.

Chapter 16

Coupling the Sprat Model With a Biogeochemical Ocean Model

In this chapter, we discuss how to couple the Sprat Marine Ecosystem Model with an existing biogeochemical ocean model, namely the Atlantic Canada Model (Brennan et al. 2014). The Atlantic Canada Model is a parametrization of the Regional Ocean Modeling System (ROMS) (Shchepetkin and McWilliams 2005) for the northwestern North Atlantic off the east coast of Canada. Coupling the Sprat Model with this model allows us to continue investigating the dynamics of the benthic predator and the forage fish complex in the eastern Scotian Shelf area, which we already examined in Chapter 15. Specifically, employing a fully-developed biogeochemical model to force the Sprat Model extends the end-to-end modeling capabilities of the latter and enables us to explore the spatial distribution of the fish in the model domain under more realistic environmental conditions.

In order to understand the technical challenges involved in coupling the Sprat Model with the Atlantic Canada Model, we present the fundamentals of ROMS and its horizontal and vertical discretization of the spatial domain in Section 16.1. Section 16.2 describes the Atlantic Canada Model as well as the specific data set we obtained from this model. Based on these considerations, Section 16.3 discusses how to extract data from the Atlantic Canada Model and how to map data between the grids of ROMS and the Sprat Model. In Section 16.4, we examine model results from a coupled Sprat Model setup with a focus on the spatial distribution of the fish in our model. Some concluding remarks are given in Section 16.5.

The output of the Sprat Marine Ecosystem Model and the analysis scripts used to prepare the results we report on in this chapter can be obtained

online (Johanson 2015b). Our implementation of the Spat Model itself is available online as well (Johanson 2015d).

16.1 ROMS: The Regional Ocean Modeling System

The Regional Ocean Modeling System (ROMS) is a free-surface, terrain-following, primitive equations ocean model that has been used in a wide range of scientific applications (Shchepetkin and McWilliams 2005). Being based on the primitive equations means, in particular, that the evolution of the concentration fields of the model (such as temperature, salinity, nutrients, etc.) is governed by an advection-diffusion equation. As described in Chapter 10, this type of equation is similar to the population balance equation used in the context of the Sprat Marine Ecosystem Model.

ROMS employs a curvilinear coordinate system that can represent the free surface at the ocean-atmosphere layer (i. e., time-dependent water depths, e. g., due to tidal fluctuations). Terrain-following coordinate transformations allow the model to use regular grids while still being able to accurately capture the topography of the ocean floor.

In the following, we characterize the horizontal (Section 16.1.1) and the vertical discretization (Section 16.1.2) of the three-dimensional spatial domain in ROMS. Our description is based on the information available on *WikiROMS* (ROMS/TOMS Group 2015).

16.1.1 Horizontal Discretization

In order to provide a boundary-following coordinate system (e. g., for modeling coastlines), ROMS employs an orthogonal coordinate transformation in the horizontal. Instead of using Cartesian coordinates (x, y) for curved domains, which would result in a curved grid, the model internally works with curvilinear coordinates $(\xi(x, y), \eta(x, y))$ that form a rectilinear grid (without curved coordinate lines). Differential distances $(\Delta\xi, \Delta\eta)$ in the (ξ, η)-domain are related to actual (physical) arc lengths via

$$(ds)_\xi = \left(\frac{1}{m(\xi, \eta)}\right) d\xi \quad \text{and} \tag{16.1}$$

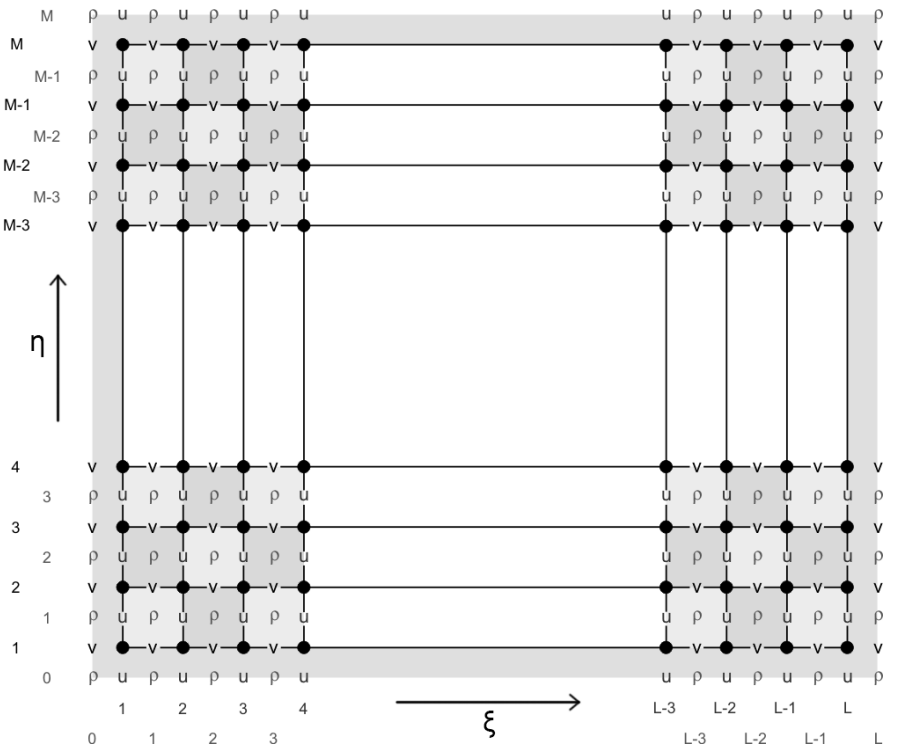


Figure 16.1. Grid points of the staggered ROMS grid in the (ξ, η)-domain. Figure adapted from *WikiROMS* (ROMS/TOMS Group 2015).

$$(ds)_\eta = \left(\frac{1}{n(\xi,\eta)}\right) d\eta \tag{16.2}$$

with the scale factors $m(\xi, \eta)$ and $n(\xi, \eta)$.

The (ξ, η)-domain is discretized using a staggered rectilinear grid as shown in Figure 16.1. The values of the concentration fields of the model (such as temperature and nutrients) are stored only for the center of each grid cell—in the so-called ρ-points. At the center of each edge of a grid cell, ROMS stores the horizontal advection velocity. Note that at the so-called

u-points (which lie on boundaries perpendicular to the ξ-direction), the model retains only the ξ-component of the advection velocity and, similarly, at v-points (which lie on boundaries perpendicular to the η-direction), only the η-component is retained.

16.1.2 Vertical Discretization

For representing space in the vertical direction, ROMS employs stretched terrain-following coordinates (Song and Haidvogel 1994). Instead of a Cartesian depth coordinate z, the model assumes a vertical coordinate $\sigma \in [-1, 0]$ that is "stretched" via a transformation to locally fit the bathymetry of the ocean floor.

The transformation from σ-coordinates to Cartesian depth coordinates is given by

$$z(t, x, y, \sigma) = S(x, y, \sigma) + \zeta(t, x, y) \left(1 + \frac{S(x, y, \sigma)}{h(x, y)}\right). \tag{16.3}$$

Here, $\zeta(t, x, y)$ is the time-varying free-surface in Cartesian coordinates and $h(x, y) > 0$ is the unperturbed water column thickness ($z = -h(x, y)$ corresponds to the bottom of the ocean). $S(x, y, \sigma)$ is the actual vertical transformation function that is given by

$$S(x, y, \sigma) = h_c\, \sigma + (h(x, y) - h_c)\, C(\sigma) \tag{16.4}$$

$$C(\sigma) = (1 - \theta_B) \frac{\sinh(\theta_S\, \sigma)}{\sinh(\theta_S)} + \theta_B \left(\frac{\tanh(\theta_S\, (\sigma + \frac{1}{2}))}{2 \tanh(\frac{1}{2}\theta_S)} - \frac{1}{2} \right), \tag{16.5}$$

where h_c, θ_S, and θ_B are control parameters to be chosen appropriately (see ROMS/TOMS Group 2015).

Figure 16.2 shows the result of the stretching transformation for a vertical transect with a constant partition of the σ-dimension $[-1, 0]$. Since the σ-direction can be discretized with the same discretization points for every location in the horizontal grid in the (ξ, η)-domain, a rectilinear grid is achieved in all three spatial dimensions, as depicted in Figure 16.3.

The three-dimensional ROMS grid consists of multiple stacked layers of the horizontal grid depicted in Figure 16.1 with the ρ-, u-, and v-points

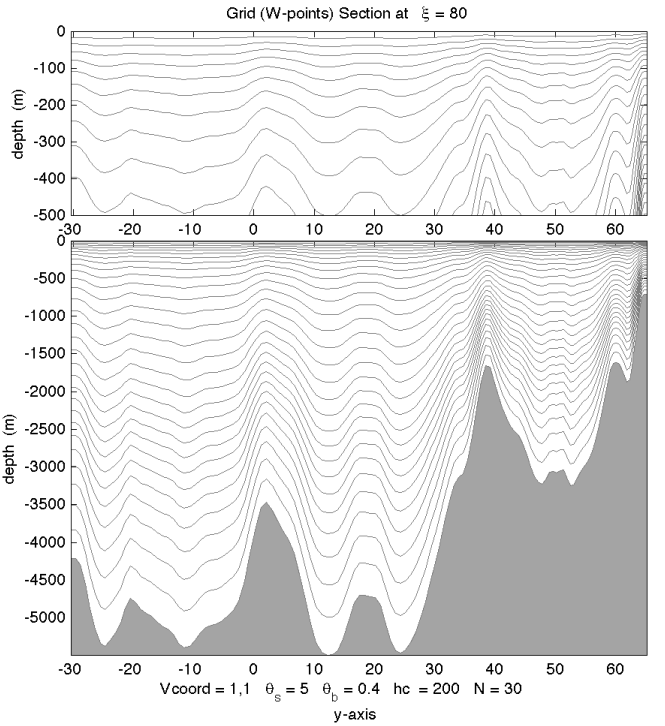


Figure 16.2. Stretched σ-layers for a vertical transect with a constant partition of the σ-dimension $[-1, 0]$. Figure from *WikiROMS* (ROMS/TOMS Group 2015).

being located vertically in the middle of each of these layers. Situated at the center of the top and bottom surfaces of each grid cell (which are orthogonal to the σ-direction) are the w-points, which correspond to the u- and v-points on the other four cell surfaces.

16.2 The Atlantic Canada Model

The Atlantic Canada Model (Brennan et al. 2014) is a parametrization of ROMS for the northwestern North Atlantic off the eastern Canadian coast. As

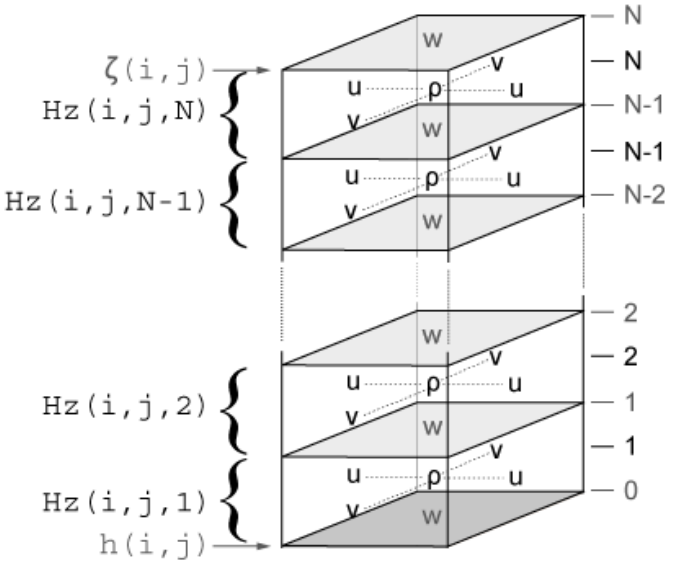


Figure 16.3. Vertical discretization in ROMS. Figure adapted from *WikiROMS* (ROMS/TOMS Group 2015).

depicted in Figure 16.4, its spatial domain extends from 36.1°N to 53.9°N latitude and 74.7°W to 45.1°W longitude and, thus, includes the entire study area of our parametrization of the Sprat Model from Chapter 15. The model grid consists of 239 times 119 horizontal cells, which corresponds to a horizontal resolution of approximately 10 km. Vertically, the σ-direction is discretized into 30 levels. The biogeochemistry of the model is governed by a module developed by Fennel et al. (2009, 2006).

Atmospheric forcing is provided to the model from an external dataset of the European Centre for Medium-Range Weather Forecasts (ECMWF) with a temporal coverage of 1979 to 2012 (Dee et al. 2011). The dataset prescribes air temperature and pressure, humidity, rain, wind stress, and net atmospheric radiation with a temporal resolution of three hours, except for the (daily) mean net surface solar radiation. With regard to spatial

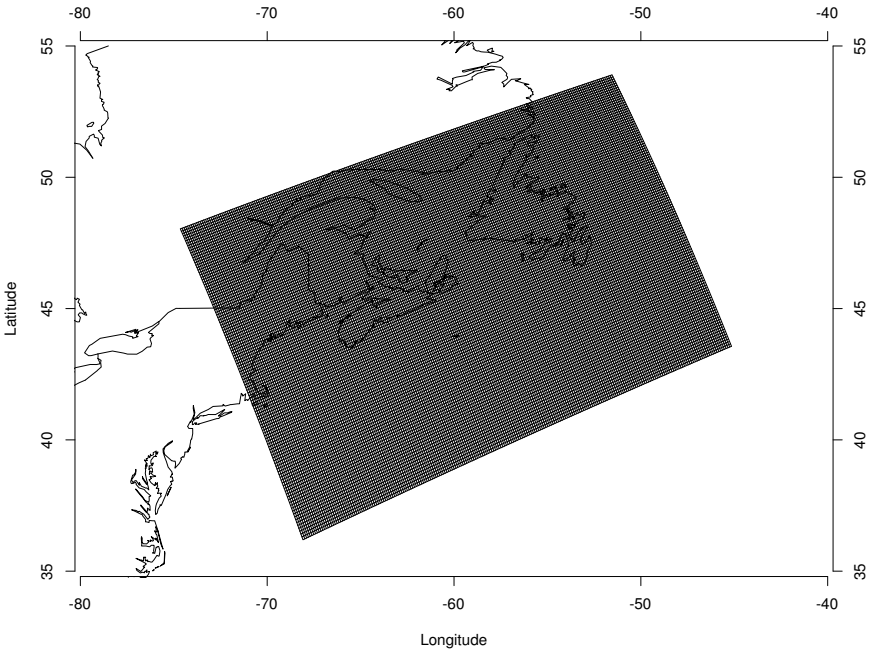


Figure 16.4. Horizontal grid of the Atlantic Canada Model.

resolution, the data is provided on a T255 spectral grid (corresponding to a resolution of about 0.7°) and was interpolated to a regular grid with a resolution of about 0.125°.

As boundary conditions for the Atlantic Canada Model, long-term monthly means (1999–2004) of output from the Urrego-Blanco and Sheng (2012) regional physical ocean model are used.

We obtained model output from the Atlantic Canada Model for four years covering the time period from the beginning of 1999 to the beginning of 2003. The time-dependent values in the dataset are averaged over periods of five days. Note that the time period covered by the dataset falls within the collapse period of the benthic predator fish stocks on the eastern Scotian Shelf, as described in Chapter 15.

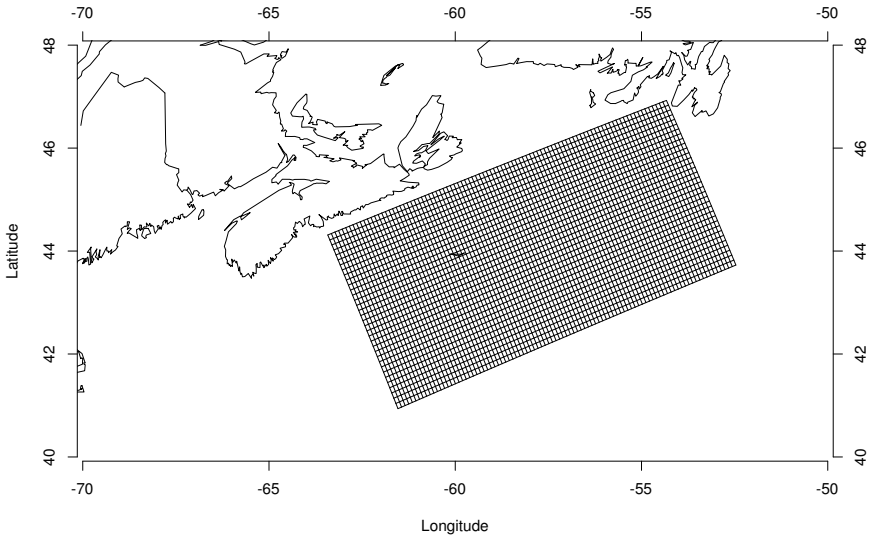


Figure 16.5. Cropped Atlantic Canada Model grid.

16.3 Coupling the Sprat Model With the Atlantic Canada Model

This section describes how we coupled our Sprat Marine Ecosystem Model presented in Chapter 11 with the Atlantic Canada Model in a one-way fashion (there is no feedback from the fish model to the biogeochemical model). The coupling process can be divided into two parts: first, we extracted and preprocessed relevant data from the model output of the Atlantic Canada Model, as presented in Section 16.3.1. This preprocessed dataset was then used to force the Sprat Model, as described in Section 16.3.2.

16.3.1 Data Extraction

We are interested in continuing to study the eastern Scotian Shelf area (as we did in Chapter 15). Therefore, we extracted model output for a subset of the spatial domain covered by the Atlantic Canada Model, namely the sub-grid

shown in Figure 16.5, which consists of 94 times 34 grid cells. ROMS—and hence also the Atlantic Canada Model—uses the Network Common Data Form (NetCDF)[1] file format for storing its simulation output. To process the output files, we employed *R 3.1.1*[2] with the *ncdf* library[3] in version 1.8.6. The extracted dataset was again stored in the NetCDF file format.

Three quantities of the model output are necessary to force the Sprat Model:

1. Vertically averaged horizontal velocities
2. Vertically integrated zooplankton concentrations
3. Bottom water temperature anomalies

The Atlantic Canada Model already provides vertically averaged horizontal velocities in its output files, which only have to be extracted for the appropriate u- and v-points. In order to vertically integrate the zooplankton concentrations, we assumed the concentrations to be constant in every cell of the grid. To obtain bottom water temperatures, we averaged the local temperature of the ten (out of 30) bottommost vertical layers for each ρ-point. In order to establish temperature *anomalies*, we calculated the mean bottom water temperature for all spatial points and all time steps and subtracted this grand mean from all the time-dependent local bottom water temperatures. Note that while in Chapter 15, we assumed the temperature anomaly to depend only on time ($\Delta T(t)$), the anomaly calculated from the Atlantic Canada Model also depends on space ($\Delta T(t, x, y)$).

Spatially averaged time series of the vertically integrated zooplankton concentrations and the bottom water temperature anomalies are depicted in Figure 16.6. The LOWESS-filtered zooplankton concentration is about five to six times lower than in our simulations from Chapter 15 (see Figure 15.9a on page 278). This is likely due to the fact that there is a high spatial variability in the distribution of zooplankton (see Figure 16.12 below) and that we sampled a region with relatively abundant zooplankton for the parametrization of the Sprat Model in Chapter 15. Additionally, there are large uncertainties associated with the measurement of zooplankton concentrations (Buitenhuis et al. 2013).

[1] http://www.unidata.ucar.edu/software/netcdf/

[2] http://www.r-project.org

[3] http://cirrus.ucsd.edu/~pierce/ncdf/

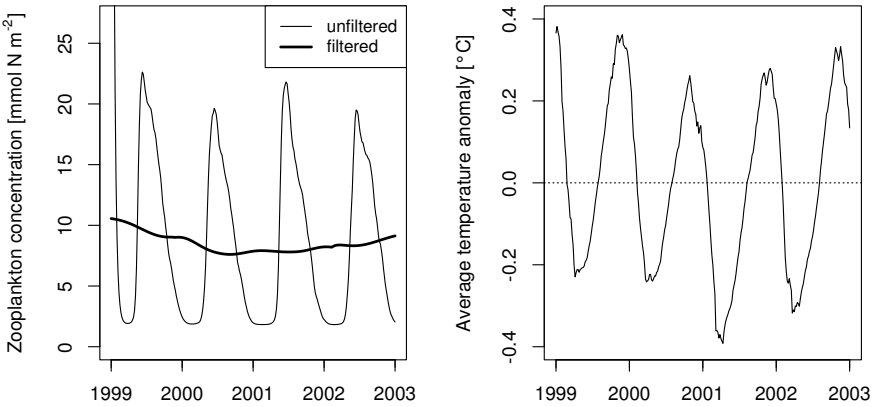


Figure 16.6. Spatially averaged zooplankton concentration and temperature anomaly. A 45 % LOWESS filter was applied to obtain the filtered zooplankton time series.

The amplitude of the inner-annual variation of the average bottom water temperature anomaly in the Atlantic Canada Model is much smaller than the inter-annual variability of the bottom water temperature on the eastern Scotian Shelf (cf. Figure 15.4 on page 268).

When extracting the water temperature data from the raw simulation output of the Atlantic Canada Model, we had to correct an apparent defect in this data: at the beginning of the year 2002, the water temperature in the model output homogeneously decreases by more than 2 °C for a single data frame (one five-day average) in the whole study area. This negative spike in temperature is reflected in a similar drop in the zooplankton concentrations at the same time. We corrected both defects by replacing the corrupted data frame of the water temperature and the zooplankton concentration field with the average of the preceding and the following data frame of the respective concentration field. Figure 16.6 already depicts the interpolated values, which have also been used for all simulation experiments we report on in Section 16.4 of this chapter.

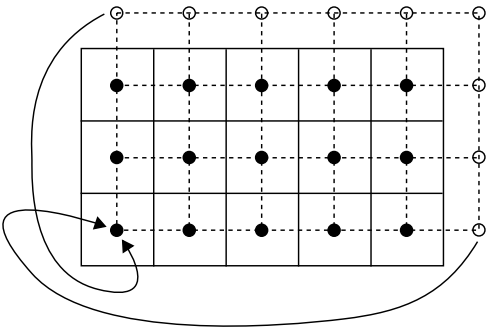


Figure 16.7. The mesh of the Sprat Model (dashed lines) is shifted relative to the Atlantic Canada Model grid (solid lines) so that its vertices match the ρ-points (black dots). Since our parametrization of the Sprat Model uses periodic boundary conditions, the vertices located on open circles are identical to those that lie on ρ-points at opposing edges.

16.3.2 Model Coupling

For coupling the Sprat Model with the Atlantic Canada Model, we employ the same parametrization of the Sprat Model as in Chapter 15. To avoid having to interpolate between the grids of the two models, we choose the mesh of the Sprat Model to match the 94 times 34 elements resolution of the cropped Atlantic Canada Model mesh (Figure 16.5). However, since the Atlantic Canada Model stores the values of concentration fields in the center of its grid cells (ρ-points) and the Sprat Model with linear finite elements stores those values at the vertices of its elements, the two grids have to be shifted relative to each other, as depicted in Figure 16.7. Since the curvature of the cropped Atlantic Canada Model grid is relatively small, we ignore it for the mesh of the Sprat Model and employ a rectilinear grid.

As discussed above, the horizontal velocities of ROMS are located on the edges of grid cells on the u- and v-points ($\xi_1, \xi_2, \eta_1, \eta_2 \in \mathbb{R}$ in Figure 16.8). To calculate a horizontal velocity V for the central ρ-point, we use the respective averages:

$$V = \left(\frac{\xi_1 + \xi_2}{2}, \frac{\eta_1 + \eta_2}{2} \right) \tag{16.6}$$

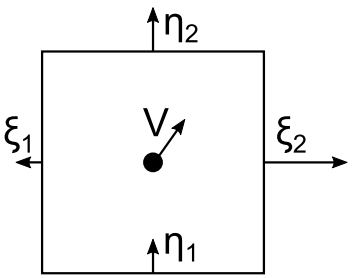


Figure 16.8. The velocity in the center of a grid cell is the average of the velocities on the boundary.

16.4 Simulation Results

In this section, we present results from conducting four simulation experiments in which we gradually use more data from the Atlantic Canada Model to force the Sprat Model:

1. The first simulation relies on the NPZ model from Chapter 11 and does not use any data from the Atlantic Canada Model.
2. The second simulation is identical with the first except for incorporating the horizontal velocities of the Atlantic Canada Model.
3. The third simulation is identical with the second except for replacing our NPZ model with the vertically integrated zooplankton concentrations from the Atlantic Canada Model.
4. The fourth simulation is identical with the third except for using the local temperature anomalies from the Atlantic Canada Model instead of the five-year running mean ΔT depicted in Figure 15.4 of Chapter 15.

16.4.1 Comparison of Aggregated Biomass Time Series

The aggregated biomass time series of the predator and forage fish complexes of the first two simulation experiments are depicted in Figure 16.9. Note that our first simulation experiment is identical with the reference simulation from Chapter 15 except for the size and shape of the spatial domain and for the time period covered by the simulation. Overall, the

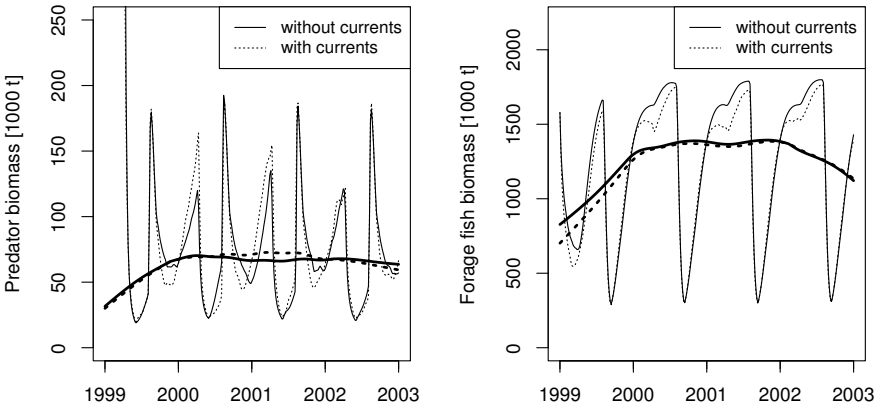


Figure 16.9. Aggregated biomass of predator and forage fish complexes for simulations with the NPZ model from Chapter 11. The bold lines result from applying a 45 % LOWESS filter to the raw data.

average biomasses in the reference simulation from Chapter 15 are about two to three times higher than for the same period of simulation time in our first simulation experiment from this chapter (cf. Figure 15.8). This difference in aggregated biomasses is likely to be explained by the fact that the spatial domain of the reference simulation in the previous chapter was assumed to be almost exactly twice as large as for our simulations in this chapter. Since with the NPZ model from Chapter 11, we assumed a homogeneous space, a difference in the size of the spatial domain directly affects the overall amount of zooplankton that is available to the fish—and, hence, the carrying capacity of the ecosystem.

Taking into account the horizontal velocities of the Atlantic Canada Model for our second simulation does not have a large effect on the average aggregated biomasses of the fish stocks. The only observable differences between the first two simulation experiments are that for the aggregated predator biomass time series, local extrema are slightly more pronounced and that for the forage fish complex, maxima are reduced a little when considering currents.

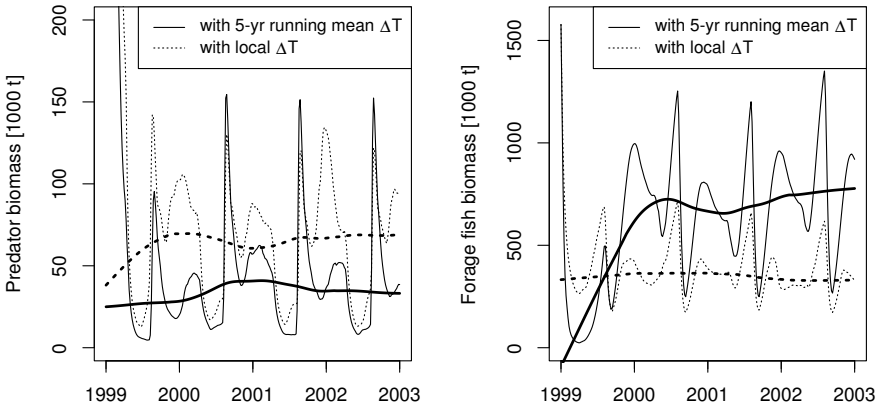


Figure 16.10. Aggregated biomass of predator and forage fish complexes for simulations with zooplankton concentrations from the Atlantic Canada Model. The bold lines result from applying a 45 % LOWESS filter to the raw data.

Aggregated biomass time series for the third and fourth simulation experiment are shown in Figure 16.10. The third simulation switches from employing the NPZ model from Chapter 11 to the use of the zooplankton concentration field from the Atlantic Canada Model. This change reduces the average biomass of both the predator and the forage fish to slightly more than half of that of the first two simulations. The reduction can easily be explained by the much lower average zooplankton concentrations in the Atlantic Canada Model compared to the parametrization of our own NPZ model, as has already been discussed in Section 16.3.1.

To obtain a fully-coupled simulation for our fourth experiment, we additionally forced the Sprat Model with the local temperature anomalies $\Delta T(t, x, y)$ that were derived from the Atlantic Canada Model. In comparison with the aggregated biomasses from the third simulation experiment, the average predator biomass is almost doubled and the average forage fish biomass is halved in the fourth experiment. The increase in predator biomass is probably due to the positive temperature anomalies in the second half of each year (see Figure 16.6). As we have seen in Chapter 15, in our model, temperature has a large effect on the predation rates of fish, to which the

Sprat Model is very sensitive. This explanation is consistent with the higher peaks in Figure 16.10 for predator biomass at the end of each year for local temperature anomalies compared to the ΔT without a yearly cycle (the five-year running mean from Chapter 15). Note that the five-year running mean temperature anomaly from the previous chapter is negative for the entire simulated time period from 1999 to 2003 (see Figure 15.4 on page 268).

The decrease of forage fish biomass from our third to the fourth experiment can be explained by the phase difference of the average zooplankton and average temperature anomaly time series in the Atlantic Canada Model (see Figure 16.6). In Chapter 15, we observed that low zooplankton intake leads to a rapid decline in forage fish biomass and that the forage rates of the forage fish complex are very sensitive to changes in the temperature anomaly. Since in each year, the peak of the average zooplankton concentration in the Atlantic Canada Model is at a time when the average local temperature anomalies are still negative, the forage fish complex cannot fully exploit the zooplankton at those times when the latter is most abundant. Hence, the carrying capacity of the ecosystem for the forage fish complex is reduced considerably.

16.4.2 Spatial Distributions

For the fourth, fully-coupled simulation experiment, we present density plots for both fish complexes and the zooplankton concentrations in the middle of certain months of the year 2002. All distributions were mapped onto the cropped grid of the Atlantic Canada Model visualized in Figure 16.5.

The distribution of the forage fish complex is depicted in Figure 16.11. A general tendency of the forage fish to avoid the shallower waters closer to the coast can be observed. In August (during their mating season), the forage fish accumulate in the eastern part of the model region, where we assumed their breeding grounds to be (not based on observations but in order to keep the model parametrization from Chapter 15, in which we arbitrarily placed the forage fish breeding grounds in the "right half" of the model domain).

By staying away from the shallower shelf waters, the forage fish follow the spatio-temporal distribution of the zooplankton shown in Figure 16.12. Throughout the entire year, the zooplankton abundance is much lower in

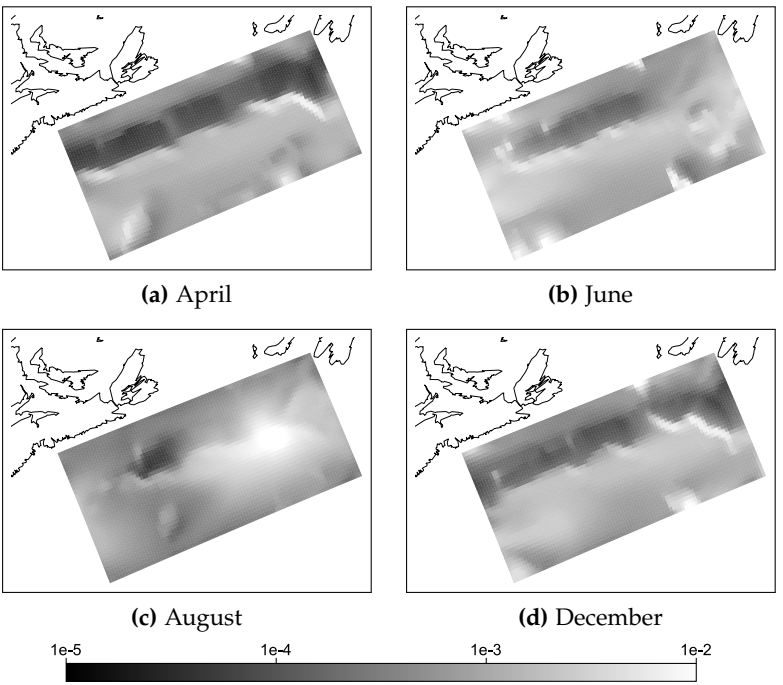


Figure 16.11. Forage fish carbon mass distributions in 2002 in kg C m^{-2}.

shallower shelf waters than in deeper waters slightly further away from the shelf.

The distribution of the predatory fish is displayed in Figure 16.13. Note that the predators are much more concentrated at certain points or fronts than the comparably more evenly distributed forage fish. In April, during the spawning season of the predator complex, the predators swim towards their breeding grounds in the west of the model region (again, the location of the breeding grounds in the model is not based on observations for the reasons stated above). Throughout the rest of the year, the predators follow local maxima in the forage fish distribution—the forage fish being their primary food source.

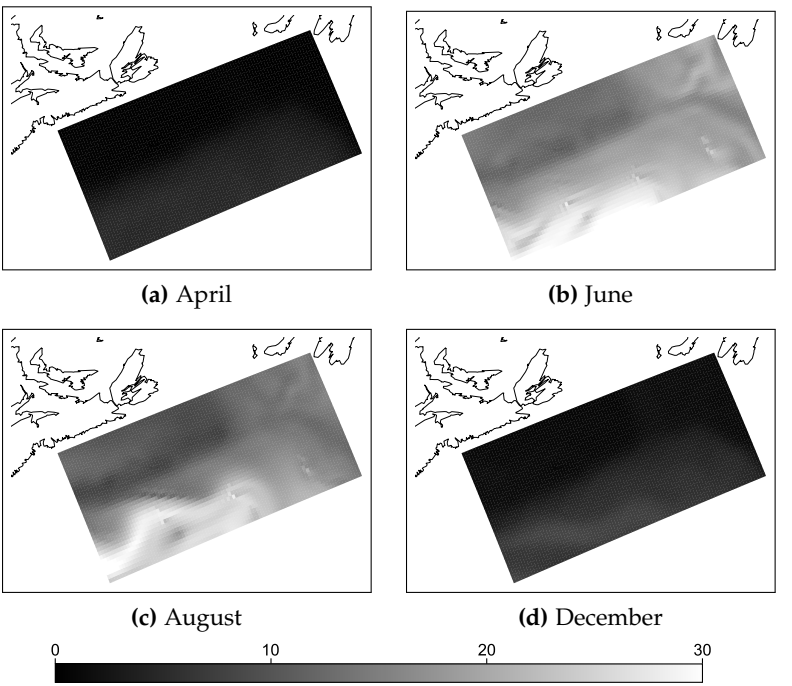


Figure 16.12. Zooplankton distributions in 2002 in mmol N m^{-2}. 1 mmol N m^{-2} corresponds to about $0.8 \cdot 10^{-4}$ kg C m^{-2}.

It is interesting to note that except in August—the mating season of the forage fish complex—, the predators are mostly absent from the *global* maximum of the forage fish distribution. This offers an explanation for the fact that we observed a large model misfit for a simulation with a degenerated spatial domain in the previous chapter in Section 15.3.1. If space is not resolved by the model, fish have no way of evading potential predators and, therefore, these predators do not carry the additional burden of locating and hunting their prey. This makes it clear why in a simulation without space, as depicted in Figure 15.10 of the previous chapter, the predator fish complex biomass is increased and that of the forage fish complex is decreased in comparison with a spatially-resolved simulation:

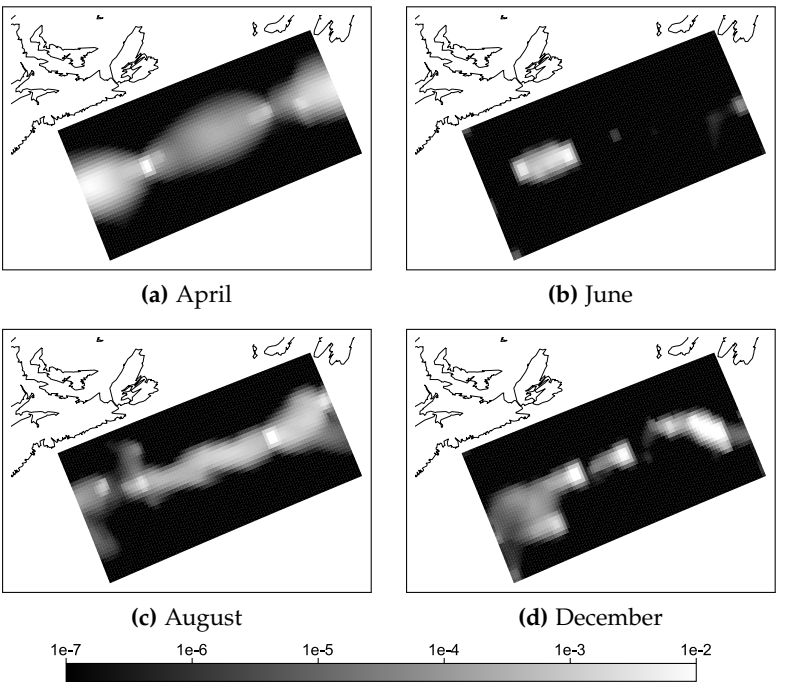


Figure 16.13. Predator carbon mass distributions in 2002 in kg C m^{-2}.

the predatory fish cannot miss those locations where their prey is most abundant.

16.5 Conclusions

In this chapter, we demonstrated that it is possible to couple the Sprat Marine Ecosystem Model with an existing biogeochemical model to obtain end-to-end modeling capabilities. This allowed us to study the spatial distribution of fish in our model parametrization for the eastern Scotian Shelf and to assess the sensitivity of the Sprat Model with respect to different environmental forcing factors. The results from our model experiments

indicate that while the Sprat Model is relatively insensitive to horizontal velocities, the spatial distribution of zooplankton and the inner-annual variability of the bottom water temperature anomaly have a large effect on model outcomes. The forage fish, which rely on zooplankton as their primary food source, follow the local maxima of the spatial zooplankton distribution and their biomass levels react sensitively to the (a-)synchronicity of zooplankton abundance and temperature anomaly time series.

Furthermore, our model results offer an explanation as to why it is important—as found in the previous chapter—to resolve space in our model in order to capture the fish stock dynamics on the eastern Scotian Shelf: in our simulations, the predator fish continuously failed to find the global maximum of forage fish to prey upon. In a model that does not consider space, the forage fish have no possibility of evading their predators in this way.

The rather low effort needed to couple the Sprat Model with the Atlantic Canada Model suggests that it would be relatively easy to implement a two-way coupling (with feedback from the Sprat Model to the biogeochemical model) between both models. A challenge that remains to be tackled for increasing the realism of the simulation is to remove the periodic boundary conditions of the Sprat Model. This would make it necessary to explicitly resolve both migration out of and back into the study area.

Chapter 17

Related Work

In this chapter, we discuss work related to the research conducted in this thesis. Section 17.1 describes related work for the Sprat Approach, and Section 17.2 examines DSLs that are similar to those we employed for the implementation of the Sprat Marine Ecosystem Model. In Section 17.3, we discuss marine end-to-end models that are comparable to our Sprat Model. Related work for our FCT FEM solver from Chapter 12 is presented in Section 17.4.

Note that, in this chapter, we do not consider work related to our contributions in the evaluation part of this thesis (Part IV) because relevant related publications are already discussed in the corresponding chapters themselves.

17.1 Software Engineering Approaches for Computational Science

This section examines existing research on software engineering approaches for computational science that is related to our Sprat Approach. In Section 17.1.1, we additionally highlight work that relates to the concept of *hierarchies of DSLs*, which is central to Sprat.

Overall, there are relatively few holistic software engineering approaches for computational science that address the whole life cycle of scientific software, as Sprat does. In contrast, most research is focused on specific aspects of this life cycle, such as testing methods for scientific software (see, for example, Kanewala and Bieman 2014). Like our approach, most other holistic engineering approaches tend to focus on MDSE and DSLs or related concepts.

The origins of employing DSLs and code generation in the development of scientific software can be traced back to a publication by Neighbors (1984), which is still highly relevant today. He presented an approach to reusable software components called Draco, which makes use of multiple DSLs to allow a system designer to transform and refine domain concepts into other domains. Although the notion of a hierarchical organization of these languages is present in Draco, the author explicitly states that the DSLs generally do not form a strict hierarchy. Apart from being strictly hierarchical, our Sprat Approach differs from Draco insofar as Sprat strongly emphasizes the separation of different roles of scientific software developers in interdisciplinary projects, which is not represented in Draco.

More recently, Palyart et al. (2012a) introduced a software engineering approach called MDE4HPC, which uses the DSL HPCML (Palyart et al. 2012b) to help scientists efficiently implement HPC applications that are independent of any specific HPC hardware architecture (see also Bruel et al. 2015). MDE4HPC, however, does not consider the aspect of collaboration in interdisciplinary teams of scientists, which is becoming increasingly important in the implementation of scientific simulation software. Furthermore, MDE4HPC focuses only on a single DSL, which, again, ignores the importance of integrating multiple scientific domains in modern HPC applications.

Schnetter et al. (2015) created a framework called Chemora for solving PDEs on modern HPC hardware. Their framework employs hierarchical code generation from DSLs to separate the concerns of the physics represented in PDEs, the numerical approximation, and the mapping onto hardware resources. While their framework draws on an idea that is similar to that underlying the Sprat Approach (separation of concerns via hierarchically organized DSLs), they do not abstract this idea from their concrete framework. Thus, they do not formulate a general software engineering approach like Sprat (with its roles and its engineering process) but only describe the implementation of their framework.

Almorsy et al. (2013) proposed to employ suites of graphical DSLs to use graphical modeling in all aspects of the scientific software development process. They provided a web-based tool which aims at enabling scientists to implement DSLs by themselves. However, it remains to be seen whether scientists are actually able and willing to design DSLs all by themselves without any support of experienced language engineers. Furthermore, Almorsy et al. do not provide an approach to how collaborating scientists

from different disciplines are supposed to organize the integration of the different graphical DSLs they are to design.

Garcia et al. (2013) introduced a component-based and aspect-oriented method for scientific software development. Their approach focuses on formal requirements engineering to enable the reuse of existing software components and their integration via aspect-oriented programming techniques. The idea is that once the requirements of a scientific application are known, it can be constructed merely by identifying suitable existing functional components and the dependency relations between them. It appears questionable, however, whether an approach that relies on sound requirements engineering by the scientists will be fruitful: as we have seen in Chapter 6, requirements are mostly unclear up front in scientific software development.

Sempolinski et al. (2015) presented a model-driven workflow approach for the collaboration of structural engineers on simulations in the design of engineering solutions. Software engineering for structural engineering exhibits many parallels to software engineering for computational science, such as the fact that structural engineers are only "accidentally" involved in software development but, nonetheless, have to tackle demanding simulation problems (cf. Chapter 6). The approach of Sempolinski et al. involves the creation of generators and workflows to integrate CAD programs and simulation packages (some of which are also used in computational science). This approach is similar to Sprat in that it uses generation to couple together high-level domain modeling tools. Nevertheless, in the end, the scope of the two approaches is different: Sprat aims at enabling scientists to implement high-quality simulation software by themselves, whereas with the approach of Sempolinski et al., the actual simulation packages are provided to the structural engineers and are not developed by them.

17.1.1 Hierarchies of Domain-Specific Languages

Work related to our concept of hierarchies of DSLs (on which the Sprat Approach is based) can be found in an overview on design patterns for DSLs by Spinellis (2001). In his article, Spinellis described the *pipeline pattern* to compose families of DSLs. In such a DSL pipeline, each stage is a DSL that handles distinct syntax elements of an input model which is passed on further down-stream through all the stages. Our concept of DSL hierarchies

extends this pattern by letting each lower stage or layer act as a domain-specific platform for the supraordinate layer. Additionally, we augment this pattern with different roles and a process model to form the Sprat Approach itself.

Preschern et al. (2012) as well as Prähofer and Hurnaus (2010) highlighted the importance of hierarchical concepts for DSLs in the context of automation systems. But instead of introducing multiple DSLs that are arranged in a hierarchical fashion, they suggested a single DSL that incorporates the concept of hierarchically nested models (Preschern et al. 2012) or hierarchical components (Prähofer and Hurnaus 2010), respectively.

An example of an approach that *horizontally* integrates DSLs for multiple domains is MENGES (Goerigk et al. 2012).

17.2 Domain-Specific Languages for Computational Science

In this section, we describe DSLs that are related to the languages we used in the engineering of the Sprat Marine Ecosystem Model, namely the Sprat PDE DSL (Section 17.2.1), the Sprat Ecosystem DSL (Section 17.2.2), and the Ansible Playbook DSL (Section 17.2.3).

17.2.1 DSLs for PDEs

Blitz++ (Veldhuizen 2000), Eigen (Guennebaud and Jacob 2010), and Armadillo (Sanderson 2010) are internal DSLs embedded into C++ that provide expression templates for matrix-vector arithmetic. They, however, are more general than the Sprat PDE DSL in that they are not specifically tailored to mesh-based PDE algorithms and they, thus, lack some important domain concepts for this purpose (especially for easily handling the geometry). Additionally, these DSLs focus more on dense and less on sparse matrices.

FEniCS (Logg et al. 2012) features the Unified Form Language (UFL), which is a DSL for specifying FEM discretizations and variational forms (Alnæs 2012). The level of abstraction of this language is higher than that of the Sprat PDE DSL and the language targets FEM practitioners rather than algorithm developers. Additionally, the FEniCS framework is limited to three-

dimensional problems (employing the Sprat Model for a three-dimensional spatial domain results in a four-dimensional problem).

There are several internal DSLs embedded into C++ that aim at the same domain and the same level of abstraction as the UFL does, namely FreeFem++ (Hecht 2012), Sundance (Long 2003), and the variational forms DSL of the Life project (Prud'homme 2006; Prud'homme 2007). These languages are not suited for implementing our FCT FEM solver for the Sprat Marine Ecosystem Model for the same reasons that the UFL is inadequate for this purpose.

Liszt (DeVito et al. 2011) is a DSL embedded in Scala that is directed at the same level of abstraction as the Sprat PDE DSL is, namely the implementation of mesh-based partial differential equation solvers. However, Liszt focuses on automatic parallelization rather than parallelization through high-level annotations, as our language does. As with the UFL, algorithms implemented with Liszt are limited to three dimensions.

Besides DSLs for FEM solvers on unstructured meshes, there are several stencil-based languages that target the implementation of finite difference methods. Examples of such DSLs include Paraiso (Muranushi 2012), which is embedded in Haskell, and the Equation Description Language of the Chemora framework (Schnetter et al. 2015). Since they are restricted to finite difference methods, these languages are not suitable for implementing our FCT FEM solver.

17.2.2 DSLs for Ecosystem Simulations

There are relatively few DSLs that address modeling ecosystem simulation experiments. A prominent example of such a language for Individual-Based Models (IBMs) is NetLogo (Wilensky 1999), which allows to express general IBMs via the concepts of agents, patches, links, and observers. As the DSL is limited to IBMs, it is not appropriate for specifying simulation experiments with the Sprat Marine Ecosystem Model.

Other examples of DSLs for simulation experiment descriptions (though not focused on ecosystem simulations) that are bound to a specific simulation package are the external description language of OMNeT++ (Varga 2001) and the ns-3 Experiment Description Language of the SAFE framework (Perrone et al. 2012).

A more general DSL for the specification of a broad range of (ecosystem) simulation experiments is SESSL by Ewald and Uhrmacher (2014), which

is embedded into Scala. The language allows to describe the parameters and data aggregation routines for series of simulation runs of supported (ecosystem) simulation systems. It can be extended via inheritance to target additional simulation packages (such as the Sprat Ecosystem Model). Nevertheless, we opted against reusing SESSL and for designing the external Sprat Ecosystem DSL because the syntax of SESSL appears to be too technical for users that have little or no programming experience (cf. the code examples given by Ewald and Uhrmacher 2014). Moreover, SESSL does not provide means for implementing error reporting on the abstraction level of the domain for invalid parametrizations of an ecosystem model.

17.2.3 Deployment DSLs

Besides Ansible, the most popular tools for automated IT administration and deployment are Puppet[1] and Chef.[2] In contrast to Ansible, they feature "heavyweight" clients on the machines to be maintained and, by default, employ a pull rather than a push scheme for configuration changes, which makes them less suitable for our purposes (cf. Section 8.1.3 of Chapter 8).

Another alternative to Ansible is CodeCloud and its XML-based Cloud Job Description Language, which can describe the deployment of compute jobs in the cloud (Caballer et al. 2014). CodeCloud, however, focuses only on the cloud and is not suitable for bare-metal compute environments (Ansible, by contrast, covers both).

17.3 End-To-End Ecosystem Modeling

This section discusses work related to the Sprat Marine Ecosystem Model, namely, prior applications of population balance equations in marine ecological modeling (Section 17.3.1) and other end-to-end models for marine ecosystems (Sections 17.3.2 to 17.3.4). The related end-to-end models are grouped according to the type of modeling framework they are based on (see Chapter 10):

[1] http://puppetlabs.com
[2] http://www.chef.io

- Section 17.3.2: models that use the approach of spatial replication of aggregated stock models
- Section 17.3.3: Individual-Based Models (IBMs)
- Section 17.3.4: Advection-Diffusion-Reaction (ADR) models

17.3.1 Population Balance Approach to Fish Stock Modeling

To the best of our knowledge, population balance equations have only been employed once before in the context of fish stock modeling, namely by Thompson and Cauley (1979). Thompson and Cauley utilized population balances to continuously capture both the age and size of fish in a model in order to provide predictions of fish size distributions for different ages. However, since they did *not* consider space and coupling with a biogeochemical model, their population balance model cannot be viewed as an end-to-end modeling approach.

17.3.2 Spatial Replication of Aggregated Stock Models for End-To-End Modeling

Most existing marine end-to-end models rely on the spatial replication of traditional aggregated stock models. Among the first end-to-end models of this type are ERSEM (Baretta et al. 1995) and ERSEM II (Baretta-Bekker et al. 1997). Both models were developed for the North Sea and divide the spatial domain into boxes rather coarsely (they use ten and 130 boxes, respectively). For each box, the dynamics of aggregated state variables that represent the lower trophic levels, fish, and even seabirds are described. The fish dynamics are modeled via a traditional stock model with a discrete age and size structure. While the two models were among the first that allowed to study the interaction of lower and higher trophic levels in marine ecosystems, the majority of their applications focused on the lower trophic levels alone (cf. Fulton 2010).

Another end-to-end model based on the spatial replication of traditional stock models is Atlantis, which has been employed in a variety of management strategy evaluations (Fulton et al. 2011, 2004a,b). Atlantis combines an NPZ model for the lower trophic levels with a weight- and age-structured

fish stock model for each spatial box. The fish model is able to represent weight-at-age dynamics depending on the actual amount of food consumed by the fish. While predation is explicitly modeled in Atlantis (including variable diets of the fish), recruitment is parametrized using the Beverton-Holt stock-recruitment relationship (Beverton and Holt 1957).

NEMURO.FISH (Megrey et al. 2007b) is a bioenergetics-based population dynamics model of Pacific herring that is coupled with the detailed NPZ model NEMURO (Kishi et al. 2007). The age-structured NEMURO.FISH model has been used to examine the bidirectional feedback mechanisms between herring and its planktonic prey. In particular, Megrey et al. (2007a) and Rose et al. (2007) studied the development of these feedback mechanisms under the influence of climate change and discussed its implications for fisheries in the face of changing climate conditions.

Similar to NEMURO with NEMURO.FISH, the fish model for anchovy population dynamics in the Black Sea by Oguz et al. (2008) is bioenergetics-based and combines an NPZ model with a model for planktivorous fish. In comparison to NEMURO.FISH, the fish model by Oguz et al. is more detailed, for example, in that recruitment is not parametrized (as it is in NEMURO.FISH) but the entire life cycle of the fish is explicitly modeled from the egg to the adult fish. The model by Oguz et al. was used to mechanistically explain certain regime shifts in the anchovy population of the Black Sea—similar to what we did with the Sprat Marine Ecosystem Model for the eastern Scotian Shelf ecosystem in Chapter 15.

Another example of the application of the spatial replication approach is the end-to-end model for the Baltic Sea by Fennel (2008), which we already introduced in Chapter 10. Besides the applications presented in the named chapter, Fennel (2009) described another interesting use of his model (and of end-to-end models in general): end-to-end models can be employed to assess the performance of "truncated" NPZ models, which do not explicitly resolve the higher trophic levels of the ecosystem. Fennel demonstrated that the effect of the higher food web can, in general, *not* be imitated reliably by a parametrization of export fluxes from a truncated NPZ model.

17.3.3 Individual-Based Models for End-To-End Modeling

The two most-prominent individual-based fish models for end-to-end ecosystem modeling are OSMOSE (Shin and Cury 2001) and InVitro (Gray

et al. 2006). While OSMOSE, which was already discussed in Chapter 10, focuses on applications that target ecological and fisheries-related questions (e. g., Shin and Cury 2004; Shin et al. 2004; Travers et al. 2007), InVitro aims at multiple-use management questions. For this purpose, InVitro includes a socio-economic model that can represent a broad range of human activities which influence decision making, such as recreational fishing, tourism, shipping, mining, etc. (see Fulton 2010; Little et al. 2006; McDonald et al. 2008).

Besides these two models, which are applied in a wide range of scenarios, there are many coupled fish IBMs that are designed for special purposes. One example of such an IBM for end-to-end modeling was developed by Utne and Huse (2012) to study the spatio-temporal distribution of pelagic fish in the Norwegian Sea. The model was employed in combination with a biogeochemical ocean model for examining the role of planktivorous fish for the abundance and spatial distribution of zooplankton in the named area (Utne et al. 2012).

17.3.4 Advection-Diffusion-Reaction End-To-End Models

ADR equations are not frequently employed in fish stock modeling and if they are used, then it is primarily for models that focus on marine top predators, like tuna. An example of such an application of ADR equations is the model by Sibert et al. (1999), who utilized the ADR framework to estimate movement and mortality parameters from tagging data for tuna in the western Pacific Ocean.

A fish model for end-to-end modeling that is based on ADR equations is SEAPODYM (Bertignac et al. 1998; Lehodey et al. 2008), which also focuses on tuna and was already introduced in Chapter 10. SEAPODYM was used in several end-to-end modeling applications, such as the prediction of the long-term development of tuna spawning conditions in the Pacific basin under the consideration of climate change (Lehodey et al. 2013, 2010).

17.4 Flux-Corrected Transport FEM With Explicit Multi-Step Methods

In this section, we review work related to the Flux-Corrected Transport (FCT) FEM solver introduced in Chapter 12. What distinguishes our solver for advection-dominant PDE systems from previous work in this domain, is that it applies the concept of FCT in combination with explicit multi-step methods and the FEM.

The fundamental idea of FCT—i. e., to locally reconstruct a high-order from a ripple-free low-order solution via anti-diffusive fluxes without creating ripple—was introduced by Boris and Book (1973, 1976) for finite difference methods. Zalesak (1979) generalized the method to multiple dimensions with his famous flux limiter, which we also used for our algorithm in Chapter 12.

The works of Löhner et al. (1987, 1988) enabled the application of FCT to the FEM. The FCT FEM was subsequently further developed by Kuzmin et al. (2012, 2003), Kuzmin and Turek (2002), and Möller et al. (2005), who focused mainly on combining FCT FEM discretizations with implicit time stepping methods. Explicit high-order (i. e., multi-step) methods have been introduced in combination with FCT by Lee et al. (2010) but only for finite volume methods and not for the FEM.

A comparative study of the approximation quality of the FCT FEM and other numerical methods for advection-dominant problems was conducted by John and Schmeyer (2008). They concluded that in their tests, FCT FEM schemes "were clearly the best" with regard to minimizing spurious oscillations while achieving high-order accuracy.

Part V

Conclusions and Future Work

Chapter 18

Conclusions and Lessons Learned

In this chapter, we draw conclusions from the research conducted as part of this thesis. The chapter is divided into one section covering our software engineering research (Section 18.1) and one regarding our ecological modeling research (Section 18.2). Both of these sections themselves are structured according to the work packages presented in the corresponding research design descriptions in Chapters 5 and 9. With respect to our software engineering research, we also discuss the role of software engineers in computational science (Section 18.1.5) and lessons learned for designing DSLs for scientists (Section 18.1.6).

18.1 Software Engineering Research

In this thesis, we introduced the model-driven software engineering approach Sprat, which aims to facilitate the collaboration of scientists from different disciplines in the development of well-engineered simulation software without the need for extensive software engineering training.

18.1.1 Software Engineering in Computational Science

To understand the requirements for an engineering approach for computational science, in Chapter 6, we identified the most pressing challenges of computational science today (its productivity and credibility crisis) and the specific characteristics of software engineering in this discipline that presently prevent overcoming these challenges. Examining the characteristics of software engineering in computational science enabled us to show that current attempts at solving the dual crisis of computational science (peer-review of source code, having software engineers implement software

for the scientists, and workshop-based software engineering training) do not suffice alone and that software engineering techniques have to be *adapted* in order to be adopted by this community. Furthermore, we demonstrated that MDSE methods and specifically Domain-Specific Languages (DSLs) are good starting points for such adaptations.

The results of our analysis of the characteristics of scientific software engineering answer our research question **SERQ1** from Chapter 5 (*What is specific about software engineering in computational science?*) with its sub-question **SERQ1.1** (*Which software engineering methods are well-suited for being adapted for computational science?*).

18.1.2 The Sprat Approach

Our Sprat Approach, which we introduced in Chapter 7, hierarchically integrates multiple DSLs to enable scientists from different disciplines to develop well-engineered simulation software without the need for elaborate software engineering training. The DSLs are integrated hierarchically in order to mirror the hierarchically layered architecture of typical scientific simulation software. Sprat takes into account the specific characteristics of computational science and the requirements for a software engineering approach for this discipline that result from these characteristics. Specifically, Sprat is designed to

- reconcile the conflicting quality requirements of performance, portability, and maintainability,
- increase the productivity of interdisciplinary teams of scientists developing software, and
- enhance the reliability of scientific results from *in silico* experiments
- without introducing accidental complexities.

Our discussion of the Sprat Approach answers the research question **SERQ2** (*How can multiple DSLs be integrated for scientific software development and how can they interact with each other?*).

18.1.3 DSLs for a Marine Ecosystem Model

To evaluate the Sprat Approach, we applied it to the engineering of the Sprat Marine Ecosystem Model in an exploratory case study. For this purpose, we designed the Sprat PDE DSL and the Sprat Ecosystem DSL and reused the Ansible Playbook DSL. The integration of these DSLs into a suitable DSL hierarchy, which is described in Chapter 8, answers the research question **SERQ3** (*Which DSLs are suitable for the implementation of the Sprat Marine Ecosystem Model with the Sprat Approach and how can they be integrated into a DSL hierarchy?*).

18.1.4 Evaluating the Sprat Approach

Our evaluation of Sprat consisted mainly in assessing the individual DSLs we designed for the Sprat Model, as detailed in the discussion of our research design in Chapter 5. To evaluate the Sprat PDE DSL, we conducted several micro- and macro-benchmark experiments to demonstrate that the runtime performance of solutions implemented with the DSL is not inferior and in some cases even superior to comparable GPL solutions. This answers our research question **SERQ4.1** (*How does the runtime performance of solutions implemented with the Sprat PDE DSL compare to equivalent GPL solutions?*). Furthermore, expert interviews with both professional DSL developers and domain experts working with PDE solvers showed that the Sprat PDE DSL can increase the comprehensibility, maintainability, and testability of PDE solver implementations. The fact that most of the interviewed domain experts can envisage using our DSL as part of their own work shows that we succeeded in designing a DSL which is actually accepted by a community often critical of new technologies. With the results from our expert interviews, we were able to answer the research question **SERQ4.2** (*How do experts rate the functional and technical quality of the Sprat PDE DSL?*).

For evaluating the Sprat Ecosystem DSL, we conducted an online survey with embedded experiments among academics working in ecology. The results from the controlled experiments show that, compared to a GPL-based solution, the Sprat Ecosystem DSL achieves much higher accuracy and efficiency (62 % and 182 % higher, respectively) for typical tasks that require program comprehension in the implementation and maintenance of ecosystem simulations. This answers our research question **SERQ4.3**

(*How much can program comprehension and efficiency for the implementation and maintenance of marine ecosystem simulations be increased by the Sprat Ecosystem DSL in comparison to a GPL-based solution?*). High user ratings of the quality of the Sprat Ecosystem DSL indicate that it successfully captures the essential domain concepts for the parametrization of marine ecosystem simulations in a concise and accessible syntax. Moreover, individual user feedback showed that the Sprat Ecosystem DSL succeeds in making scientific software development accessible to domain experts with little programming knowledge:

> "[The Sprat Ecosystem DSL] seems quite interesting and useful. I have quite little programming experience and do not use programming languages much (just a little Matlab and recently R), but I might use more programming in the future.
>
> When it got to the exercises in the survey, I feared they may be too hard for me, but I could solve them all and now I am really happy about that."
> (Feedback from a participant of the online survey)

These considerations answer the research question **SERQ4.4** (*How do experts rate the functional and technical quality of the Sprat Ecosystem DSL?*).

Together, our results for the research questions **SERQ4.1** to **4.4**, which we presented in Chapters 13 and 14, answer the research question **SERQ4** (*Does the Sprat Approach improve the productivity of scientists in developing software and does it raise the quality of solutions implemented by them?*—with *a solution of high quality* being understood as one that reconciles the conflicting quality requirements of performance, portability, and maintainability and that produces scientifically reliable results; cf. Section 18.1.2). The positive evaluation of the individual DSLs and the fact that we were able to implement a simulation as complex as the Sprat Marine Ecosystem Model as part of this thesis demonstrate that the Sprat Approach indeed facilitates the efficient implementation of well-engineered scientific software. Note that our exploratory case study for evaluating the Sprat Approach involved only domain experts—i. e., only professionals.

18.1.5 Software Engineers in Computational Science

As with any other approach for bridging the gap between software engineering and computational science, Sprat requires software engineers to assist scientists in the development of scientific software (in our case, language engineers have to design and implement DSLs). This can be problematic because positions for software engineers to provide development support in scientific research institutions have typically not been supported by funding agencies in the past (Carver et al. 2007). The neglect of such positions by most funding bodies should be reconsidered, as it has been shown that investing in such positions can have a markedly positive impact (Killcoyne and Boyle 2009). In the meantime, Sprat minimizes the input that is needed from trained software engineers by letting the scientists stay in full control of all development activities of the scientific software itself (using the DSLs designed by professional language engineers).

18.1.6 Lessons Learned From Applying Sprat

As a final conclusion from applying the Sprat Approach, we present several lessons learned during our exploratory case study that are of general interest when developing DSLs for computational science. These lessons learned concern four areas of DSL design:

1. *Abstraction level of the meta-model*: the more related the application domain of a DSL is to computation, the more it needs to be possible for the users to influence how computations are executed (i. e., the closer to the underlying hardware platform the language must be). This especially allows full control over the runtime performance of solutions. In our experience, such languages are best implemented as internal DSLs embedded in relatively low-level programming languages, such as C++, which enables to reuse much of the existing language facilities of the host language.

2. *Concrete syntax*: the scientists participating in our study favor DSLs that do not prescribe too much structure of models because they want to have full control over their code and want to be able to experiment with it quickly and freely.

3. *DSL tooling*: in application domains not related to computing (such as biology) external DSLs with their own tooling are more likely to be accepted. The tooling should make sure that only scientifically reasonable DSL models are permitted and, otherwise, confront the user with error messages on the abstraction level of the domain (which would not be possible after generating code into programming languages with a lower abstraction level).

4. *Documentation*: professional DSL designers often focus on language references based on a formal meta-model or on the abstract syntax while scientists favor code examples to get a condensed overview on the capabilities of a DSL and to become productive quickly.

18.2 Ecological Modeling Research

In this thesis, we introduced a novel modeling approach for the fish component of marine end-to-end models that is based on population balance equations.

18.2.1 Population Balances for Fish Stock Modeling

By comparing the theoretical foundations of our population balance approach with those of other modeling frameworks for fish in marine end-to-end models in Chapter 10, we demonstrated that our approach combines the advantages of the other frameworks while avoiding their main drawbacks. Specifically, most parameters of a population balance model are observable in individual fish, the food web structure is an emergent property of the model, and it is based on the well-developed mathematical theory of Partial Differential Equations (PDEs), which also makes its integration with existing biogeochemical ocean models relatively easy. The examination and comparison of the different modeling approaches answers our research question **EMRQ1** from Chapter 9 (*How can fish be integrated with (existing) biogeochemical ocean models from a conceptional and technical perspective?*).

18.2.2 The Sprat Marine Ecosystem Model

The Sprat Marine Ecosystem Model, which we presented in Chapter 11, is a fish stock model based on our population balance approach that is coupled with existing biogeochemical ocean models. A key contribution in developing the Sprat Model is the selection of relevant biological processes to be represented in a spatially-explicit, PDE-based fish model with a focus on the individual fish and the description of how these processes are to be modeled. This contribution answers our research question **EMRQ2** (*Which interactions of fish with each other, with other components of the ecosystem, and with the environment have to be modeled and how can these interactions be represented in the context of population balance equations?*).

18.2.3 A State-of-the-Art PDE Solver

For approximating the solution of the Sprat Model with a state-of-the-art Finite Element Method (FEM) solver, we developed a Flux-Corrected Transport (FCT) FEM scheme that uses explicit multi-step methods to integrate the solution in time. We showed that our solver produces high accuracy solutions without spurious oscillations for advection-dominant problems—thereby answering our research question **EMRQ3** (*Which state-of-the-art PDE solver is suitable for approximating the solution of the equation system employed in the Sprat Marine Ecosystem Model?*).

18.2.4 Evaluating the Sprat Marine Ecosystem Model

To evaluate the Sprat Model and answer the research question **EMRQ4** (*Which fish population dynamics within an ecosystem can be reproduced by the Sprat Marine Ecosystem Model?*), in Chapters 15 and 16, we applied our model to the eastern Scotian Shelf ecosystem with its intertwined direct and indirect fish stock interactions, which previously could not be modeled satisfactorily. The model was successful at reproducing the fish stock dynamics and proved to be a viable tool for mechanistically explaining the observed restructuring of the eastern Scotian Shelf ecosystem. Our simulation results provided new insights into the main drivers of regime shifts in marine ecosystems by suggesting that the prolonged collapse of fish stocks is likely not to be caused by the effects of fishing alone. The stocks rather have to be

already affected by other environmental pressures which make them more vulnerable to fishing.

Values for most parameters of the Sprat Model could directly be found in the literature and databases like FishBase (Froese and Pauly 2015). To achieve a good model fit with observed data, we mainly manipulated the parameters related to the foraging rate of the fish, to which the Sprat Model also turned out to be most sensitive; this answers research question **EMRQ4.1** (*How can the Sprat Model be parametrized and how sensitive is the model to small changes of the parameter values?*).

In a simulation experiment that did not resolve space, the otherwise spatially-explicit Sprat Model was not able to reproduce the fish stock dynamics of the eastern Scotian Shelf, which indicates that the spatial structure of fish stocks is important to model intertwined direct and indirect links in marine ecosystems. Careful examination of the spatial fish distributions in our model revealed that the predatory fish continuously fail to find the global maximum of their prey. In a model that does not consider space, the forage fish have no possibility of evading their predators, which explains why certain dynamics cannot be reproduced without considering space. This analysis answers our research question **EMRQ4.2** (*How important is the explicit representation of space in the Sprat Model for its predictive capabilities?*).

By running counterfactual simulations that assume the implementation of different fisheries management strategies in the past, we were able to quantitatively assess the impact of these strategies on the modeled fish stocks. We showed that among the management measures tested by us, only a Marine Protected Area (MPA) that is established early and covers a large area is effective at preventing the collapse of the commercially valuable benthic predator stocks. Furthermore, we identified certain management measures that bring about "ecological surprises" as they produce the exact opposite of their intended effect in our simulations. Our quantitative evaluation of different management scenarios answers the research question **EMRQ4.3** (*How can the Sprat Model be employed in the assessment of different fisheries management strategies?*).

By coupling the Sprat Model with the Atlantic Canada Model, we obtained a fully-developed end-to-end model. This allowed us to study the tempo-spatial distribution of fish and the impact of different environmental forcing factors on them in detail. The description of the coupling process

in Chapter 16 answers our research question **EMRQ4.4** (*How can the Sprat Model be coupled with existing biogeochemical ocean models?*).

All in all, the Sprat Marine Ecosystem Model proved to be a promising end-to-end modeling tool for mechanistically explaining intertwined direct and indirect interactions in marine ecosystems that could not be modeled previously. Furthermore, our model is able to quantitatively assess the impact of planned ecosystem management measures while considering changing environmental conditions and complex feedback mechanisms between ecosystem components. This ability to predict the long-term effects of management strategies is essential for successful ecosystem-based fisheries management.

Chapter 19

Future Work

This chapter presents an outlook on directions for future research regarding both the Sprat Approach (Section 19.1) and the Sprat Marine Ecosystem Model (Section 19.2).

19.1 The Sprat Approach

Future work for the Sprat Approach includes incorporating model-based performance engineering and testing techniques (Section 19.1.1), Sprat as a concept for Simulation Software as a Service (Section 19.1.2), extending the Sprat PDE DSL (Section 19.1.3) as well as the Sprat Ecosystem DSL (Section 19.1.4), and further evaluation of Sprat (Section 19.1.5).

19.1.1 Model-Based Performance Engineering and Testing

Since, with the Sprat Approach, all implementation artifacts are already high-level models, additional model-based techniques can be employed without any significant effort for the scientific software developers. Among these techniques are model-based performance engineering and model-based testing.

Often, performance optimization requires considerable changes in software design. Therefore, performance should already be considered in the design phase of software on an architectural level. However, as can be seen from Section 6.2.1 c) in Chapter 6, scientists typically develop software in a highly iterative manner with a focus on the scientific problems at hand, which implies that usually there is no distinct design phase. This problem can be circumvented by employing DSLs to construct models of the scientific software to be implemented—as it is always the case with the

Sprat Approach anyway. If the software is implemented using domain-level abstractions, model-based performance prediction and optimization techniques (Balsamo et al. 2004) can be employed without forcing the scientists to adopt rigid software processes. Since hardware resources are always limited and since especially in HPC, *performance really matters*, the application of such Software Performance Engineering (SPE) (Bondi 2014) approaches to systematically optimize the runtime efficiency of scientific software is a promising area for future work.

As discussed in Section 6.2.1 b) of Chapter 6, testing scientific software is essential for the reliability of its output but is often difficult because of the frequent lack of test oracles. A potential solution to this challenge lies in combining model-based testing (Schieferdecker 2012) with approaches that can perform effective testing without pre-defined oracles. Such approaches include the so-called *metamorphic testing* for non-deterministic programs (Guderlei and Mayer 2007), for example via machine learning techniques to automatically establish metamorphic relations (Kanewala and Bieman 2013).

19.1.2 Simulation Software as a Service

In the future, the Sprat Approach could be used to make simulation software development available as a Platform as a Service (Mell and Grance 2011). This could be achieved by creating web-based Integrated Development Environments (IDEs) for all the DSLs of a DSL hierarchy and by integrating these environments in a common service structure. In this way, the scientific software developers would not have to install any software on their computers and could collaborate on the development of the simulation software in a shared code repository directly through a web browser. We noticed that being able to use programming languages in a web frontend was highly welcomed in our online survey for the evaluation of the Sprat Ecosystem DSL because it prevents technical complexities with setting up a development environment and allows the scientists concentrate fully on the domain problem at hand (cf. Chapter 14).

Once the scientific software is ready to be used, simulation runs could also be specified via the same web interface utilized for implementing the software (e. g., via the Sprat Ecosystem DSL that serves to specify ecological simulations) and could be executed online in a cloud computing environ-

ment. Furthermore, the results from such simulations could directly be visualized, compared, and shared in the web-based simulation frontend and it would be traceable which specific version of the simulation code produced these results. This vision extends Sprat from a Platform as a Service to a concept for *Simulation Software as a Service*. The notion of Simulation Software as a Service has previously been explored by Guo et al. (2011), Juzna et al. (2014), and Madduri et al. (2015).

19.1.3 The Sprat PDE DSL

Future work for the Sprat PDE DSL includes implementing the DSL as an external language extension to C/C++ using a framework such as JetBrains MPS (cf. Chapter 2). The expert interviews conducted to evaluate the Sprat PDE DSL revealed that while all domain experts favor embedding the language into a GPL, some of the experts are interested in having access to generated source code, which is not possible with the current implementation. Implementing the Sprat PDE DSL as an external language extension would allow to access generated source code and would make it possible to achieve an even more "natural" concrete syntax. However, many computational scientists working in HPC are reluctant to employ "newer" technologies (such as external language extensions) that have not been tested by time and could theoretically stop being supported (cf. Section 6.2.2 b); note that the experts in our interviews did not share these concerns). Therefore, it would be interesting to study under which conditions such an external language extension would, in practice, actually be accepted by a broader range of domain experts.

Further areas of future work for the Sprat PDE DSL are adding support for more types of meshes and sparse matrices as well as introducing more Domain-Specific Optimizations (DSOs).

19.1.4 The Sprat Ecosystem DSL

As discussed in Chapter 8, a semi-graphical prototype of the Sprat Ecosystem DSL was already developed with JetBrains MPS (cf. Chapter 2) by a student as part of his master's thesis, which was supervised in the context of this thesis. Using pre-structured tables and advanced typesetting for mathematical formulas, the prototype showed promising results with regard

to making the Sprat Ecosystem DSL even more accessible to users with little programming experience. It would be worthwhile to study the influence of a semi-graphical notation on program comprehension in comparison to a purely textual DSL implementation via controlled experiments.

Additional future work includes implementing further generators for the Sprat Ecosystem DSL to facilitate generating parametrizations for other ecosystem models than the Sprat Marine Ecosystem Model. In this way, ecologists could target multiple ecosystem models with a single simulation description, which would allow them to easily compare the predictions of different models for a single simulation scenario.

19.1.5 Additional Evaluation Approaches

Further research has to be carried out with respect to the evaluation of the Sprat Approach. Ideally, a full cost-benefit analysis of Sprat should be conducted as part of a controlled experiment in which two teams of scientific software developers engineer the same simulation software with and without our MDSE approach. This would enable quantifying the short- and long-term trade-offs between the additional effort of developing custom DSLs and increased developer productivity.

Conducting such a full cost-benefit analysis, however, is costly and difficult (e. g., one has to control confounding variables such as developer expertise etc.). Therefore, it might be more promising to concentrate on another evaluation scenario: applying Sprat to already existing simulation software. Such a scenario would also allow to measure productivity with and without the Sprat Approach by comparing the productivity of modifying the existing simulation software before and after introducing Sprat to the development process.

A relatively simple way to apply the Sprat Approach to existing simulation software is to introduce a domain hierarchy for the software and assign DSLs only to the higher levels of the hierarchy (see Figure 19.1). This means that the DSLs on the higher hierarchy levels—which typically correspond to domains less affiliated with computing—act as something similar to a user-friendly programmable interface to the (legacy) simulation software. Thus, while relatively few code would have to be re-engineered, the simulation software becomes much more accessible to domain experts who want to use or customize it but are less proficient in programming.

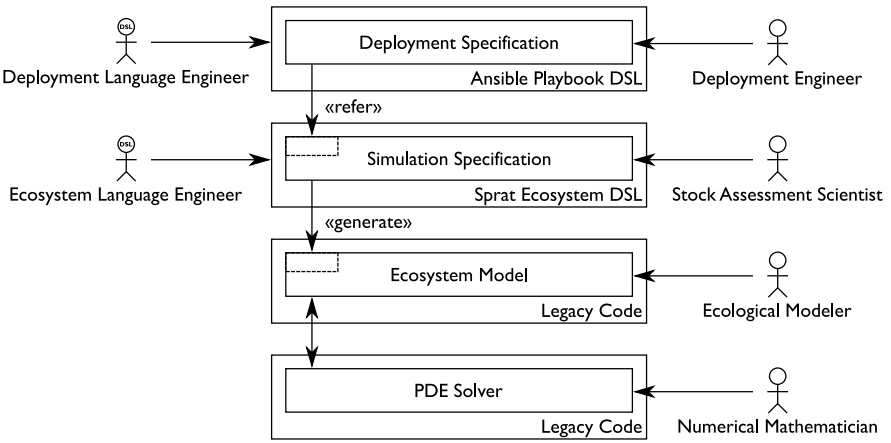


Figure 19.1. Domain hierarchy for a fictitious legacy marine ecosystem model.

Another approach to employing Sprat for legacy simulation software is to combine Sprat with existing model-driven software modernization techniques, such as DynaMod (Hoorn et al. 2011).

19.2 The Sprat Marine Ecosystem Model

Future research directions for the Sprat Marine Ecosystem Model include increasing its predictive capabilities for its application to the eastern Scotian Shelf (Section 19.2.1), advanced habitat modeling and vertical resolution of fish movement (Section 19.2.2), two-way coupling with different biogeochemical ocean models for new scenarios (Section 19.2.3), and inter-model comparisons (Section 19.2.4).

19.2.1 The Eastern Scotian Shelf Ecosystem

In Chapter 15, we already discussed that the predictive capabilities of the Sprat Model for the eastern Scotian Shelf could likely be increased by taking into account migration out of and back into the study area. This could compensate the unrealistically high decline of the forage fish stocks during

the winter months when zooplankton abundance on the shelf is low. In addition to explicitly modeling migration, future work includes quantifying the impact of the inner-annual variation of zooplankton abundance on the overall fish biomass levels. For this purpose, the model needs to be forced with multiple synthetic zooplankton abundance profiles that exhibit different amplitudes of inner-yearly variation (including constant zooplankton levels).

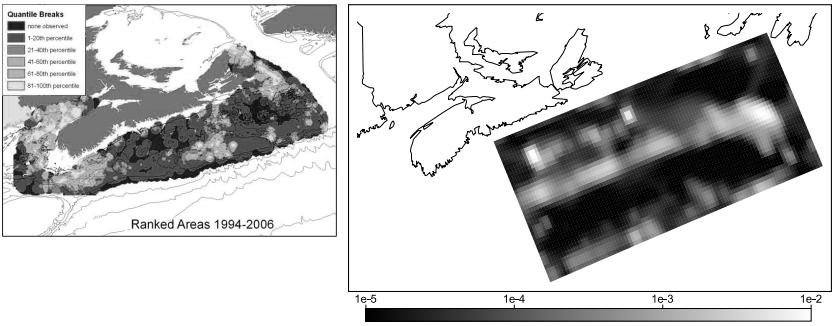
So far, in all applications of the Sprat Model to the Scotian Shelf, we have simulated only two fish species representing the benthic predator and the forage fish complexes. Future parametrizations of the model should include multiple fish species that correspond to the actual biological species found in the ecosystem. It would be interesting to see whether simulation results from the Sprat Model agree with the hypothesis that food webs with many weak to intermediate strength links are more stable than those with few strong trophic links (cf. McCann 2000; McCann et al. 1998; Neutel et al. 2002).

The temperature anomaly ΔT in the Sprat Model presently affects the background mortality of the fish, their metabolic rates, their foraging rates, and their growth. Since the model exhibits a high sensitivity to temperature, it would be worthwhile to consider including temperature in other processes such as the swimming speed or the reactive movement direction.

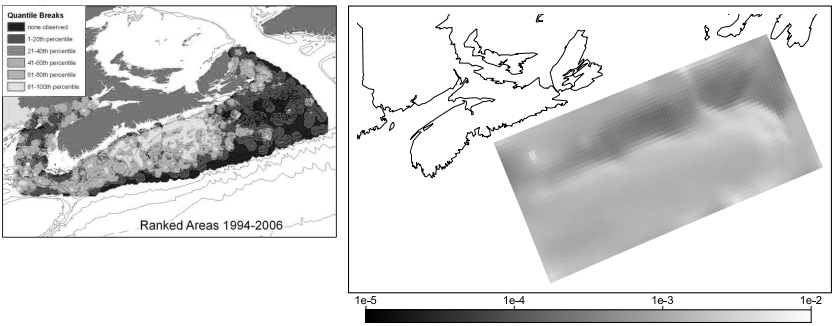
Currently, the Sprat model considers only the amount of available prey for reactive movement decisions. Since survival is essential for both the individual fish and the species as a whole, reactive movement could also incorporate the amount of potential predators at nearby locations. In this way, the Sprat Model could be used to evaluate the impact of placing emphasis either on predator evasion or on foraging in reactive movement strategies.

19.2.2 Habitat Modeling and Vertical Resolution

Figure 19.2 compares the observed spatial distributions of fish from the Scotian Shelf Groundfish Atlas (Horsman and Shackell 2009) with those predicted by the Sprat Model coupled with the Atlantic Canada Model. At least for herring/forage fish, we find a relatively good agreement of observed and simulated fish distributions directly on the eastern Scotian Shelf. However, our model suggests that most of the predator and forage



(a) Atlantic cod/predator fish complex



(b) Herring/forage fish complex

Figure 19.2. Comparison of observed spatial distributions of fish on the Scotian Shelf (left) with those predicted by the Sprat Model (right). The observed distributions have been taken from the Scotian Shelf Groundfish Atlas (Horsman and Shackell 2009) for the years 1994–2006. The brightness of each stratum of the mapped area corresponds to which quantile with respect to relative biomass it belongs to (e. g., the brightest area is comprised of the top 19 % of the strata with respect to relative fish biomass). The distributions from the Sprat Model show the average fish biomass in kg C m^{-2} for 1999–2003 for a simulation in which the model is coupled with the Atlantic Canada Model.

fish biomass should, on average, be found slightly off the shelf itself. We cannot compare this prediction of the Sprat Model with data from the Groundfish Atlas because the latter only samples fish on the shelf itself and, hence, does not contain information concerning fish in deeper waters. Nevertheless, contrary to the results from our model, we would expect that most of the benthic predators should be located in relatively shallow waters on the shelf within depths ranging from 150 to 200 meters (which is the depth preference of Atlantic cod; cf. Froese and Pauly 2015).

To increase the realism of spatial fish distributions in the Sprat Model, future research should focus on incorporating advanced predictive habitat models (Guisan and Zimmermann 2000) into the reactive movement component of fish in our model. Such habitat models do not only take into account the amount of potential prey and predators in nearby spatial locations but consider also other factors, such as water depth, temperature, salinity, etc., for computing the suitability of a certain location. Using multiple habitat models for different sub-populations of a fish stock would additionally allow the Sprat Model to resolve the stock structure of a species within an ecosystem.

As an alternative to including water depth into the reactive movement in a vertically integrated spatial stock model, one could also employ a version of the Sprat Model with three instead of two spatial dimensions. In this way, the model could capture the vertical behavior of fish, which would make it possible to study phenomena such as habitat compression induced by oxygen minimum zones (Stramma et al. 2012).

19.2.3 Two-Way Coupling With Different Ocean Models

We coupled the Sprat Model with the Atlantic Canada Model in a one-way fashion: our fish model was forced by the output of the biogeochemical ocean model but there was no feedback from the Sprat Model to the Atlantic Canada Model. Such feedback should include the reduction of zooplankton abundance due to grazing by fish and the return of nutrients to the biogeochemical model due to respiration/excretion by fish.

Since coupling the Sprat Model with the Atlantic Canada Model proved to be relatively easy (see Chapter 16), a two-way coupling of both models should not be much harder to achieve. We suggest to introduce this coupling in two stages: in the first stage, the Sprat Model still remains an

independent application of the Atlantic Canada Model and maintains its own solver infrastructure. At this stage, a coupler component is needed that mediates between the biogeochemical ocean model and the fish model to exchange data after every time step. In the second stage, the Sprat Model is integrated with the biogeochemical model and reuses the solver infrastructure provided by the implementation of the latter model. In this case, there is no need for a dedicated coupler component as data exchange is trivial to implement.

Of course, future work for the Sprat Model does not only include two-way coupling of the Sprat Model with the ROMS-based Atlantic Canada Model but also with other biogeochemical ocean models that may be based on different modeling systems. Coupling the Sprat Model with further biogeochemical models offers the opportunity to apply the Sprat Model to a broad range of ecosystems in order to try to explain the fish stock dynamics found in those systems. For example, the Sprat Model could be used to study the Peruvian upwelling system with its recurring regime shifts favoring either anchovies or sardines, which is in some ways similar to the regime shift on the Scotian Shelf that we analyzed in this thesis (Chavez et al. 2003).

19.2.4 Inter-Model Comparisons

It seems worthwhile to compare the Sprat Model with other fish models for end-to-end modeling such as SEAPODYM (Bertignac et al. 1998; Lehodey et al. 2008) or OSMOSE (Shin and Cury 2001) by parametrizing the models for defined scenarios and comparing their outputs. This would allow to study how different design choices of the models affect their predictions. Furthermore, comparing the Sprat Model with fish models that build upon other theoretical frameworks (such as ADR models and IBMs) makes it possible to investigate how the theoretical advantages of the population balance approach (see Chapter 10) translate into practice.

Such a model comparison could be conducted in the context of the evaluation of concretely planned fisheries management measures. Evaluating these measures with multiple fish models that rely on different modeling frameworks can either increase the confidence in the predictions of the models (if they agree) or point out uncertainties regarding particular aspects of

the future development of the ecosystem (if the predictions of the models disagree in a certain manner).

Part VI

Appendices

Appendix A

Parameters of the Sprat Marine Ecosystem Model

In Table A.1, we list the parameters of our NPZ model from Chapter 11. The global parameters of the Sprat Marine Ecosystem Model are presented in Table A.2 as well as its species parameters in Table A.3.

Table A.1. Parameters of the NPZ model for the Sprat Marine Ecosystem Model.

Symbol	Description	Unit
γ	Zooplankton assimilation efficiency	d.u. (%)
ι_s	Half saturation irradiance	MJ m^{-2}
V_m	Uptake rate of nutrients	s^{-1}
k_s	Half saturation nutrients	mmol N m^{-2}
R_m	Uptake rate of phytoplankton	s^{-1}
λ_s	Ivlev constant for grazing	$(\text{mmol N})^{-1}$ m^2
ε_P	Mortality rate phytoplankton	$(\text{mmol N})^{-1}$ m^2 s^{-1}
ε_Z	Mortality rate zooplankton	$(\text{mmol N})^{-1}$ m^2 s^{-1}
ε_f	Zooplankton mortality due to grazing	s^{-1}
η_r	Remineralization rate	s^{-1}
N_m	Maximum nutrient concentration	mmol N m^{-2}
C_c	Cloud coverage	d.u. (%)

Table A.2. Global parameters of the Sprat Marine Ecosystem Model.

Symbol	Description	Unit
τ	Minimal predator-prey mass ratio	d.u.
r_{view}	Perfect information radius	m
ζ_0	Scaling for feeding migration threshold	kg^{-1}

Table A.3. Species parameters of the Sprat Marine Ecosystem Model.

Symbol	Description	Unit
$C_{\text{C/dm}}$	Carbon to dry mass ratio	d.u.
$C_{\text{d/wm}}$	Dry to wet mass ratio	d.u.
w_{egg}	Egg dry mass	kg
w_{forage}	Minimum forage dry mass	kg
M_{plankton}	Maximum Z consumption wet mass	kg
M_{mature}	Wet mass at maturity	kg
M_{max}	Maximum wet mass	kg
a	Parameter a for length-weight relationship	d.u.
b	Parameter b for length-weight relationship	d.u.
ς	Cruise speed	BL s^{-1}
s	Location of spawning ground	(m, m)
S	Mating season	n.a.
θ_{yolk}	Duration till larvae reach w_{forage}	s
E	Assimilation efficiency	d.u.
ε_B	Background mortality rate	$\text{m}^2\ \text{s}^{-1}$
ζ_B	Scaling of mortality due to temperature	$\text{s}^{-1}\ {}^{\circ}\text{C}^{-1}$
Φ	Net wet mass fecundity	kg^{-1}
η	Predation rate	s^{-1}
F_0	Predation half saturation	kg m^{-2}
μ	Zooplankton grazing rate	s^{-1}
Z_0	Zooplankton grazing half saturation	kg m^{-2}

Appendix B

Specification of the Meta-Model of the Sprat Ecosystem DSL

This chapter contains the full Object-Z specification for the Sprat Ecosystem DSL meta-model. For explanations regarding the specification, see Section 8.3.1 of Chapter 8.

B.1 Expressions With Units

$[NAME, UNIT\_TYPE, FP\_NUMBER]$

$FP\_EXPR ::=$
$\quad fp\_terminal\_number\langle\langle FP\_NUMBER\rangle\rangle \mid$
$\quad fp\_unary\_plus\langle\langle FP\_EXPR\rangle\rangle \mid$
$\quad fp\_unary\_minus\langle\langle FP\_EXPR\rangle\rangle \mid$
$\quad fp\_plus\langle\langle FP\_EXPR \times FP\_EXPR\rangle\rangle \mid$
$\quad fp\_minus\langle\langle FP\_EXPR \times FP\_EXPR\rangle\rangle \mid$
$\quad fp\_times\langle\langle FP\_EXPR \times FP\_EXPR\rangle\rangle \mid$
$\quad fp\_divide\langle\langle FP\_EXPR \times FP\_EXPR\rangle\rangle$

Unit

name : *NAME*
type : *UNIT_TYPE*
conversionFactor : *FP_NUMBER*

Expression_With_Unit

expr : *FP_EXPR*
unit : *Unit*

B.2 Properties

Property

key : *NAME*
unitType : *UNIT_TYPE*
unitTypeModifier : *UNIT_TYPE*
valueExpr : *Expression_With_Unit*
valueModifierExpr : *Expression_With_Unit*
valueName : *NAME*

$valueExpr.unit.type = unitType$
$valueModifierExpr.unit.type = unitTypeModifier$

Property_Description

key : *NAME*
unitType : *UNIT_TYPE*
unitTypeModifier : *UNIT_TYPE*

Name_Property

Property

Name_Property_Description

Property_Description

Quantity_Property

Property

Quantity_Property_Description

Property_Description

Quantity_Property_With_Modifier

Quantity_Property

Quantity_Property_With_Modifier_Description

Property_Description

B.3 Data Aggregation Statements

FP_RANGE_EXPR ::=
ran_terminal_number⟨⟨*FP_NUMBER*⟩⟩ |
ran_unary_plus⟨⟨*FP_RANGE_EXPR*⟩⟩ |
ran_unary_minus⟨⟨*FP_RANGE_EXPR*⟩⟩ |
ran_plus⟨⟨*FP_RANGE_EXPR* × *FP_RANGE_EXPR*⟩⟩ |
ran_minus⟨⟨*FP_RANGE_EXPR* × *FP_RANGE_EXPR*⟩⟩ |
ran_times⟨⟨*FP_RANGE_EXPR* × *FP_RANGE_EXPR*⟩⟩ |
ran_divide⟨⟨*FP_RANGE_EXPR* × *FP_RANGE_EXPR*⟩⟩ |
ran_range⟨⟨*FP_EXPR* × *FP_EXPR*⟩⟩

Record_Function_Argument

name : *NAME*
valueExpr : *FP_EXPR*
valueRangeExpr : *FP_RANGE_EXPR*
valueName : *NAME*

Record_Function_Argument_Description

name : *NAME*

RF_Expression_Argument

Record_Function_Argument

RF_Expression_Argument_Description

Record_Function_Argument_Description

RF_Range_Expression_Argument

Record_Function_Argument

RF_Range_Expression_Argument_Description

Record_Function_Argument_Description

RF_Name_Argument

Record_Function_Argument

RF_Name_Argument_Description

Record_Function_Argument_Description

Record_Function_Description

$name : NAME$
$args : \text{seq}\downarrow Record\_Function\_Argument\_Description$

This makes *Record_Function_Argument_Description* effectively abstract.

$\forall a : args \bullet second(a) \in RF\_Expression\_Argument\_Description \cup$
$RF\_Range\_Expression\_Argument\_Description \cup$
$RF\_Name\_Argument\_Description$

$validRecordFunctions : \mathbb{F}_1\ Record\_Function\_Description$

Record_Function_Call

$$
\begin{array}{l}
name : NAME \\
args : \text{seq}\downarrow Record\_Function\_Argument©
\end{array}
$$

Record function calls must match one of the record function descriptions given in *validRecordFunctions*.

$$
\begin{array}{l}
\exists f : validRecordFunctions \bullet name = f.name \land \\
\#args = \#f.args \land \forall a : f.args \bullet \\
\quad ((args(first(a))).name = (second(a)).name \land \\
\quad (second(a) \in RF\_Expression\_Argument\_Description \implies \\
\qquad args(first(a)) \in RF\_Expression\_Argument) \land \\
\quad (second(a) \in RF\_Range\_Expression\_Argument\_Description \implies \\
\qquad args(first(a)) \in RF\_Range\_Expression\_Argument) \land \\
\quad (second(a) \in RF\_Name\_Argument\_Description \implies \\
\qquad args(first(a)) \in RF\_Name\_Argument))
\end{array}
$$

$$
\begin{array}{l}
RECORD\_EXPR ::= \\
\quad rec\_terminal\_number\langle\langle FP\_NUMBER\rangle\rangle \mid \\
\quad rec\_terminal\_function\langle\langle Record\_Function\_Call\rangle\rangle \mid \\
\quad rec\_unary\_plus\langle\langle RECORD\_EXPR\rangle\rangle \mid \\
\quad rec\_unary\_minus\langle\langle RECORD\_EXPR\rangle\rangle \mid \\
\quad rec\_plus\langle\langle RECORD\_EXPR \times RECORD\_EXPR\rangle\rangle \mid \\
\quad rec\_minus\langle\langle RECORD\_EXPR \times RECORD\_EXPR\rangle\rangle \mid \\
\quad rec\_times\langle\langle RECORD\_EXPR \times RECORD\_EXPR\rangle\rangle \mid \\
\quad rec\_divide\langle\langle RECORD\_EXPR \times RECORD\_EXPR\rangle\rangle
\end{array}
$$

$$
[TIME\_SEQUENCE]
$$

Data_Aggregation_Statement

$$
\begin{array}{l}
name : NAME \\
when : TIME\_SEQUENCE \\
what : RECORD\_EXPR
\end{array}
$$

B.4 Ecosystem Simulation Description

$$
\begin{array}{l}
EnvironmentProperties : \mathbb{F}_1 \downarrow Property\_Description \\
SpeciesProperties : \mathbb{F}_1 \downarrow Property\_Description \\
InputProperties : \mathbb{F}_1 \downarrow Property\_Description \\
OutputProperties : \mathbb{F}_1 \downarrow Property\_Description \\
\hline
\forall a, b : EnvironmentProperties \bullet a.key = b.key \implies a = b \\
\forall a, b : SpeciesProperties \bullet a.key = b.key \implies a = b \\
\forall a, b : InputProperties \bullet a.key = b.key \implies a = b \\
\forall a, b : OutputProperties \bullet a.key = b.key \implies a = b
\end{array}
$$

Ecosystem_Environment

$properties : \mathbb{F}_1 \downarrow Property©$

The *properties* are exactly those described by *EnvironmentProperties*.

$\#properties = \#EnvironmentProperties$

$\forall p : properties \bullet \exists q : EnvironmentProperties \bullet$
$p.key = q.key \land$
$(p \in Name\_Property \implies q \in Name\_Property\_Description) \land$
$(p \in Quantity\_Property \implies q \in Quantity\_Property\_Description \land$
$\quad p.unitType = q.unitType) \land$
$(p \in Quantity\_Property\_With\_Modifier \implies$
$\quad q \in Quantity\_Property\_With\_Modifier\_Description \land$
$\quad p.unitType = q.unitType \land p.unitTypeModifier = q.unitTypeModifier)$

Species

$properties : \mathbb{F}_1 \downarrow Property©$

$\#properties = \#SpeciesProperties$

$\forall p : properties \bullet \exists q : SpeciesProperties \bullet p.key = q.key \land$
$(p \in Name\_Property \implies q \in Name\_Property\_Description) \land$
$(p \in Quantity\_Property \implies q \in Quantity\_Property\_Description \land$
$\quad p.unitType = q.unitType) \land$
$(p \in Quantity\_Property\_With\_Modifier \implies$
$\quad q \in Quantity\_Property\_With\_Modifier\_Description \land$
$\quad p.unitType = q.unitType \land p.unitTypeModifier = q.unitTypeModifier)$

Simulation_Input

$$
\begin{array}{|l}
\hline
properties : \mathbb{F}_1 \downarrow Property© \\
\hline
\#properties = \#InputProperties \\
\forall p : properties \bullet \exists q : InputProperties \bullet p.key = q.key \land \\
(p \in Name\_Property \implies q \in Name\_Property\_Description) \land \\
(p \in Quantity\_Property \implies q \in Quantity\_Property\_Description \land \\
\quad p.unitType = q.unitType) \land \\
(p \in Quantity\_Property\_With\_Modifier \implies \\
\quad q \in Quantity\_Property\_With\_Modifier\_Description \land \\
\quad p.unitType = q.unitType \land p.unitTypeModifier = q.unitTypeModifier) \\
\hline
\end{array}
$$

Simulation_Output

$$
\begin{array}{|l}
\hline
properties : \mathbb{F}_1 \downarrow Property© \\
aggregation : \mathbb{F}\, Data\_Aggregation\_Statement© \\
\hline
\#properties = \#OutputProperties \\
\forall p : properties \bullet \exists q : OutputProperties \bullet p.key = q.key \land \\
(p \in Name\_Property \implies q \in Name\_Property\_Description) \land \\
(p \in Quantity\_Property \implies q \in Quantity\_Property\_Description \land \\
\quad p.unitType = q.unitType) \land \\
(p \in Quantity\_Property\_With\_Modifier \implies \\
\quad q \in Quantity\_Property\_With\_Modifier\_Description \land \\
\quad p.unitType = q.unitType \land p.unitTypeModifier = q.unitTypeModifier) \\
\hline
\end{array}
$$

Ecosystem_Simulation

ecosystem : *Ecosystem_Environment*©
species : $\mathbb{F}_1$ *Species*©
input : *Simulation_Input*©
output : *Simulation_Output*©

Appendix C

Material for the Evaluation of the Sprat Ecosystem DSL

This chapter contains supplementary material (questionnaires and task descriptions) for the evaluation of the Sprat Ecosystem DSL in Chapter 14. The sections of this chapter correspond to the pages of our online survey. A screenshot of the introductory page of the survey can be found in Figure C.1.

C.1 Page 1: Academic Background

We would like you to state your name and e-mail address so that we could contact you if we need any clarification of your input. You may, however, choose to leave these two fields blank.

The data you provide during the survey will only be used in the context of the research project mentioned before and personal information will never be disclosed to third parties.

1. Your name [free-form question; optional]
2. Your e-mail address [free-form question; optional]
3. What is your highest educational qualification? (e. g., Bachelor, Master, Ph.D. etc.) [free-form question]
4. In which discipline did you acquire this qualification? [free-form question]
5. What is your current occupation? (e. g., bachelor/master/Ph.D. student, post-doc, professor etc.) [free-form question]

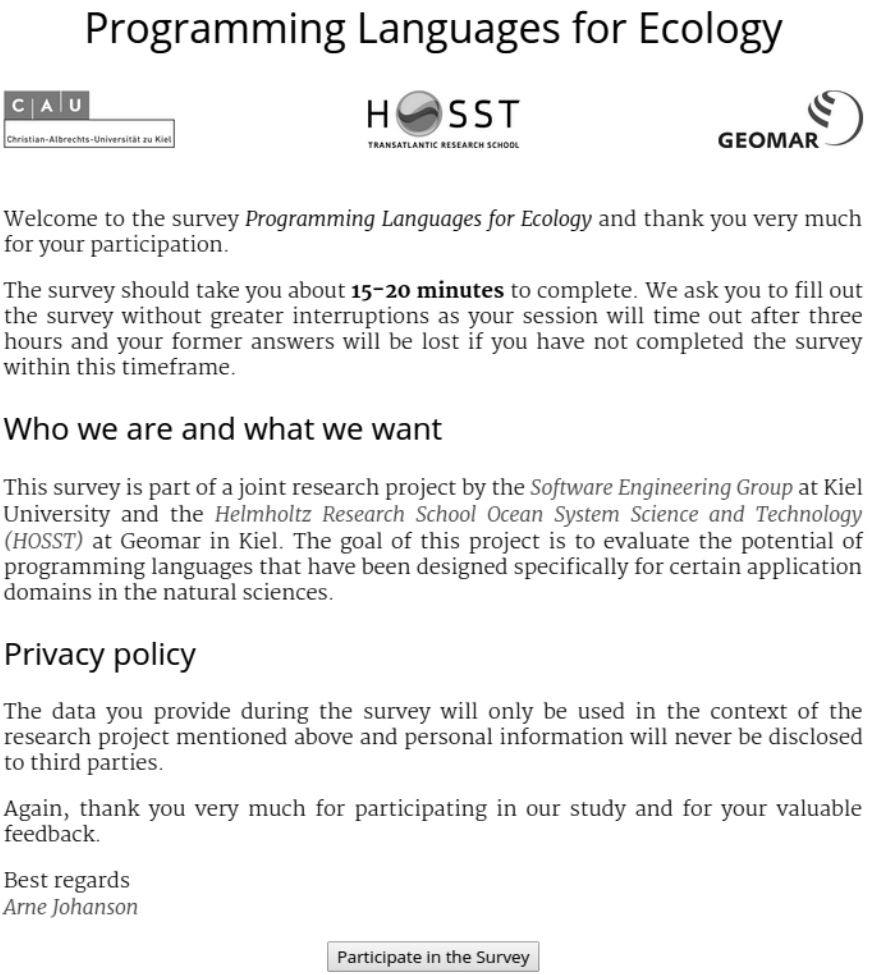


Figure C.1. Introduction to the online survey.

C.2 Page 2: Programming Experience

1. Have you ever written a computer program using a programming language? (e.g., C, Matlab, Java, R etc.) [yes/no]
2. Have you ever been required to write a computer program for work/for your studies? [yes/no]
3. If applicable, which programming languages (including languages for special purposes like R or Matlab) have you used so far? [free-form question; optional]
4. How would you rate the level of your programming skills? [non-programmer (never written a program, difficulty with reading), beginner (problems writing easy programs, can read), regular (can write basic programs, no problems reading), advanced (can write complex programs), expert (years of experience)]
5. How experienced are you with the programming language C/C++? [non-programmer (never written a program, difficulty with reading), beginner (problems writing easy programs, can read), regular (can write basic programs, no problems reading), advanced (can write complex programs), expert (years of experience)]
6. How interested are you in programming in general? [6-point Likert scale]
7. If applicable, what was the primary source of information for learning how to program? (e. g., internet, book, a colleague/friend etc.) [free-form question; optional]
8. If applicable: I am confident that the programs I have written are correct (i. e., that they actually compute what I intended them to). [6-point Likert scale]

C.3 Page 3: General Introduction to the Exercises

On the following pages you will be given a total of five small programming exercises. Please try to solve these exercises even if you are not familiar with the programming languages or with programming in general.

Below the introduction to each task you will find a button to start the corresponding exercise. Please make sure you have read and understood the instructions *before* starting the exercise. The instructions will remain visible, though, while you complete the exercise.

Please do not use other material to solve the exercises than the information given to you in the description of the exercise (e. g., do not search the Internet).

C.4 Page 4/5: Exercise: Parametrization With EcoDSL/C++

The exercises of these pages are shown in Figures C.2 and C.3.

C.5 Page 6/7: Exercise: Recording With EcoDSL/C++

The exercises of these pages are shown in Figures C.4 and C.5.

C.6 Page 8: Feedback: Parametrization With EcoDSL/C++

Please answer the following questions based on the tasks you just performed:

1. Concepts of marine ecosystem simulations—such as fish species—can easily be specified with **EcoDSL**. [6-point Likert scale]
2. Concepts of marine ecosystem simulations—such as fish species—can easily be specified with **C++**. [6-point Likert scale]
3. **EcoDSL** seems simple to use. [6-point Likert scale]
4. **C++** seems simple to use. [6-point Likert scale]
5. The **EcoDSL** programs were easy to understand. [6-point Likert scale]
6. The **C++** programs were easy to understand. [6-point Likert scale]

Exercise: Parameterization with EcoDSL

In this exercise you will program the parameters of a marine ecosystem simulation using a programming language that has been designed specifically for this task. We will refer to this programming language as *EcoDSL*.

EcoDSL Parameterization Example

The following piece of code gives you an example for using EcoDSL. It specifies ecosystem parameters and introduces two fish species (cod and herring) as well as their parameters.

```
Ecosystem {
    OxygenLevel: 0.1 [mmol/l]
    WindSpeed: 12 [km/h]
}
Species cod {
    MaxMass: 1 [kg]
    IngestionExponent: 0.5
}
Species herring {
    MaxMass: 100 [g]
    IngestionExponent: 0.75
}
```

Note that the decimal separator of a rational number has to be a period ('.') and *cannot* be replaced by, e.g., a comma (',') as it is done in some locales.

Your Task

Your task is to modify the EcoDSL example in order to parameterize a simulation that features two fish species (cod and herring) and implements the ecosystem as well as the species parameters given in the following two tables.

Ecosystem parameter	**Value**
Area	1000 m^2
PredatorPreyRatio	3.5
SeabirdPredationFactor	0.025 d^-1

Species parameter	**Value for cod**	**Value for herring**
MaxLength	1.2 m	45 cm
MaxSwimmingSpeed	2.52 km/h	40 cm/s
GrowthRate	0.1 d^-1	0.15 d^-1

Figure C.2. Parametrization exercise with EcoDSL.

Exercise: Parameterization with C++

In this exercise you will program the parameters of a marine ecosystem simulation using the programming language C++.

C++ Parameterization Example

The following piece of code gives you an example of how to use C++ to specify ecosystem parameters (`OxygenLevel` and `WindSpeed`) as well as parameters for two fish species (`MaxMass` and `IngestionExponent`).

```
struct ModelParameters {
    static constexpr double OxygenLevel = 0.1;
    static constexpr double WindSpeed = 12.0;
    const real MaxMass[2];
    const real IngestionExponent[2];
    ModelParameters() :
        MaxMass {1.0, 0.1},
        IngestionExponent {0.5, 0.75}
    {}
};
```

Note that the decimal separator of a rational number has to be a period ('.') and *cannot* be replaced by, e.g., a comma (',') as it is done in some locales.

Your Task

Your task is to modify the C++ example in order to parameterize a simulation that features two fish species (cod and herring) and implements the ecosystem as well as the species parameters given in the two tables below.

Note: as C++ has no built-in units for numerical quantities, you have to pay close attention not to make conversion errors. You have to convert each quantity to the unit that the C++ model expects (specified in the tables below). To convert between units you can write standard arithmetic expressions such as `0.1 * 1000.0` to convert, e.g., 0.1 kilogram to gram.

Ecosystem parameter	Unit assumed in the C++ code	Value
Area	m^2	1000 m^2
PredatorPreyRatio	dimensionless	3.5
SeabirdPredationFactor	d^-1	0.025 d^-1

Species parameter	Unit assumed in the C++ code	Value for cod (species 1)	Value for herring (species 2)
MaxLength	m	1.2 m	45 cm
MaxSwimmingSpeed	m/s	2.52 km/h	40 cm/s
GrowthRate	d^-1	0.1 d^-1	0.15 d^-1

Figure C.3. Parametrization exercise with C++.

Exercise: Recording with EcoDSL

Once you start this exercise, you will be given a piece of EcoDSL code that aggregates and stores selected data for an ecosystem simulation. For this exercise, assume that two fish species (cod and herring) have already been parameterized.

Your Task

Your task is to modify the existing code in a way that the following data is recorded *instead* of the data currently recorded:

1. Record the **dry biomass of herring every 12 hours**.
2. **After the simulation**, record the **total wet fish biomass** of the system (i.e., the wet biomass of cod and herring combined).

Additional Information

The following so-called *record functions* are available to specify which data to aggregate:

```
wetBiomass(species = ...)
matureWetBiomass(species = ...)
dryBiomass(species = ...)
matureDryBiomass(species = ...)
fecundity(species = ...)
```

You can combine various record functions using standard arithmetic operations (`+`, `-`, `*`, `/`) to form *record expressions* such as:

```
dryBiomass(species = cod) + dryBiomass(species = herring)
```

In order to specify *when* in the course of the simulation the data is recorded, the following identifiers are available:

```
@beforeSimulation
@afterSimulation
@every(... [h])
```

Figure C.4. Recording exercise with EcoDSL.

Exercise: Recording with C++

Once you start this exercise, you will be given a piece of C++ code that aggregates and stores selected data for an ecosystem simulation. For this exercise, assume that two fish species (cod and herring) have already been parameterized.

Your Task

Your task is to modify the existing code in a way that the following data is recorded *instead* of the data currently recorded:

1. Record the **dry biomass of herring every 12 hours**.
2. **After the simulation**, record the **total wet fish biomass** of the system (i.e., the wet biomass of cod and herring combined).

Note, that in C++ the species are addressed not by name but by an index (cod = 0, herring = 1).

Additional Information

The following so-called *record functions* are available to specify which data to aggregate (where you have to replace `speciesIndex` with the appropriate index of the fish species):

```
wetBiomass(dof, speciesIndex)
matureWetBiomass(dof, speciesIndex)
dryBiomass(dof, speciesIndex)
matureDryBiomass(dof, speciesIndex)
fecundity(dof, speciesIndex)
```

You can combine various record functions using standard arithmetic operations (`+`, `-`, `*`, `/`) to form *record expressions* such as:

```
dryBiomass(dof, 0) + dryBiomass(dof, 1)
```

In order to specify *when* in the course of the simulation the data is recorded, the following identifiers are available:

```
RecordWhen::BEFORESIMULATION
RecordWhen::AFTERSIMULATION
RecordWhen::EVERY
```

Figure C.5. Recording exercise with C++.

7. **EcoDSL** seems too technical to specify concepts of marine ecosystem simulations. [6-point Likert scale]
8. **C++** seems too technical to specify concepts of marine ecosystem simulations. [6-point Likert scale]
9. It is easy to introduce changes to existing **EcoDSL** programs. [6-point Likert scale]
10. It is easy to introduce changes to existing **C++** programs. [6-point Likert scale]
11. I was able to understand the meaning of **EcoDSL** source code quickly. [6-point Likert scale]
12. I was able to understand the meaning of **C++** source code quickly. [6-point Likert scale]
13. Which difficulties did you have while using EcoDSL? [free-form question; optional]
14. Which suggestions do you have for improving the EcoDSL programming language? [free-form question; optional]

C.7 Page 9: Exercise: Modifying EcoDSL

The exercise of this page is shown in Figure C.6.

C.8 Page 10: Feedback: Modifying EcoDSL

Please answer the following questions based on the task you just performed:

1. How would you rate your expertise regarding the Java programming language? [non-programmer (never written a program, difficulty with reading), beginner (problems writing easy programs, can read), regular (can write basic programs, no problems reading), advanced (can write complex programs), expert (years of experience)]
2. The meaning of the Java code was easy to comprehend. [6-point Likert scale]

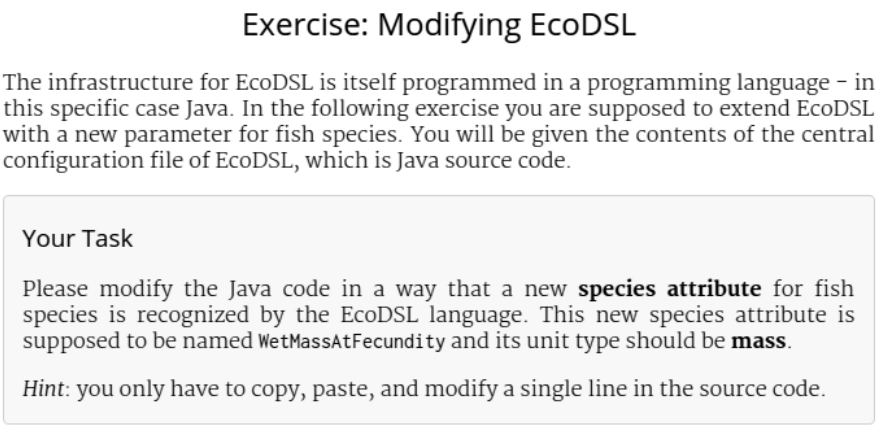


Figure C.6. Exercise in modifying EcoDSL.

3. It was easy to recognize the statements in the Java code that were relevant for completing the task. [6-point Likert scale]

Appendix D

Material for the Evaluation of the Sprat PDE DSL

This chapter contains supplementary material (interview guides and analysis categories) for the evaluation of the Sprat PDE DSL in Chapter 13.

D.1 Interview Guide for Domain Experts

1. What is your professional background? What kind of programming experience do you have?
2. Which software do you develop at least partially in you research group and how is this software utilized? In which context is the software developed? (E. g., for a single publication?)
3. Which programming languages and libraries/frameworks do you use in the development and why have you selected them? Which role does longevity and the number of dependencies play here?
4. How do you approach software development methodically?
5. How do you verify the correctness of your software and its output?
6. How long is the software developed/used? How do new employees accustom themselves with the software? Do you maintain a documentation for the software?
7. What kind of learning material would you like to be provided with if you wanted to learn how to use the Sprat PDE DSL?

8. Does the Sprat PDE DSL feature all the necessary concepts for implementing FEM solvers? Would you like to add any concepts that are frequently used in the development of PDE solvers?
9. How would you rate the simplicity or accessibility of the Sprat PDE DSL? How would you judge the required effort to become proficient in the language?
10. How closely does the syntax of the Sprat PDE DSL resemble the typical representation of algorithms in your discipline? Do you have any suggestions for improving the syntax?
11. How do you rate the facilities provided by the Sprat PDE DSL to practice defensive programming?
12. How often do you have to experiment with implementations of algorithms? How do you think could the Sprat PDE DSL support you along this direction?
13. How well do you think can programs written with the Sprat PDE DSL be maintained?
14. How would you judge the potential benefits of the Sprat PDE DSL for your discipline and your personal work?
15. How long do you think C/C++ will play a relevant role for developing new software in your discipline?
16. If you wanted to use the Sprat PDE DSL, would you view its dependencies as a problem?
17. Would you rather have the Sprat PDE DSL to be internal or external?
18. Could you imagine to use the Sprat PDE DSL for your own software development? What are the arguments for and against?

D.2 Interview Guide for DSL Developers

1. What is your professional background? What kind of programming experience do you have?

2. What kind of learning material would you like to be provided with if you wanted to learn how to use the Sprat PDE DSL or what kind of material would you provide to your users if you had developed the DSL?
3. How would you rate the simplicity or accessibility of the Sprat PDE DSL? Do you have any suggestions for improving the syntax or do you notice anything that seems odd to you?
4. On a scale from taciturn to talkative, where would you locate the Sprat PDE DSL? What is your opinion on that?
5. How do you rate the quality of the separation between topology and algorithm implementation that can be achieved with the Sprat PDE DSL? Do you recognize any violations of this separation?
6. How do you rate the use of Boost Proto for embedding the Sprat PDE DSL into C++? Do you know any alternative technologies?
7. How do you rate the architectural quality of the embedding of the Sprat PDE DSL? How well do you think can the implementation be maintained?
8. How would you yourself have embedded the language into C++? Which principles/methodology would you follow?

D.3 Categories for Coding the Interviews

Background:

1. Professional background
2. Programming experience

Software development in the work of the domain experts:

1. Purpose of the software
2. A single software library vs. multiple projects
3. Programming languages
4. Dependencies
5. Approach to the implementation of algorithms
6. Verification

7. Documentation

Learning material for DSLs:

1. Reference
2. Motivation/overview
3. Examples

Meta-model and syntax:

1. Conformity and orthogonality of the meta-model
2. Quality of the syntax
3. Prescribed vs. flexible program structure
4. Maintainability of DSL programs
5. Usefulness of the DSL

Technical implementation:

1. C++ as host language
2. Dependencies of the DSL
3. Maintainability of the DSL implementation
4. Internal vs. external DSL

Bibliography

Abrahams, David and Aleksey Gurtovoy (2004). *C++ template metaprogramming: concepts, tools, and techniques from Boost and beyond*. Addison-Wesley.

Alexander, K. and S.M. Easterbrook (2015). "The software architecture of climate models: a graphical comparison of CMIP5 and EMICAR5 configurations". In: *Geoscientific Model Development Discussions* 8.1, pp. 351–379.

Almorsy, Mohamed, John Grundy, Richard Sadus, Willem van Straten, David G. Barnes, and Owen Kaluza (2013). "A suite of domain-specific visual languages for scientific software application modelling". In: *Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2013*. IEEE, pp. 91–94.

Alnæs, M. S. (2012). "UFL: a finite element form language". In: *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Ed. by A. Logg, K.A. Mardal, and G. Wells. Vol. 84. LNCSE. Springer. Chap. 17, pp. 299–334.

Ansible Incorporated (2015). *Ansible documentation*. URL: http://docs.ansible.com/.

Apple Incorporated (2015). *The Swift programming language – language reference*. URL: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AboutTheLanguageReference.html.

April, A. and A. Abran (2012). *Software maintenance management: evaluation and continuous improvement*. Wiley.

Aranda, Jorge (2012). *Software Carpentry assessment report*. Tech. rep. Software Carpentry. URL: http://software-carpentry.org/bib/aranda-assessment-2012-07.pdf.

Balsamo, Simonetta, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni (2004). "Model-based performance prediction in software development: a survey". In: *Software Engineering* 30.5, pp. 295–310.

Baretta, J.W., W. Ebenhöh, and P. Ruardij (1995). "The European Regional Seas Ecosystem Model, a complex marine ecosystem model". In: *Netherlands Journal of Sea Research* 33.3, pp. 233–246.

Baretta-Bekker, J.G., J.W. Baretta, and W. Ebenhöh (1997). "Microbial dynamics in the marine ecosystem model ERSEM II with decoupled carbon assimilation and nutrient uptake". In: *Journal of Sea Research* 38.3, pp. 195–211.

Barnes, Nick (2010). "Publish your computer code: it is good enough". In: *Nature* 467.7317, pp. 753–753.

Basili, Victor R., Gianluigi Caldiera, and H. Dieter Rombach (1994). "Goal Question Metric Paradigm". In: *Encyclopedia of Software Engineering*. Wiley, pp. 528–532.

Basili, Victor R., Daniela Cruzes, Jeffrey C. Carver, Lorin M. Hochstein, Jeffrey K. Hollingsworth, Marvin V. Zelkowitz, et al. (2008). "Understanding the high-performance-computing community: a software engineer's perspective". In: *IEEE Software* 25.4, pp. 29–36.

Beck, Kent (2000). *Extreme Programming explained: embrace change*. Addison-Wesley.

Bell, Gordon, Tony Hey, and Alex Szalay (2009). "Beyond the data deluge". In: *Science* 323.5919, pp. 1297–1298.

Benioff, Marc R. et al. (2005). *Computational science: ensuring America's competitiveness*. Tech. rep. President's Information Technology Advisory Committee (PITAC).

Bertignac, M., P. Lehodey, and J. Hampton (1998). "A spatial population dynamics simulation model of tropical tunas using a habitat index based on environmental parameters". In: *Fisheries Oceanography* 7.3-4, pp. 326–334.

Bettini, Lorenzo (2013). *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing.

Beverton, R.J.H. and S.J. Holt (1957). *On the dynamics of exploited fish populations*. Vol. XIX. Fishery Investigations Series II. London: United Kingdom Ministry of Agriculture, Fisheries and Food.

Boisclair, D. and M. Tang (1993). "Empirical analysis of the influence of swimming pattern on the net energetic cost of swimming in fishes". In: *Journal of Fish Biology* 42.2, pp. 169–183.

Bondi, A. B. (2014). *Foundations of software and system performance engineering: process, performance modeling, requirements, testing, scalability, and practice*. Addison-Wesley.

Boris, Jay P. and David L. Book (1973). "Flux-corrected transport. I. SHASTA, a fluid transport algorithm that works". In: *Journal of Computational Physics* 11.1, pp. 38–69.

— (1976). "Flux-corrected transport. III. Minimal-error FCT algorithms". In: *Journal of Computational Physics* 20.4, pp. 397–431.

Bortz, Jürgen and Nicola Döring (2006). *Forschungsmethoden und Evaluation für Human- und Sozialwissenschaftler*. 4th ed. Springer.

Bortz, Jürgen and Christof Schuster (2010). *Statistik für Human- und Sozialwissenschaftler*. 7th ed. Springer.

Boyer, Timothy P., John I. Antonov, Olga K. Baranova, Hernan E. Garcia, Daphne R. Johnson, Alexey V. Mishonov, et al. (2013). *World Ocean Database 2013*. NOAA.

Brambilla, Marco, Jordi Cabot, and Manuel Wimmer (2012). *Model-driven software engineering in practice*. Synthesis Lectures on Software Engineering 1. Morgan & Claypool.

Brander, Keith M. (2007). "Global fish production and climate change". In: *Proceedings of the National Academy of Sciences* 104.50, pp. 19709–19714.

Brennan, Catherine E., Laura Bianucci, and Katja Fennel (2014). "Sensitivity of northwestern North Atlantic shelf circulation to surface and boundary forcing: a regional model assessment". In: *EGU General Assembly Conference Abstracts*. Vol. 16, p. 3863.

Brenner, Susanne and L. Ridgway Scott (2008). *The mathematical theory of finite element methods*. 3rd ed. Springer.

Brock, Thomas D. (1981). "Calculating solar radiation for ecological studies". In: *Ecological Modelling* 14.1, pp. 1–19.

Brown, Culum, Kevin Laland, and Jens Krause (2008). *Fish cognition and behavior*. John Wiley & Sons.

Brown, J., M.G. Knepley, and B.F. Smith (2015). "Run-time extensibility and librarization of simulation software". In: *Computing in Science & Engineering* 17.1, pp. 38–45.

Bruel, Jean-Michel, Benoit Combemale, Ileana Ober, and Hélene Raynal (2015). "MDE in practice for computational science". In: *Proceedings of the International Conference on Computational Science '15*, pp. 1–10.

Buitenhuis, Erik, M. Vogt, R. Moriarty, N. Bednarsek, S.C. Doney, K. Leblanc, et al. (2013). "MAREDAT: towards a world atlas of MARine Ecosystem DATa". In: *Earth System Science Data* 5, pp. 227–239.

Bundy, Alida (2005). "Structure and functioning of the eastern Scotian Shelf ecosystem before and after the collapse of groundfish stocks in the early 1990s". In: *Canadian Journal of Fisheries and Aquatic Sciences* 62.7, pp. 1453–1473.

Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, et al. (1996). *Pattern-oriented software architecture, volume 1: a system of patterns*. Wiley.

Buttari, Alfredo, Jack Dongarra, Jakub Kurzak, Julien Langou, Piotr Luszczek, and Stanimire Tomov (2007). "The impact of multicore on math software". In: *Applied Parallel Computing. State of the Art in Scientific Computing*. Vol. 4699. LNCS. Springer, pp. 1–10.

Caballer, Miguel, Carlos de Alfonso, Germán Moltó, Eloy Romero, Ignacio Blanquer, and Andrés García (2014). "CodeCloud: a platform to enable execution of programming models on the clouds". In: *Journal of Systems and Software* 93, pp. 187–198.

Campagne, Fabien (2014). *The MPS language workbench: volume I*. Fabien Campagne.

Carver, Jeffrey C. (2009). "First international workshop on software engineering for computational science & engineering". In: *Computing in Science & Engineering* 11.2, pp. 7–11.

Carver, Jeffrey C. and Tom Epperly (2014). "Software engineering for computational science and engineering". In: *Computing in Science & Engineering* 16.3, pp. 6–9.

Carver, Jeffrey C., Dustin Heaton, Lorin Hochstein, and Roscoe Bartlett (2013). "Self-perceptions about software engineering: a survey of scientists and engineers". In: *Computing in Science & Engineering* 15.1, pp. 7–11.

Carver, Jeffrey C., Lorin Hochstein, Richard P. Kendall, Taiga Nakamura, Marvin V. Zelkowitz, Victor R. Basili, et al. (2006). "Observations about software development for high end computing". In: *CTWatch Quarterly* 2.4A, pp. 33–38.

Carver, Jeffrey C., Richard P. Kendall, Susan E. Squires, and Douglass E. Post (2007). "Software development environments for scientific and engineering software: a series of case studies". In: *29th International Conference on Software Engineering (ICSE'07)*. IEEE, pp. 550–559.

Ceruzzi, P.E. (2003). *A history of modern computing*. 2nd ed. MIT Press.

Chavez, F.P., J. Ryan, S.E. Lluch-Cota, and M. Ñiquen (2003). "From anchovies to sardines and back: multidecadal change in the Pacific Ocean". In: *Science* 299.5604, pp. 217–221.

Christensen, Villy, Carl J. Walters, and Daniel Pauly (2005). *Ecopath with Ecosim: a user's guide*. Tech. rep. Fisheries Centre, University of British Columbia, Vancouver.

Clarke, Andrew and Nadine M. Johnston (1999). "Scaling of metabolic rate with body mass and temperature in teleost fish". In: *Journal of Animal Ecology* 68.5, pp. 893–905.

Collie, Jeremy, Cóilín Minto, Boris Worm, and Richard Bell (2013). "Predation on prerecruits can delay rebuilding of depleted cod stocks". In: *Bulletin of Marine Science* 89.1, pp. 107–122.

Consel, Charles and Renaud Marlet (1998). "Architecture software using a methodology for language development". In: *Principles of Declarative Programming*. Vol. 1490. LNCS. Springer, pp. 170–194.

Cury, Philippe, Yunne-Jai Shin, Benjamin Planque, Joël Marcel Durant, Jean-Marc Fromentin, Stephanie Kramer-Schadt, et al. (2008). "Ecosystem oceanography for global change in fisheries". In: *Trends in Ecology & Evolution* 23.6, pp. 338–346.

Czarnecki, Krzysztof, John T. O'Donnell, Jörg Striegnitz, and Walid Taha (2004). "DSL implementation in MetaOCaml, Template Haskell, and C++". In: *Domain-Specific Program Generation*. Vol. 3016. LNCS. Springer, pp. 51–72.

Dagum, Leonardo and Ramesh Menon (1998). "OpenMP: an industry standard API for shared-memory programming". In: *Computational Science & Engineering* 5.1, pp. 46–55.

DeAngelis, Donald L. and Louis J. Gross, eds. (1992). *Individual-based models and approaches in ecology: populations, communities and ecosystems*. Chapman & Hall.

DeAngelis, Donald L. and K. A. Rose (1992). "Which individual-based approach is most appropriate for a given problem?" In: *Individual-Based Models and Approaches in Ecology*. Ed. by Donald L. DeAngelis and Louis J. Gross. Chapman & Hall, pp. 67–87.

Dee, D.P., S.M. Uppala, A.J. Simmons, Paul Berrisford, P. Poli, S. Kobayashi, et al. (2011). "The ERA-Interim reanalysis: configuration and performance of the data assimilation system". In: *Quarterly Journal of the Royal Meteorological Society* 137.656, pp. 553–597.

Derrick, John and Eerke A. Boiten (2014). *Refinement in Z and Object-Z. Foundations and advanced applications*. 2nd ed. Springer.

DeVito, Z., N. Joubert, F. Palacios, S. Oakley, M. Medina, M. Barrientos, et al. (2011). "Liszt: a domain-specific language for building portable mesh-based PDE solvers". In: *Proceedings of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12.

Dongarra, Jack, Dennis Gannon, Geoffrey Fox, and Ken Kennedy (2007). "The impact of multicore on computational science software". In: *CT-Watch Quarterly* 3.1, pp. 1–10.

Dorussen, Han, Hartmut Lenz, and Spyros Blavoukos (2005). "Assessing the reliability and validity of expert interviews". In: *European Union Politics* 6.3, pp. 315–337.

Duncan, Ralph (1990). "A survey of parallel computer architectures". In: *Computer* 23.2, pp. 5–16.

Easterbrook, S.M. and T.C. Johns (2009). "Engineering the software for understanding climate change". In: *Computing in Science & Engineering* 11.6, pp. 65–74.

Efftinge, Sven, Moritz Eysholdt, Jan Köhnlein, Sebastian Zarnekow, Robert von Massow, Wilhelm Hasselbring, et al. (2012). "Xbase: implementing domain-specific languages for Java". In: *Proceedings of the 11th International Conference on Generative Programming and Component Engineering*, pp. 112–121.

Erdweg, Sebastian, Tijs van der Storm, Markus Völter, Meinte Boersma, Remi Bosman, William R. Cook, et al. (2013). "The state of the art in language workbenches". In: *Software Language Engineering*. Vol. 8225. LNCS. Springer, pp. 197–217.

Evans, Geoffrey T. and John S. Parslow (1985). "A model of annual plankton cycles". In: *Biological Oceanography* 3.3, pp. 327–347.

Ewald, Roland and Adelinde M. Uhrmacher (2014). "SESSL: a domain-specific language for simulation experiments". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 24.2, pp. 11:1–11:25.

Fasham, M.J.R., H.W. Ducklow, and S.M. McKelvie (1990). "A nitrogen-based model of plankton dynamics in the oceanic mixed layer". In: *Journal of Marine Research* 48.3, pp. 591–639.

Faulk, Stuart, Eugene Loh, Michael L. Van De Vanter, Susan Squires, and Lawrence G. Votta (2009). "Scientific computing's productivity grid-

lock: how software engineering can help". In: *Computing in Science & Engineering* 11.6, pp. 30–39.

Fennel, Katja, Damian Brady, Dominic DiToro, Robinson W. Fulweiler, Wayne S. Gardner, Anne Giblin, et al. (2009). "Modeling denitrification in aquatic sediments". In: *Biogeochemistry* 93.1-2, pp. 159–178.

Fennel, Katja, John Wilkin, Julia Levin, John Moisan, John O'Reilly, and Dale Haidvogel (2006). "Nitrogen cycling in the Middle Atlantic Bight: results from a three-dimensional model and implications for the North Atlantic nitrogen budget". In: *Global Biogeochemical Cycles* 20.3, pp. 1–14.

Fennel, Wolfgang (2008). "Towards bridging biogeochemical and fish-production models". In: *Journal of Marine Systems* 71.1, pp. 171–194.

— (2009). "Parameterizations of truncated food web models from the perspective of an end-to-end model approach". In: *Journal of Marine Systems* 76.1, pp. 171–185.

— (2010). "A nutrient to fish model for the example of the Baltic Sea". In: *Journal of Marine Systems* 81.1, pp. 184–195.

Fernö, A., T.J. Pitcher, W. Melle, L. Nøttestad, S. Mackinson, C. Hollingworth, et al. (1998). "The challenge of the herring in the Norwegian Sea: making optimal collective spatial decisions". In: *Sarsia* 83.2, pp. 149–167.

Fisher, Aaron, Robert N. Rieben, Garry H. Rodrigue, and Daniel White (2005). "A generalized mass lumping technique for vector finite-element solutions of the time-dependent Maxwell equations". In: *Antennas and Propagation* 53.9, pp. 2900–2910.

Flato, G., J. Marotzke, B. Abiodun, P. Braconnot, S.C. Chou, W. Collins, et al. (2013). "Evaluation of climate models". In: *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*. Cambridge University Press, pp. 741–866.

Fletcher, C.A.J. (1983). "The group finite element formulation". In: *Computer Methods in Applied Mechanics and Engineering* 37.2, pp. 225–244.

Fowler, Martin (2010). *Domain-specific languages*. Addison-Wesley.

Frank, Kenneth T., Brian Petrie, Jae S. Choi, and William C. Leggett (2005). "Trophic cascades in a formerly cod-dominated ecosystem". In: *Science* 308.5728, pp. 1621–1623.

Frank, Kenneth T., Brian Petrie, Jonathan A.D. Fisher, and William C. Leggett (2011). "Transient dynamics of an altered large marine ecosystem". In: *Nature* 477.7362, pp. 86–89.

Franks, Peter J.S. (2002). "NPZ models of plankton dynamics: their construction, coupling to physics, and application". In: *Journal of Oceanography* 58.2, pp. 379–387.

Franks, Peter J.S. and Changsheng Chen (2001). "A 3-D prognostic numerical model study of the Georges Bank ecosystem. Part II: biological–physical model". In: *Deep Sea Research Part II: Topical Studies in Oceanography* 48.1, pp. 457–482.

Froese, R. and D. Pauly (2015). *Fishbase*. URL: http://www.fishbase.org/.

Fuller, Samuel H. and Lynette I. Millett (2011). "Computing performance: game over or next level?" In: *Computer* 44.1, pp. 31–38.

Fulton, Elizabeth A. (2010). "Approaches to end-to-end ecosystem models". In: *Journal of Marine Systems* 81.1, pp. 171–183.

Fulton, Elizabeth A., Jason S. Link, Isaac C. Kaplan, Marie Savina-Rolland, Penelope Johnson, Cameron Ainsworth, et al. (2011). "Lessons in modelling and management of marine ecosystems: the Atlantis experience". In: *Fish and Fisheries* 12.2, pp. 171–188.

Fulton, Elizabeth A., John S. Parslow, Anthony D.M. Smith, and Craig R. Johnson (2004a). "Biogeochemical marine ecosystem models II: the effect of physiological detail on model performance". In: *Ecological Modelling* 173.4, pp. 371–406.

Fulton, Elizabeth A., Anthony D.M. Smith, and Craig R. Johnson (2004b). "Biogeochemical marine ecosystem models I: IGBEM – a model of marine bay ecosystems". In: *Ecological Modelling* 174.3, pp. 267–307.

Galante, Guilherme, Luis Carlos Erpen De Bona, Antonio Roberto Mury, and Bruno Schulze (2014). "Are public clouds elastic enough for scientific computing?" In: *Service-Oriented Computing – Proceedings ICSOC 2013 Workshops*. Springer, pp. 294–307.

Galesic, Mirta (2006). "Dropouts on the web: effects of interest and burden experienced during an online survey". In: *Journal of Official Statistics* 22.2, pp. 313–328.

Garcia, Javier Corral, César Gómez Martin, José Luis González Sánchez, and David Cortés Polo (2013). "Development of scientific applications with high-performance computing through a component-based and aspect-oriented methodology". In: *International Journal of Advanced Computer Science* 3.8, pp. 400–408.

Gašević, Dragan, Dragan Djurić, and Vladan Devedžić (2009). *Model driven engineering and ontology development*. 2nd ed. Springer.

Gessner, Jörn, Gerd-Michael Arndt, Ralph Tiedemann, Ryszard Bartel, and Frank Kirschbaum (2006). "Remediation measures for the Baltic sturgeon: status review and perspectives". In: *Journal of Applied Ichthyology* 22.Suppl. 1, pp. 23–31.

Gilliam, J.F. and D.F. Fraser (1987). "Habitat selection under predation hazard: test of a model with foraging minnows". In: *Ecology* 68.6, pp. 1856–1862.

Goerigk, Wolfgang, Reinhard von Hanxleden, Wilhelm Hasselbring, Gregor Hennings, Reiner Jung, Holger Neustock, et al. (2012). "Entwurf einer domänenspezifischen Sprache für elektronische Stellwerke". In: *Software Engineering 2012: Fachtagung des GI-Fachbereichs Softwaretechnik, 27. Februar – 2. März 2012 in Berlin*. Gesellschaft für Informatik e.V., pp. 119–130.

Gray, R., E.A. Fulton, L.R. Little, and R. Scott (2006). *Operating model specification within an agent based framework. North West Shelf joint environmental management study.* Tech. rep. CSIRO.

Grimm, Volker, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, et al. (2006). "A standard protocol for describing individual-based and agent-based models". In: *Ecological Modelling* 198.1, pp. 115–126.

Grimm, Volker and Steven F. Railsback (2005). *Individual-based modeling and ecology*. Princeton University Press.

Guderlei, Ralph and Johannes Mayer (2007). "Statistical metamorphic testing: testing programs with random output by means of statistical hypothesis tests and metamorphic testing". In: *Seventh International Conference on Quality Software (QSIC'07)*. IEEE, pp. 404–409.

Guennebaud, Gaël, Benoît Jacob, et al. (2010). *Eigen v3*. URL: `http://eigen.tuxfamily.org/`.

Guisan, Antoine and Niklaus E. Zimmermann (2000). "Predictive habitat distribution models in ecology". In: *Ecological Modelling* 135.2, pp. 147–186.

Guo, Song, Fan Bai, and Xiaolin Hu (2011). "Simulation software as a service and service-oriented simulation experiment". In: *International Conference on Information Reuse and Integration (IRI 2011)*. IEEE, pp. 113–116.

Hackbusch, Wolfgang (1985). *Multi-grid methods and applications*. Springer.

Hannay, J.E., H.P. Langtangen, C. MacLeod, D. Pfahl, J. Singer, and G. Wilson (2009). "How do scientists develop and use scientific software?"

In: *ICSE Workshop on Software Engineering for Computational Science and Engineering, 2009 (SECSE'09)*, pp. 1–8.

Hasselbring, Wilhelm (2015). "Formalization of federated schema architectural style variability". In: *Journal of Software Engineering and Applications* 8.2, pp. 72–92.

Hatton, Les and Andy Roberts (1994). "How accurate is scientific software?" In: *Software Engineering, IEEE Transactions on* 20.10, pp. 785–797.

Heaton, Dustin and Jeffrey C. Carver (2015). "Claims about the use of software engineering practices in science: a systematic literature review". In: *Information and Software Technology* 67, pp. 207–219.

Hebert, D. and R.G. Pettipas (2014). *Physical oceanographic conditions on the Scotian Shelf and in the eastern Gulf of Maine (NAFO areas 4V,W,X) during 2013*. Tech. rep. NAFO, pp. 1–24.

Hecht, Frédéric (2012). "New development in FreeFem++". In: *Journal of Numerical Mathematics* 20.3-4, pp. 251–266.

Hellweger, Ferdi L. and Vanni Bucci (2009). "A bunch of tiny individuals: individual-based modeling for microbes". In: *Ecological Modelling* 220.1, pp. 8–22.

Hempel, G. and J.H.S. Blaxter (1967). "Egg weight in Atlantic herring (Clupea harengus L.)" In: *Journal du Conseil* 31.2, pp. 170–195.

Hevner, Alan R., Richard C. Linger, Rosann Webb Collins, Mark G. Pleszkoch, and Gwendolyn H. Walton (2005). *The impact of function extraction technology on next-generation software engineering*. Tech. rep. Carnegie Mellon University.

Hilborn, Ray and Carl J. Walters (1987). "A general model for simulation of stock and fleet dynamics in spatially heterogeneous fisheries". In: *Canadian Journal of Fisheries and Aquatic Sciences* 44.7, pp. 1366–1369.

— (1992). *Quantitative fisheries stock assessment: choice, dynamics and uncertainty*. Chapman & Hall.

Hinsen, Konrad (2015). "The approximation tower in computational science: why testing scientific software is difficult". In: *Computing in Science & Engineering* 17.4, pp. 72–77.

Hochstein, Lorin, Jeffrey Carver, Forrest Shull, Sima Asgari, Victor Basili, Jeffrey K. Hollingsworth, et al. (2005). "Parallel programmer productivity: a case study of novice parallel programmers". In: *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*. IEEE, pp. 35–43.

Hoorn, André van, Sören Frey, Wolfgang Goerigk, Wilhelm Hasselbring, Holger Knoche, Sönke Köster, et al. (2011). "DynaMod project: dynamic analysis for model-driven software modernization". In: *Joint Proceedings of the 1st International Workshop on Model-Driven Software Migration (MDSM 2011) and the 5th International Workshop on Software Quality and Maintainability (SQM 2011)*, pp. 12–13.

Horsman, Tracy and Nancy Shackell (2009). *Atlas of important habitat for key fish species of the Scotian Shelf, Canada*. Tech. rep. Fisheries and Oceans Canada, Maritimes Region, Bedford Institute of Oceanography.

Houde, Edward D. (2009). "Recruitment variability". In: *Fish Reproductive Biology: Implications for Assessment and Management*. Ed. by T. Jakobsen, M.J. Fogarty, B.A. Megrey, and E. Moksness. Wiley, pp. 91–171.

Hove, Siw Elisabeth and Bente Anda (2005). "Experiences from conducting semi-structured interviews in empirical software engineering research". In: *11th IEEE International Software Metrics Symposium (METRICS 2005)*. IEEE, pp. 1–10.

Howison, James and James D. Herbsleb (2011). "Scientific software production: incentives and collaboration". In: *Proceedings of the Conference on Computer-Supported Cooperative Work 2011 (CSCW'11)*. ACM, pp. 513–522.

Huston, Michael, Donald DeAngelis, and Wilfred Post (1988). "New computer models unify ecological theory". In: *BioScience* 38.10, pp. 682–691.

Ince, Darrel C., Leslie Hatton, and John Graham-Cumming (2012). "The case for open computer programs". In: *Nature* 482.7386, pp. 485–488.

Iosup, Alexandru, Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick H.J. Epema (2011). "Performance analysis of cloud computing services for many-tasks scientific computing". In: *Parallel and Distributed Systems* 22.6, pp. 931–945.

ISO 14977 (1996). *Information technology—syntactic metalanguage—Extended BNF*. ISO 14977:1996. Geneva, Switzerland: International Organization for Standardization.

ISO 25010 (2011). *Systems and software engineering—systems and software quality requirements and evaluation (SQuaRE)—system and software quality models*. ISO 25010:2011. Geneva, Switzerland: International Organization for Standardization.

Jedlitschka, Andreas, Marcus Ciolkowski, and Dietmar Pfahl (2008). "Reporting experiments in software engineering". In: *Guide to Advanced*

Empirical Software Engineering. Ed. by Forrest Shull, Janice Singer, and Dag I.K. Sjøberg. Springer, pp. 201–228.

Jennings, Simon and Michel J. Kaiser (1998). "The effects of fishing on marine ecosystems". In: *Advances in Marine Biology* 34, pp. 201–352.

Jobling, M. (1981). "Mathematical models of gastric emptying and the estimation of daily rates of food consumption for fish". In: *Journal of Fish Biology* 19.3, pp. 245–257.

Johanson, Arne N. (2015a). *Benchmarks for the Sprat PDE DSL*. URL: http://dx.doi.org/10.5281/zenodo.19286.

— (2015b). *Data and analysis scripts for the application of the Sprat Marine Ecosystem Model to the eastern Scotian Shelf*. URL: http://dx.doi.org/10.5281/zenodo.34613.

— (2015c). *Data and scripts for the Sprat Ecosystem DSL survey*. URL: http://dx.doi.org/10.5281/zenodo.30557.

— (2015d). *Sprat – downloads – domain-speficic languages and the Sprat Model*. URL: http://www.sprat.uni-kiel.de/en/downloads.

Johanson, Arne N. and Wilhelm Hasselbring (2014a). "Hierarchical combination of internal and external domain-specific languages for scientific computing". In: *Proceedings of the 2014 European Conference on Software Architecture Workshops*. ECSAW'14. ACM, pp. 17:1–17:8.

— (2014b). "Sprat: hierarchies of domain-specific languages for marine ecosystem simulation engineering". In: *Proceedings TMS SpringSim'14*. SCS, pp. 187–192.

John, Volker and Ellen Schmeyer (2008). "Finite element methods for time-dependent convection-diffusion-reaction equations with small diffusion". In: *Computer Methods in Applied Mechanics and Engineering* 198.3, pp. 475–494.

Jones, B. and M.G. Kenward (2014). *Design and analysis of cross-over trials*. Taylor & Francis.

Joppa, Lucas N., Greg McInerny, Richard Harper, Lara Salido, Kenji Takeda, Kenton O'Hara, et al. (2013). "Troubling trends in scientific software use". In: *Science* 340.6134, pp. 814–815.

Juzna, Jernej, Peter Cesarek, Dana Petcu, and Vlado Stankovski (2014). "Solving solid and fluid mechanics problems in the cloud with moSAiC". In: *Computing in Science & Engineering* 16.3, pp. 68–77.

Kaiser, Robert (2014). *Qualitative Experteninterviews: Konzeptionelle Grundlagen und praktische Durchführung*. Springer.

Kanewala, Upulee and James M. Bieman (2013). "Using machine learning techniques to detect metamorphic relations for programs without test oracles". In: *IEEE 24th International Symposium on Software Reliability Engineering (ISSRE 2013)*, pp. 1–10.

— (2014). "Testing scientific software: a systematic literature review". In: *Information and Software Technology* 56.10, pp. 1219–1232.

Kelly, Diane F. (2007). "A software chasm: software engineering and scientific computing". In: *Software, IEEE* 24.6, pp. 120–119.

Kendall, Richard, Jeffrey C. Carver, David Fisher, Dale Henderson, Andrew Mark, Douglass Post, et al. (2008). "Development of a weather forecasting code: a case study". In: *Software, IEEE* 25.4, pp. 59–65.

Kieburtz, Richard B., Laura McKinney, Jeffrey M. Bell, James Hook, Alex Kotov, Jeffrey Lewis, et al. (1996). "A software engineering experiment in software component generation". In: *Proceedings of the 18th International Conference on Software Engineering (ICSE'96)*. IEEE, pp. 542–552.

Killcoyne, Sarah and John Boyle (2009). "Managing chaos: lessons learned developing software in the life sciences". In: *Computing in Science & Engineering* 11.6, pp. 20–29.

Kishi, Michio J., Makoto Kashiwai, Daniel M. Ware, Bernard A. Megrey, David L. Eslinger, Francisco E. Werner, et al. (2007). "NEMURO – a lower trophic level model for the North Pacific marine ecosystem". In: *Ecological Modelling* 202.1, pp. 12–25.

Kleppe, Anneke (2008). *Software language engineering: creating domain-specific languages using metamodels*. Addison-Wesley.

Knight, J. (2002). "Safety critical systems: challenges and directions". In: *Proceedings ICSE'02*. IEEE, pp. 547–550.

Kogge, Peter and John Shalf (2013). "Exascale computing trends: adjusting to the 'New Normal' in computer architecture". In: *Computing in Science & Engineering* 15.6, pp. 16–26.

Kolovos, Dimitrios S., Richard F. Paige, Tim Kelly, and Fiona A.C. Polack (2006). "Requirements for domain-specific languages". In: *Proceedings of ECOOP Workshop on Domain-Specific Program Development (DSPD)*, pp. 1–4.

Korman, Abraham K. (1971). *Industrial and organizational psychology*. Prentice-Hall.

Kosar, Tomaž, Marjan Mernik, and Jeffrey C. Carver (2012). "Program comprehension of domain-specific and general-purpose languages: com-

parison using a family of experiments". In: *Empirical Software Engineering* 17.3, pp. 276–304.

Koziolek, Heiko (2008). *Parameter dependencies for reusable performance specifications of software components*. PhD thesis. Universität Oldenburg.

Krishnamurthi, Shriram and Jan Vitek (2015). "The real software crisis: repeatability as a core value". In: *Communications of the ACM* 58.3, pp. 34–36.

Kuzmin, Dimitri, Rainald Löhner, and Stefan Turek (2012). *Flux-corrected transport: principles, algorithms, and applications*. 2nd ed. Springer.

Kuzmin, Dimitri, Matthias Möller, and Stefan Turek (2003). "Multidimensional FEM-FCT schemes for arbitrary time stepping". In: *International Journal for Numerical Methods in Fluids* 42.3, pp. 265–295.

Kuzmin, Dimitri and Stefan Turek (2002). "Flux correction tools for finite elements". In: *Journal of Computational Physics* 175.2, pp. 525–558.

Kuzmin, Dmitri (2010). *A guide to numerical methods for transport equations*. Tech. rep. Friedrich-Alexander-Universität Erlangen-Nürnberg. URL: http://www.mathematik.uni-dortmund.de/~kuzmin/Transport.pdf.

Lee, Jin-Luen, Rainer Bleck, and Alexander E. MacDonald (2010). "A multistep flux-corrected transport scheme". In: *Journal of Computational Physics* 229.24, pp. 9284–9298.

Lehodey, Patrick, Inna Senina, Beatriz Calmettes, John Hampton, and Simon Nicol (2013). "Modelling the impact of climate change on Pacific skipjack tuna population and fisheries". In: *Climatic Change* 119.1, pp. 95–109.

Lehodey, Patrick, Inna Senina, and Raghu Murtugudde (2008). "A spatial ecosystem and populations dynamics model (SEAPODYM) – modeling of tuna and tuna-like populations". In: *Progress in Oceanography* 78.4, pp. 304–318.

Lehodey, Patrick, Inna Senina, John Sibert, Laurent Bopp, B. Calmettes, J. Hampton, et al. (2010). "Preliminary forecasts of Pacific bigeye tuna population trends under the A2 IPCC scenario". In: *Progress in Oceanography* 86.1, pp. 302–315.

Leslie, Patrick H. (1945). "On the use of matrices in certain population mathematics". In: *Biometrika* 3, pp. 183–212.

LeVeque, Randall J., Ian M. Mitchell, and Victoria Stodden (2012). "Reproducible research for scientific computing: tools and strategies for changing the culture". In: *Computing in Science & Engineering* 14.4, pp. 13–17.

Likert, Rensis (1932). "A technique for the measurement of attitudes". In: *Archives of Psychology* 22.140, pp. 5–55.

Little, L.R., E.A. Fulton, R. Gray, D. Hayes, V. Lyne, R. Scott, et al. (2006). *Multiple use management strategy evaluation for the North West Shelf: results and discussion. North West Shelf joint environmental management study.* Tech. rep. CSIRO.

Logg, A., K.A. Mardal, and G. Wells (2012). *Automated solution of differential equations by the finite element method: the FEniCS book*. Vol. 84. LNCSE. Springer.

Löhner, Rainald, Ken Morgan, Jaime Peraire, and Mehdi Vahdati (1987). "Finite element flux-corrected transport (FEM-FCT) for the Euler and Navier-Stokes equations". In: *International Journal for Numerical Methods in Fluids* 7.10, pp. 1093–1109.

Löhner, Rainald, Ken Morgan, Mehdi Vahdati, Jay P. Boris, and David L. Book (1988). "FEM-FCT: combining unstructured grids with high resolution". In: *Communications in Applied Numerical Methods* 4.6, pp. 717–729.

Long, Kevin R. (2003). "Sundance rapid prototyping tool for parallel PDE optimization". In: *Large-Scale PDE-Constrained Optimization*. Vol. 30. LNCSE. Springer, pp. 331–341.

Madduri, Ravi, Alex Rodriguez, Thomas Uram, Katrin Heitmann, Tanu Malik, Saba Sehrish, et al. (2015). "PDACS: a portal for data analysis services for cosmological simulations". In: *Computing in Science & Engineering* 17.5, pp. 18–26.

Mahr, Bernd (2009). "Die Informatik und die Logik der Modelle". In: *Informatik-Spektrum* 32.3, pp. 228–249.

Maxwell, Joseph (1992). "Understanding and validity in qualitative research". In: *Harvard Educational Review* 62.3, pp. 279–301.

Mayring, Philipp (2015). *Qualitative Inhaltsanalyse: Grundlagen und Techniken*. 12th ed. Beltz.

McCann, Kevin Shear (2000). "The diversity–stability debate". In: *Nature* 405.6783, pp. 228–233.

McCann, Kevin, Alan Hastings, and Gary R. Huxel (1998). "Weak trophic interactions and the balance of nature". In: *Nature* 395.6704, pp. 794–798.

McDonald, A.D., L.R. Little, R. Gray, E. Fulton, K.J. Sainsbury, and V.D. Lyne (2008). "An agent-based modelling approach to evaluation of

multiple-use management strategies for coastal marine ecosystems". In: *Mathematics and Computers in Simulation* 78.2, pp. 401–411.

Megrey, Bernard A., Kenneth A. Rose, Shin-ichi Ito, Douglas E. Hay, Francisco E. Werner, Yasuhiro Yamanaka, et al. (2007a). "North Pacific basin-scale differences in lower and higher trophic level marine ecosystem responses to climate impacts using a nutrient-phytoplankton-zooplankton model coupled to a fish bioenergetics model". In: *Ecological Modelling* 202.1, pp. 196–210.

Megrey, Bernard A., Kenneth A. Rose, Robert A. Klumb, Douglas E. Hay, Francisco E. Werner, David L. Eslinger, et al. (2007b). "A bioenergetics-based population dynamics model of Pacific herring (Clupea harengus pallasi) coupled to a lower trophic level nutrient-phytoplankton-zooplankton model: description, calibration, and sensitivity analysis". In: *Ecological Modelling* 202.1, pp. 144–164.

Mell, Peter and Tim Grance (2011). *The NIST definition of cloud computing*. Tech. rep. National Institute of Standards and Technology.

Merali, Z. (2010). "Computational science: error, why scientific programming does not compute". In: *Nature* 467.7317, pp. 775–777.

Mernik, Marjan, Jan Heering, and Anthony M. Sloane (2005). "When and how to develop domain-specific languages". In: *ACM Computing Surveys (CSUR)* 37.4, pp. 316–344.

Message Passing Interface Forum (2012). *MPI: a message-passing interface standard. Version 3.0*. URL: http://www.mpi-forum.org/docs/.

Messina, Paul (2015). "Gaining the broad expertise needed for high-end computational science and engineering research". In: *Computing in Science & Engineering* 17.2, pp. 89–90.

Minto, Cóilín and Boris Worm (2012). "Interactions between small pelagic fish and young cod across the North Atlantic". In: *Ecology* 93.10, pp. 2139–2154.

Möller, Matthias, Dimitri Kuzmin, and Stefan Turek (2005). "Implicit finite element discretizations based on the flux-corrected transport algorithm". In: *International Journal for Numerical Methods in Fluids* 47.10-11, pp. 1197–1203.

Moloney, Coleen L., Michael A. St John, Kenneth L. Denman, David M. Karl, Friedrich W. Köster, Svein Sundby, et al. (2011). "Weaving marine food webs from end to end under global change". In: *Journal of Marine Systems* 84.3, pp. 106–116.

Morin, A., J. Urban, P.D. Adams, I. Foster, A. Sali, D. Baker, et al. (2012). "Shining light into black boxes". In: *Science* 336.6078, pp. 159–160.

Motika, Christian, Steven Smyth, and Reinhard von Hanxleden (2014). "Compiling SCCharts – a case-study on interactive model-based compilation". In: *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*. Springer, pp. 461–480.

Muranushi, Takayuki (2012). "Paraiso: an automated tuning framework for explicit solvers of partial differential equations". In: *Computational Science & Discovery* 5.1, pp. 1–40.

Myers, Ransom A. and Boris Worm (2003). "Rapid worldwide depletion of predatory fish communities". In: *Nature* 423.6937, pp. 280–283.

Naur, Peter and Brian Randell (1969). *Software Engineering: report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO*. NATO Scientific Affairs Division.

Neighbors, James M. (1984). "The Draco approach to constructing software from reusable components". In: *IEEE Transactions on Software Engineering* 10.5, pp. 564–574.

Neill, W.H. (1979). "Mechanisms of fish distribution in heterothermal environments". In: *American Zoologist* 19.1, pp. 305–317.

Neutel, Anje-Margriet, Johan A.P. Heesterbeek, and Peter C. de Ruiter (2002). "Stability in real food webs: weak links in long loops". In: *Science* 296.5570, pp. 1120–1123.

Newell, Perlis, and Simon (1967). "Letter to the editor". In: *Science* 157, pp. 1373–1374.

Niebler, Eric (2007). "Proto: a compiler construction toolkit for DSELs". In: *Proceedings of the 2007 Symposium on Library-Centric Software Design*. ACM, pp. 42–51.

Object Management Group (2014). *Object Constraint Language (OCL), version 2.4*. URL: http://www.omg.org/spec/OCL/.

— (2015a). *Meta Object Facility (MOF), version 2.5*. URL: http://www.omg.org/spec/MOF/.

— (2015b). *Model Driven Architecture (MDA)*. URL: http://www.omg.org/mda/.

— (2015c). *Unified Modeling Language (UML), version 2.5*. URL: http://www.omg.org/spec/UML/.

Oguz, Temel, Baris Salihoglu, and Bettina Fach (2008). "A coupled plankton-anchovy population dynamics model assessing nonlinear controls of

anchovy and gelatinous biomass in the Black Sea". In: *Marine Ecology Progress Series* 369, pp. 229–256.
Ohlberger, Jan, Georg Staaks, Peter L.M. van Dijk, and Franz Hölker (2005). "Modelling energetic costs of fish swimming". In: *Journal of Experimental Zoology Part A: Comparative Experimental Biology* 303.8, pp. 657–664.
Ouellet, Patrick, Yvan Lambert, and Isabelle Bérubé (2001). "Cod egg characteristics and viability in relation to low temperature and maternal nutritional condition". In: *ICES Journal of Marine Science* 58.3, pp. 672–686.
Palyart, Marc, David Lugato, Ileana Ober, and Jean-Michel Bruel (2012a). "MDE4HPC: an approach for using model-driven engineering in high-performance computing". In: *Proceedings SDL'11: Integrating System and Software Modeling*. Vol. 7083. LNCS, pp. 247–261.
Palyart, Marc, Ileana Ober, David Lugato, and Jean-Michel Bruel (2012b). "HPCML: a modeling language dedicated to high-performance scientific computing". In: *Proceedings of the 1st International Workshop on Model-Driven Engineering for High Performance and Cloud Computing*, pp. 1–6.
Peng, Roger D. (2011). "Reproducible research in computational science". In: *Science* 334.6060, pp. 1226–1227.
Perrone, L. Felipe, Christopher S. Main, and Bryan C. Ward (2012). "SAFE: simulation automation framework for experiments". In: *Proceedings of the 2012 Winter Simulation Conference*, pp. 1–12.
Pissanetzky, Sergio (2014). *Sparse matrix technology*. Elsevier Science.
Post, Douglass E. (2013). "The changing face of scientific and engineering computing". In: *Computing in Science & Engineering* 15.6, pp. 4–6.
Post, Douglass E. and Lawrence G. Votta (2005). "Computational science demands a new paradigm". In: *Physics Today* 58.1, pp. 35–41.
Prabhu, Prakash, Thomas B. Jablin, Arun Raman, Yun Zhang, Jialu Huang, Hanjun Kim, et al. (2011). "A survey of the practice of computational science". In: *State of the Practice Reports*. SC'11. ACM, pp. 19:1–19:12.
Prähofer, Herbert and Dominik Hurnaus (2010). "MONACO – a domain-specific language supporting hierarchical abstraction and verification of reactive control programs". In: *Proceedings of the 8th International Conference on Industrial Informatics (INDIN)*, pp. 908–914.
Preschern, Christopher, Andrea Leitner, and Christian Kreiner (2012). "Domain-specific language architecture for automation systems: an indus-

trial case study". In: *Proceedings of the 8th European Conference on Modelling Foundations and Applications (ECMFA)*, pp. 1–12.

Prud'homme, Christophe (2006). "A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations". In: *Scientific Programming* 14.2, pp. 81–110.

Prud'homme, Christophe (2007). "Life: overview of a unified C++ implementation of the finite and spectral element methods in 1D, 2D and 3D". In: *Applied Parallel Computing. State of the Art in Scientific Computing*. Vol. 4699. LNCS. Springer, pp. 712–721.

Quinn, Terrance J. and Richard B. Deriso (1999). *Quantitative fish dynamics*. Oxford University Press.

Radtke, Hagen, Thomas Neumann, and Wolfgang Fennel (2013). "A eulerian nutrient to fish model of the Baltic Sea – a feasibility-study". In: *Journal of Marine Systems* 125, pp. 61–76.

Railsback, Steven F. and Volker Grimm (2012). *Agent-based and individual-based modeling: a practical introduction*. Princeton University Press.

Railsback, Steven F., Roland H. Lamberson, Bret C. Harvey, and Walter E. Duffy (1999). "Movement rules for individual-based models of stream fish". In: *Ecological Modelling* 123.2, pp. 73–89.

Ramkrishna, Doraiswami (2000). *Population balances: theory and applications to particulate systems in engineering*. Academic Press.

Redfield, A. C., B. H. Ketchum, and F. A Richards (1966). "The influence of organisms on the composition of sea-water". In: *The Sea*. Ed. by M. N. Hill. Wiley, pp. 26–77.

Reed, R.K. (1977). "On estimating insolation over the ocean". In: *Journal of Physical Oceanography* 7.3, pp. 482–485.

Ricker, W.E. (1975). *Computation and interpretation of biological statistics of fish populations*. Vol. 191. Bulletin of the Fisheries Research Board of Canada. Department of the Environment, Fisheries and Marine Service.

Rieutord, Michel (2014). *Fluid dynamics: an introduction*. Springer.

Robson, John M. (1986). "Algorithms for maximum independent sets". In: *Journal of Algorithms* 7.3, pp. 425–440.

ROMS/TOMS Group (2015). *WikiROMS: documentation portal*. URL: http://www.myroms.org/wiki/.

Rose, Kenneth A., J. Icarus Allen, Yuri Artioli, Manuel Barange, Jerry Blackford, François Carlotti, et al. (2010). "End-to-end models for the analysis

of marine ecosystems: challenges, issues, and next steps". In: *Marine and Coastal Fisheries* 2.1, pp. 115–130.

Rose, Kenneth A., Francisco E. Werner, Bernard A. Megrey, Maki Noguchi Aita, Yasuhiro Yamanaka, Douglas E. Hay, et al. (2007). "Simulated herring growth responses in the Northeastern Pacific to historic temperature and zooplankton conditions generated by the 3-dimensional NEMURO nutrient-phytoplankton-zooplankton model". In: *Ecological Modelling* 202.1, pp. 184–195.

Royce, Winston W. (1970). "Managing the development of large software systems". In: *Proceedings of WESCON'70*. IEEE, pp. 328–338.

Runeson, Per, Martin Höst, Austen Rainer, and Björn Regnell (2012). *Case study research in software engineering: guidelines and examples*. Wiley.

Sanders, Rebecca and Diane F. Kelly (2008). "Dealing with risk in scientific software development". In: *Software, IEEE* 25.4, pp. 21–28.

Sanderson, Conrad (2010). *Armadillo: an open source C++ linear algebra library for fast prototyping and computationally intensive experiments*. Tech. rep. NICTA.

Scheffer, M., J.M. Baveco, D.L. DeAngelis, K.A. Rose, and E.H. van Nes (1995). "Super-individuals a simple solution for modelling large populations on an individual basis". In: *Ecological Modelling* 80.2, pp. 161–170.

Schieferdecker, Ina (2012). "Model-based testing". In: *IEEE Software* 1, pp. 14–18.

Schnetter, Erik, Marek Blazewicz, Steven R. Brandt, David M. Koppelman, and Frank Löffler (2015). "Chemora: a PDE-solving framework for modern high-performance computing architectures". In: *Computing in Science & Engineering* 17.2, pp. 53–64.

Schnute, Jon (1987). "A general fishery model for a size-structured fish population". In: *Canadian Journal of Fisheries and Aquatic Sciences* 44.5, pp. 924–940.

Segal, Judith (2005). "When software engineers met research scientists: a case study". In: *Empirical Software Engineering* 10.4, pp. 517–536.

— (2007). "Some problems of professional end user developers". In: *Symposium on Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007*. IEEE, pp. 111–118.

— (2008). "Models of scientific software development". In: *Proceedings of the 1st International Workshop on Software Engineering for Computational Science and Engineering (SECSE'08)*, pp. 1–7.
— (2009). "Some challenges facing software engineers developing software for scientists". In: *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE, pp. 9–14.
Segal, Judith and Chris Morris (2008). "Developing scientific software". In: *Software, IEEE* 25.4, pp. 18–20.
Sempolinski, Peter, Douglas Thain, Zhigang Wei, and Ahsan Kareem (2015). "Adapting collaborative software development techniques to structural engineering". In: *Computing in Science & Engineering* 17.5, pp. 27–34.
Shapiro, Samuel S., Martin B. Wilk, and Hwei J. Chen (1968). "A comparative study of various tests for normality". In: *Journal of the American Statistical Association* 63.324, pp. 1343–1372.
Shchepetkin, Alexander F. and James C. McWilliams (2005). "The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model". In: *Ocean Modelling* 9.4, pp. 347–404.
Shin, Yunne-Jai and Philippe Cury (2001). "Exploring fish community dynamics through size-dependent trophic interactions using a spatialized individual-based model". In: *Aquatic Living Resources* 14.2, pp. 65–80.
— (2004). "Using an individual-based model of fish assemblages to study the response of size spectra to changes in fishing". In: *Canadian Journal of Fisheries and Aquatic Sciences* 61.3, pp. 414–431.
Shin, Yunne-Jai, L.J. Shannon, and Philippe Cury (2004). "Simulations of fishing effects on the southern Benguela fish community using an individual-based model: learning from a comparison with Ecosim". In: *African Journal of Marine Science* 26.1, pp. 95–114.
Shin, Yunne-Jai, Morgane Travers, and Olivier Maury (2010). "Coupling low and high trophic levels models: towards a pathways-orientated approach for end-to-end models". In: *Progress in Oceanography* 84.1, pp. 105–112.
Shull, Forrest, Jeffrey Carver, Lorin Hochstein, and Victor Basili (2005). "Empirical study design in the area of high-performance computing (HPC)". In: *Proceedings of the International Symposium on Empirical Software Engineering 2005*. IEEE, pp. 1–10.
Sibert, John R., John Hampton, David A. Fournier, and Peter J. Bills (1999). "An advection-diffusion-reaction model for the estimation of fish move-

ment parameters from tagging data, with application to skipjack tuna (Katsuwonus pelamis)". In: *Canadian Journal of Fisheries and Aquatic Sciences* 56.6, pp. 925–938.

Simenstad, Charles A. and Gregor M. Cailliet (1986). *Contemporary studies on fish feeding*. Kluwer.

Sinclair, M. and T.D. Iles (1985). "Atlantic herring (Clupea harengus) distributions in the Gulf of Maine – Scotian Shelf area in relation to oceanographic features". In: *Canadian Journal of Fisheries and Aquatic Sciences* 42.5, pp. 880–887.

Smith, Graeme (2000). *The Object-Z specification language*. Kluwer.

Solingen, R. van and E. Berghout (1999). *The Goal/Question/Metric method: a practical guide for quality improvement of software development*. McGraw-Hill.

Song, Yuhe and Dale Haidvogel (1994). "A semi-implicit ocean circulation model using a generalized topography-following coordinate system". In: *Journal of Computational Physics* 115.1, pp. 228–244.

Spinellis, Diomidis (2001). "Notable design patterns for domain-specific languages". In: *Journal of Systems and Software* 56.1, pp. 91–99.

Spivey, J. Michael (1992). *The Z notation: a reference manual*. Prentice-Hall.

Stachowiak, Herbert (1973). *Allgemeine Modelltheorie*. Springer.

Stahl, Thomas and Markus Völter (2006). *Model-driven software development: technology, engineering, management*. Wiley.

Steinberg, D., F. Budinsky, M. Paternostro, and E. Merks (2008). *EMF: Eclipse modeling framework*. 2nd ed. Addison-Wesley.

Stock, Charles A., Michael A. Alexander, Nicholas A. Bond, Keith M. Brander, William W.L. Cheung, Enrique N. Curchitser, et al. (2011). "On the use of IPCC-class models to assess the impact of climate on living marine resources". In: *Progress in Oceanography* 88.1, pp. 1–27.

Stramma, Lothar, Eric D. Prince, Sunke Schmidtko, Jiangang Luo, John P. Hoolihan, Martin Visbeck, et al. (2012). "Expansion of oxygen minimum zones may reduce available habitat for tropical pelagic fishes". In: *Nature Climate Change* 2.1, pp. 33–37.

Strembeck, Mark and Uwe Zdun (2009). "An approach for the systematic development of domain-specific languages". In: *Software: Practice and Experience* 39.15, pp. 1253–1292.

Student (1908). "The probable error of a mean". In: *Biometrika* 6.1, pp. 1–25.

Swain, Douglas P. and Ghislain A. Chouinard (2008). "Predicted extirpation of the dominant demersal fish in a large marine ecosystem: Atlantic cod (Gadus morhua) in the southern Gulf of St. Lawrence". In: *Canadian Journal of Fisheries and Aquatic Sciences* 65.11, pp. 2315–2319.

Thompson, R.W. and D.A. Cauley (1979). "A population balance model for fish population dynamics". In: *Journal of Theoretical Biology* 81.2, pp. 289–307.

Travers, Morgane, Yunne-Jai Shin, Simon Jennings, and Philippe Cury (2007). "Towards end-to-end models for investigating the effects of climate and fishing in marine ecosystems". In: *Progress in Oceanography* 75.4, pp. 751–770.

Urrego-Blanco, Jorge and Jinyu Sheng (2012). "Interannual variability of the circulation over the eastern Canadian shelf". In: *Atmosphere-Ocean* 50.3, pp. 277–300.

Utne, Kjell Rong, Solfrid Sætre Hjøllo, Geir Huse, and Morten Skogen (2012). "Estimating the consumption of Calanus finmarchicus by planktivorous fish in the Norwegian Sea using a fully coupled 3D model system". In: *Marine Biology Research* 8.5-6, pp. 527–547.

Utne, Kjell Rong and Geir Huse (2012). "Estimating the horizontal and temporal overlap of pelagic fish distribution in the Norwegian Sea using individual-based modelling". In: *Marine Biology Research* 8.5-6, pp. 548–567.

Varga, András (2001). "The OMNeT++ discrete event simulation system". In: *Proceedings of the European Simulation Multiconference (ESM'2001)*, pp. 1–7.

Veldhuizen, Todd L. (2000). "Blitz++: the library that thinks it is a compiler". In: *Advances in Software Tools for Scientific Computing*. Springer, pp. 57–87.

Vessey, Iris (1997). "Problems versus solutions: the role of the application domain in software". In: *Papers Presented at the 7th Workshop on Empirical Studies of Programmers*. ACM, pp. 233–240.

Videler, J.J. (1993). *Fish swimming*. Vol. 10. Fish and Fisheries Series. Chapman & Hall.

Videler, J.J. and C.S. Wardle (1991). "Fish swimming stride by stride: speed limits and endurance". In: *Reviews in Fish Biology and Fisheries* 1.1, pp. 23–40.

Völter, Markus (2013). *DSL engineering: designing, implementing and using domain-specific languages*. CreateSpace.

Ware, D.M. (1978). "Bioenergetics of pelagic fish: theoretical change in swimming speed and ration with body size". In: *Journal of the Fisheries Board of Canada* 35.2, pp. 220–228.

Watson, Reg, Dirk Zeller, and Daniel Pauly (2014). "Primary productivity demands of global fishing fleets". In: *Fish and Fisheries* 15.2, pp. 231–241.

Webb Collins, Rosann, Alan R. Hevner, Gwendolyn H. Walton, and Richard C. Linger (2008). "The impacts of function extraction technology on program comprehension: a controlled experiment". In: *Information and Software Technology* 50.11, pp. 1165–1179.

Wilcoxon, Frank (1945). "Individual comparisons by ranking methods". In: *Biometrics Bulletin* 1.6, pp. 80–83.

Wilensky, U. (1999). *NetLogo*. URL: http://ccl.northwestern.edu/netlogo/.

Wilson, Gregory V. (2006a). "Software carpentry". In: *Computing in Science & Engineering* 8, pp. 66–69.

— (2006b). "Where's the real bottleneck in scientific computing?" In: *American Scientist* 94.1, pp. 5–6.

— (2014). "Software carpentry: lessons learned". In: *F1000Research* 3, pp. 1–11.

Wohlin, Claes, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén (2012). *Experimentation in software engineering*. Springer.

Worm, Boris, Ray Hilborn, Julia K. Baum, Trevor A. Branch, Jeremy Collie, Christopher Costello, et al. (2009). "Rebuilding global fisheries". In: *Science* 325.5940, pp. 578–585.

Worm, Boris and Ransom A. Myers (2003). "Meta-analysis of cod-shrimp interactions reveals top-down control in oceanic food webs". In: *Ecology* 84.1, pp. 162–173.

Yodzis, Peter (2001). "Must top predators be culled for the sake of fisheries?" In: *Trends in Ecology & Evolution* 16.2, pp. 78–84.

Zalesak, Steven T. (1979). "Fully multidimensional flux-corrected transport algorithms for fluids". In: *Journal of Computational Physics* 31.3, pp. 335–362.

Zeigler, B.P., H. Praehofer, and T.G. Kim (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. 2nd ed. Academic Press.

Zwanenburg, K.C.T., D. Bowen, A. Bundy, K. Frank, K. Drinkwater, R. O'Boyle, et al. (2002). "Decadal changes in the Scotian Shelf large marine ecosystem". In: *Large Marine Ecosystems* 10, pp. 105–150.

This page is intentionally left blank

This page is intentionally left blank

This page is intentionally left blank