

FERNANDO MENDES COIMBRA DE MENDONÇA

**A PYTHON BASED SYSTEM TO MANAGE SIMULATIONS
OF COASTAL OPERATIONAL MODELS**



UNIVERSIDADE DO ALGARVE
INSTITUTO SUPERIOR DE ENGENHARIA

2020

FERNANDO MENDES COIMBRA DE MENDONÇA

**A PYTHON BASED SYSTEM TO MANAGE SIMULATIONS
OF COASTAL OPERATIONAL MODELS**

**Mestrado em Engenharia Mecânica
(Especialidade em Energia, Climatização e Refrigeração)**

Trabalho efetuado sob a orientação de:

Prof. Dr. Flávio Martins

Dr. João Janeiro



**UNIVERSIDADE DO ALGARVE
INSTITUTO SUPERIOR DE ENGENHARIA**

2020

A python-based system to manage simulations of coastal operational models

Declaração de autoria de trabalho

Declaro ser o autor deste trabalho, que é original e inédito. Autores e trabalhos consultados estão devidamente citados no texto e constam da listagem de referências incluída.

Copyright ©Fernando Mendes Coimbra de Mendonça

A Universidade do Algarve reserva para si o direito, em conformidade com o disposto no Código do Direito de Autor e dos Direitos Conexos, de arquivar, reproduzir e publicar a obra, independentemente do meio utilizado, bem como de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição para fins meramente educacionais ou de investigação e não comerciais, conquanto seja dado o devido crédito ao autor e editor respetivos.

ABSTRACT

Operational oceanography is understood as the constant endeavor of observing seas and oceans, collecting, interpreting and disseminating the measured data, in order to design methods for analyzing behavior and predicting future conditions. Consequently, ocean modelling is one of the most important activities developed within this context, as it helps to understand the aspects and phenomena of those ecosystems. In this scenario, the goal of this work is to build a simulation management system, programmed in the Python language, for coastal hydrodynamic models made in MOHID modelling system environment. MOHID is a three-dimensional model with numerical tools to solve governing equations that describe the fluid flow and to reproduce several processes of the marine environment. Python allows the rapid creation of sophisticated algorithms for all types of tasks, having the necessary resources to perform the most essential function in simulation management: the control of the inputs and outputs of a model. The management system used SOMA as the application example. SOMA is the operational validated high resolution hydrodynamic model of the Algarve coast based on MOHID. In this setup it is necessary to manage two distinct simulation cycles: the daily runs or forecast simulations, and weekly runs, which initialize the model for a new week cycle. The main body of the program was divided into two basic layers, one to process the corresponding forcing data and other to manage the simulations. The algorithm was designed to work with a pre-determined generic structure of folders and file nomenclature. MOHID's keyword feature was also adopted to specify the parameters to configure the tool. SOMA has become the first model to be controlled by the tool and still keeps its forecast cycles active, nevertheless, due to its generic feature, the simulation management system presented in this work is prepared to manage any other MOHID based model in operational mode.

Keywords: operational oceanography, MOHID modelling system, Python, simulation management system, SOMA, Algarve coast.

RESUMO

Os oceanos desempenham um papel de extrema importância na manutenção da vida no planeta. Eles são responsáveis por regular o clima e temperatura globais, reciclar nutrientes, gerar oxigênio, absorver dióxido de carbono da atmosfera entre outros. Para a humanidade, este ambiente possui ainda uma relevância extremamente elevada em relação a aspectos econômicos, servindo principalmente como fonte de alimentos e de exploração de recursos fósseis, via de transporte, desenvolvimento de atividades turísticas e recreativas, geração de energia. Por esses motivos, torna-se essencial compreender suas características e os diversos fenômenos que nele se desenvolvem. Dessa maneira, a oceanografia operacional é entendida como a atividade constante de observação, coleta, interpretação e divulgação dos dados medidos, a fim de projetar métodos de análise de comportamento e previsão de condições futuras.

A modelação oceânica é uma das atividades mais importantes desenvolvidas no âmbito da oceanografia operacional. Um modelo consiste na representação matemática de fenômenos, que neste caso são equações para descrever a dinâmica dos fluidos nos oceanos. O conjunto de expressões matemáticas que representam o movimento dos fluidos forma um sistema de equações diferenciais, que são construídas respeitando os princípios fundamentais de conservação da física, dando origem às equações de transporte. Uma vez que essas equações não têm solução analítica, elas têm de ser resolvidas usando métodos de modelação matemática. Para isso as equações são discretizadas usando métodos numéricos e transformadas em código computacional para avançar ou iterar o estado do oceano no tempo.

As equações de transporte representam o movimento do fluido e descrevem, cada uma, a conservação de uma propriedade. Elas são, portanto, formadas pelas equações de Navier-Stokes, que correspondem à conservação de massa e momentum, mais o balanço de energia e as equações de estado, que correlacionam propriedades termodinâmicas e fazem o vínculo entre temperatura e salinidade com a densidade. Em modelos oceânicos, elas são frequentemente discretizadas no espaço pelo método de volumes finitos na abordagem euleriana, que consiste na avaliação da conservação de propriedades dentro de um volume de controle, enquanto um fluxo de água passa por ele. O sistema de equações é resolvido

individualmente para cada um destes elementos que juntos constituem a malha do domínio do modelo, de forma que cada iteração expressa a conservação em cada um deles.

Embora já seja um negócio muito valioso, estima-se que a economia oceânica ainda possa dobrar nos próximos anos, à medida que a população cresce nos centros urbanos próximos às áreas costeiras. Assim, crescimento azul é o termo criado para designar práticas que visam o crescimento econômico, mas que também garantem a preservação desse ambiente para as gerações futuras. A busca pelo desenvolvimento sustentável provoca uma demanda crescente por conhecimento em relação aos oceanos, especialmente em regiões costeiras, fazendo com que os dados gerados na oceanografia operacional sejam essenciais para as atividades socioeconômicas do crescimento azul. Sendo assim, este trabalho tem como objetivo desenvolver um sistema de gestão de simulações, programado na linguagem Python, para modelos hidrodinâmicos costeiros que sejam construídos no ambiente do sistema de modelação MOHID.

O sistema MOHID é uma ferramenta numérica capaz de reproduzir diversos processos do ambiente marinho. Ele é um modelo hidrodinâmico tridimensional que soluciona os sistemas de equações de transporte para escoamentos incompressíveis, em que o equilíbrio hidrostático e a aproximação de Boussinesq são assumidos. A discretização espacial vertical é realizada por meio de coordenadas genéricas, em que a coluna de água pode ser dividida em vários subdomínios para melhor se adequar as regiões heterogêneas de todo o domínio. A discretização temporal é processada por um algoritmo ADI semi-implícito com dois níveis de tempo por iteração.

O código do MOHID está escrito em ANSI FORTRAN 95, que permite a programação por meio do paradigma da orientação por objetos. Logo, o sistema possui uma arquitetura modular, de forma que cada módulo representa uma classe responsável por gerir um tipo específico de informação. Este “design” possibilita a simulação simultânea de modelos aninhados e, conseqüentemente, o uso da metodologia de redução de escala, sendo cada domínio uma nova instância de classes individuais. Alguns dos módulos requerem informações extras para serem executados corretamente. Esses parâmetros são fornecidos através de um sistema de palavras-chave escrito em formato de texto ASCII em um arquivo específico para cada um. O conjunto desses arquivos de entradas caracteriza um modelo no ambiente MOHID.

O Python é uma linguagem de programação multiplataforma de código aberto e licença gratuita que se tem tornado cada vez mais popular. Ela é construída usando uma abordagem orientada por objetos, baseado em classes, possuindo uma vasta biblioteca de módulos versáteis, nativos e externos, que permitem a criação rápida de algoritmos sofisticados para todos os tipos de tarefas. Devido às suas características o Python também tem sido amplamente adotado no domínio científico. Em modelação oceânica ele possui ferramentas para lidar com formatos de dados mais comuns, suporte para OPeNDAP, métodos para realizar análise harmônica de marés e de visualização científica. A linguagem também possui os recursos necessários para realizar a tarefa mais essencial durante os ciclos de previsão, que é o controle das entradas e saídas de um modelo.

O sistema de gestão foi desenvolvido utilizando como base a operacionalização do Sistema de Modelação e Monitorização Operacional do Algarve, ou SOMA, modelo hidrodinâmico validado de alta resolução da costa algarvia que utiliza como base o sistema de modelação MOHID. Desta forma, a ferramenta faz a gestão de dois ciclos de simulação distintos: as corridas diárias, que equivalem às simulações de previsões contínuas, e as semanais, que inicializam o modelo para gerar novas condições iniciais menos deterioradas. Além disso, foi preciso desenvolver também operações para processar os dados de forçamento correspondentes. Por esse motivo, o corpo principal do programa foi dividido em duas camadas, uma para os dados de fontes externas e outra para gerir simulações.

A gestão de simulações consiste basicamente em coordenar a execução de diferentes operações e o manuseamento de arquivos. Por esse motivo, o algoritmo do programa foi concebido de forma a trabalhar em cima de uma estrutura genérica pré-determinada de pastas e de nomenclatura de arquivos. Esta arquitetura foi sendo aprimorada à medida que os padrões de uma rotina de simulação realizada pelo MOHID eram identificados. Além disso, foi adotado o método de leitura de palavras-chave para a especificação dos parâmetros para executar cada modelo. Depois de adaptado a estas normas, o SOMA passou a ser o meio para testar as versões da ferramenta. Consequentemente, este modelo é primeiro a ser controlado por ela, razão pela qual foram criados módulos para processar dados oceânicos do Mercador e atmosféricos do Skiron.

O uso da linguagem de programação orientada por objetos contribuiu para simplificar o código do programa, uma vez que foram construídos métodos para serem usados mais de uma vez em diferentes situações, evitando assim a escrita repetitiva de partes iguais do código. Além disso, módulos que desenvolvem tarefas fora do contexto de simulações, como formatação de resultados ou processamento de dados de forçamentos, deram origem a outras funcionalidades já que poderiam ser acessados de forma independente. Porém, da forma como foi projetado e em conjunto com a estrutura de pastas imposta, a grande limitação do programa de gestão é de ainda não permitir a operacionalização de modelos que tenham mais do que um subdomínio definido para o mesmo nível. Além disso, quando o código divide a execução do programa em para realizar tarefas simultâneas, as mensagens impressas de cada módulo em execução surgem simultâneas na janela de prompt, tornando-se confusas para o usuário. Estas são algumas das melhorias que se irão efetuar no futuro próximo.

Apesar das limitações apresentadas o sistema de gestão de simulação apresentado neste trabalho está preparado para converter modelos no modo operacional. Ainda assim, sua construção é um processo contínuo, logo, ele continuará a passar por atualizações e aperfeiçoamentos. Neste sentido, as opções para executar operações independentes, como formatação de resultados e o processamento de dados de forçamento, já estão sendo programadas e em breve serão integradas ao programa. Considerando ainda que Python possui uma vasta biblioteca embutida e diversas outras ferramentas externas, que oferecem uma ampla gama de recursos, também como trabalhos futuros, ir-se-á otimizar a paralelização de tarefas e até mesmo construção de uma interface gráfica para melhorar a visualização das mensagens. Além disso, o programa tem espaço para crescer χ ομ novos módulos para outras fontes de forçamento e até com o desenvolvimento de novas funcionalidades.

Palavras-chave: oceanografia operacional, sistema de modelação MOHID, Python, sistema de gestão de simulações, SOMA, costa algarvia.

CONTENTS

1 INTRODUCTION.....	1
2 STATE OF THE ART.....	3
2.1 Operational Oceanography.....	3
2.1.1 Observation Methods.....	4
2.1.2 Observation Network.....	12
2.1.3 Operational Oceanography and Blue Growth.....	13
2.2 Numerical Ocean Modelling.....	14
2.2.1 MOHID Modelling System.....	17
2.3 Python Programming Language.....	21
3 METHODOLOGY.....	25
4 RESULTS.....	32
4.1 Main data structure.....	32
4.2 User inputs.....	34
4.3 Simulation operation.....	37
4.3.1 Daily run cycle.....	39
4.3.2 Weekly run cycle.....	41
4.3.3 Forcing layer.....	44
4.3.4 Formatting outputs.....	47
4.4 Failures statistics.....	49
4.5 SOMA Outputs.....	51
5 DISCUSSION.....	55
6 CONCLUSION.....	60

BIBLIOGRAPHY.....	62
APPENDICES.....	69
Appendix A – Operation selector.....	69
Appendix B – Initial data file reader.....	70
Appendix C – Common operations.....	72
Appendix D – Simulation control.....	73
Appendix E – Weekly run cycle control.....	76
Appendix F – Daily run cycle control.....	78
Appendix G – Simulation class.....	80
Appendix H – Forcing data selector.....	84
Appendix I – MOHID outputs database formatting.....	86
Appendix J – SOMA outputs OCASO formatting.....	89
Appendix K – OCASO formatting supporting class.....	91
Appendix L – Mercator module.....	96
Appendix M – Skiron module.....	101

INDEX OF FIGURES

Figure 2.1 - Navis BGC Argo float. From: (Roemmich et al., 2019).....	6
Figure 2.2 - Surface drifter common structure. Adapted from: (Wu et al., 2019).....	7
Figure 2.3 - Diagram of coastal moorings. From: (Bailey et al., 2019).....	8
Figure 2.4 - Oceanscan light autonomous underwater vehicle used by the research center CIMA at University of Algarve.....	9
Figure 2.5 - HF radar basic operating principle. From: (Medclic, 2020).....	9
Figure 2.6 - Positions of active Argo floats in May of 2020. From: (Argo Program, 2020)	10
Figure 2.7 - Spacial and temporal resolution at the sea surface of <i>in-situ</i> and satellites observation techniques. From: (Send, 2006).....	12
Figure 2.8 - Vertical grid division into four different sub-domains. From: (F. Martins et al., 2001).....	18
Figure 2.9 - MOHID generic finite volume element geometry. From: (F. Martins et al., 2001).....	19
Figure 3.1 - SOMA levels and location of in-situ observation systems. From: (Janeiro et al., 2017).....	26
Figure 3.2 - SOMA current bathymetry.....	27
Figure 3.3 - Simulation cycles diagram.....	28
Figure 3.4 - SOMA's built in MOHID GUI.....	29
Figure 3.5 - Hierarchical structure instructions for three nested models.....	29
Figure 3.6 - MOHID modules instructions for the father domain of SOMA system.....	30
Figure 3.7 - Simulation time instructions.....	30
Figure 3.8 - Simulation cycles essential operations sequence.....	31
Figure 4.1 - Scheme of a generic project structure.....	34

Figure 4.2 - SOMA init.dat set up.....	37
Figure 4.3 - Main processes of the simulation management.....	38
Figure 4.4 - Simulation operation algorithm logic.....	39
Figure 4.5 - Daily run cycle operations part 1.....	40
Figure 4.6 - Daily run cycle operations part 2.....	41
Figure 4.7 - Weekly run cycle operations part 1.....	42
Figure 4.8 - Weekly run cycle operations part 2.....	43
Figure 4.9 - Weekly run cycle operations part 3.....	43
Figure 4.10 - Forcing layer first stage.....	44
Figure 4.11 - Forcing layer second stage.....	45
Figure 4.12 - Simulation output formatting module operations for model database.....	47
Figure 4.13 - Simulation output formatting module operations for OCASO project.....	48
Figure 4.14 - SOMA surface velocity forecast in July 18 th	52
Figure 4.15 - SOMA surface temperature forecast in July 18 th	52
Figure 4.16 - SOMA 100 meters depth temperature forecast in July 18 th	53
Figure 4.17 - SOMA surface velocity forecast in June 19 th	53
Figure 4.18 - SOMA surface temperature forecast in June 19 th	54
Figure 4.19 - SOMA 100 meters depth temperature forecast in June 19 th	54

INDEX OF TABLES

Table 3.1 - Essential Python modules in simulation manager.....	31
Table 4.1 - Mandatory keywords.....	35
Table 4.2 - Optional keywords.....	35
Table 4.3 - Dependent mandatory keywords.....	36
Table 4.4 - Description of the shapes adopted.....	38
Table 4.5 - Simulation layer log codes.....	41
Table 4.6 - Failures in simulation layer.....	49
Table 4.7 - Simulation layer failures proportion.....	49
Table 4.8 - Failures in forcing layer.....	50

1 INTRODUCTION

Mankind still has a lot to learn about the oceans, however it is already well known that they are of extremely importance in the Earth's ecosystem. It is an environment that plays a key role in the planet's climate, being a determining factor for regional climates affecting the distribution of rainfall, droughts, floods. Also, since ancient times, the seas are of great relevance for humanity being used as a source of food, recreation, transportation and for extraction of valuable fossil resources.

Due to their importance, oceans various physical, chemical and biological phenomena are constantly investigated and studied through continuous observation, in order to produce helpful tools for their exploitation. In this context, the area of operational oceanography includes making, disseminating and interpreting measurements of the seas and oceans properties that will compose a set of historical data of this environment. These can be further applied in numerical models that will help to assess and to understand their states so that it would be possible to make predictions of their future state.

Operational oceanography is always facing improvement challenges, especially those related to shallow waters. The hydrodynamic processes of these sites are complex due to rapid changes induced by the most dynamic variations caused by winds, waves, tides, sediment transport, human activity. In view of this difficulty, the Algarve Operational Modelling and Monitoring System (SOMA) has emerged as a high-resolution operational model, aiming to provide future conditions of the sea state in the coastal region of south Portugal.

The objective of this work is to develop a coastal simulation management system and use SOMA as the means to test it, since the last is a validated model and is enabled to predict Algarve's coast hydrodynamic behavior and its water properties. The system created in this work is here named as SMS-Coastal. It is built for the Windows operating system and programmed using the open-source, high-level and object-oriented language Python, in order to manage daily simulations of the operational model. Additionally, SOMA's numerical calculation is made by the hydrodynamic model MOHID, therefore it should be a pillar of the SMS-Coastal main structure. Furthermore, this work had been developed within the scope of the Coastal Environmental Observatory of the Southwest (OCASO)

project. Thus, once completed, the tool should be able to manage the Algarve's coastal model forecast data, making it available in a specific format as a product under the project for state alerts.

SMS-Coastal should also be able to handle any other coastal model with equivalent data structure as SOMA and that uses MOHID system for the numerical solution of the hydrodynamic equations. For that reason, its structure shall be generic enough to receive user input, such as simulation date, number of levels, forecast days range, simulation step time and others, download and convert corresponding external forcing data files, manage the simulation itself, generate monitoring reports and make result files available to be interpreted by a graphic interface. This universal architecture shall make forecast processes possible and fully automatic, keeping the models running, checking the availability of initial and boundary conditions and attesting the success of the simulations.

Besides this introduction, this document was divided into five more chapters. The second chapter makes a state of the art review of subjects related to this work, starting with the definition of operational oceanography. Then, it briefly presents the concept of ocean modelling and also describes the hydrodynamic model MOHID, the numerical tool used to simulate the models managed by SMS-Coastal. Finally, the basic concepts of Python are presented, the programming language used to write the algorithm of the application of this work, as well as its use as a management tool. The third chapter describes the methodology adopted to build the management system, which was based on the SOMA system operationalization. Chapter four explains SMS-Coastal general aspects, its requirements to properly work and its operations sequence. It also presents some aspects related to its use to keep SOMA operational. In the fifth chapter, the main features of the program's operation are discussed, and the sixth, consists of the general conclusions of the work done. The SMS-Coastal's Python algorithm is presented in the appendices section, each of which corresponding to a module of the program.

2 STATE OF THE ART

2.1 Operational Oceanography

The oceans are an extremely important environment for the planet and humanity. They were the birthplace for the origin of life and still are the home to countless species. Due to the occurrence of various phenomena, they develop a fundamental role in the Earth's ecosystem, being responsible to regulate global climate and temperature, to recycle nutrients, to generate oxygen and to absorb carbon dioxide of the atmosphere. This environment has equally unlimited relevance in economic aspects: it is one of the main sources of food, it is the most widely used means of transportation in global trade, provides more than 30% of the global supply of fossil hydrocarbons and offers great potential for renewable energy production (Bari, 2017).

The use of marine resources by man began still in remote times of prehistory, which made possible to blossom some understanding regarding tides, currents and waves (Schiller, 2011). However, oceanography as the area that studies all aspects of the seas, from their physical description to the interpretation of the phenomena that occur in them and their interaction with the continents and the atmosphere, only began to be significant in the great discoveries era, period between the 15th and 17th centuries. The maritime expansion, motivated by the intercontinental business, contributed to the development of knowledge in cartography and of ocean's surface.

After the great expansion, according to Schiller (2011), scientific journeys began to be realized, primarily to produce a more precise mapping of the continents, but also to improve oceanography knowledge. The first scientific texts regarding sea currents, depth of the ocean and meteorological data began to come out as expeditions with the same purpose increased in late 18th century. Nevertheless, a large-scale expedition was not possible until 1872, when a 4-year mission was made specifically to collect a wide range of ocean properties such as temperature, chemical variables of seawater, currents and even soil geology. That was a landmark in modern oceanography and thus soon enough, observed phenomena could be quantitatively described.

As the number of scientific expeditions increased, there were entirely specialized vessels for oceanographic studies. The technological instrumentation used has undergone intense improvements since the 20th century, especially during World War II that boosted this evolution, once knowing more about a region was strategically advantageous and could mean a victory against the enemy. At that time the first sonars capable of mapping the seabed emerged.

Now that the techniques of collecting ocean data were intensified and improved, a large amount of ocean data was available. That combined with previously formulated theories enabled the development of the first mathematical models that would describe hydrodynamic processes in the oceans. By the end of the last century, these models already could be implemented in computational environments and oceanographic tools began to be produced to carry out forecasts states (Böning & Semtner, 2001).

As new numerical models have emerged, the need for more sea properties data has triggered the development of *in-situ* and remote collection equipment to provide almost real-time information of temperature, salinity, velocity on the surface. That created a cycle of information data in a way that these and other measures are used in operational models to provide an integrated description of the state of the ocean and allowing the rising of new monitoring and forecasting techniques (Le Traon, 2011). The last, in turn, play a fundamental role in understanding the dynamics and physical processes of this environment.

Operational oceanography is therefore understood as the constant activity of observing, collecting, interpreting and disseminating the measured data, in order to design methods for analyzing behavior and predicting future conditions (Prandle, 2000). The information generated by this activity is very important for scientific community in the sense of always boosting the development of marine sector technology (She et al., 2016).

2.1.1 Observation Methods

The activities related to operational oceanography help to develop new ways of understanding and predicting ocean behavior and climate evolution. Consequently, there will always be a growing need for information from this environment (Robinson, 2010a;

von Schuckmann et al., 2016). In this manner, one of the pillars of the operational oceanography are the observations systems, that make the ocean data acquisition to feed operational models to produce forecasts and other products (Dombrowsky, 2011).

For operational systems to describe oceans' physical state they require, among other properties, primarily information regarding temperature, salinity, density and absolute currents (Send, 2006). Biogeochemical models need other state variables such as nutrients, oxygen, chlorophyll, phytoplankton and zooplankton biomass. The perfect method to acquire all that data would be one on which measurements would cover all three-dimensional space and time. However, there is no single ideal way to do this and therefore observations systems combine techniques that can be divided basically into two types: locally measurements (*in-situ*) and remotely.

2.1.1.1 In-situ methods

In-situ instruments record mainly physical sea properties and each one of them has its capacity in terms of spatial and temporal resolution (Ravichandran, 2011). The operational network of those data recording mechanisms is non-trivial and requires a great deal of combined effort and coordinated actions, since tasks such as sensor lifetime and their coverage area mapping are critical to avoid measuring gaps. *In-situ* methods can be yet classified into those which are based on fixed points and those whose location varies with time. According to Ravichandran (2011) and Send (2006) most common are:

Profiling floats

Generally used to make temperature and salinity profiles in a water column. The idea is to build a long lifetime, low cost device, with light-weight and low electric power consumption sensors, that sends all the data to satellite systems when at the surface. They passively follow the horizontal currents flow and make buoyancy changes to cover the water column. Biogeochemical properties can also be measured. 2.1 shows an example of profiling float from Argo program, in which the location of the sensors for conductivity-temperature-depth (CTD), oxygen (O₂), acidity or basicity (pH) and nitrates (NO₃) were pointed out.



**Figure 2.1 - Navis BGC Argo float. From:
(Roemmich et al., 2019)**

Surface drifters

Buoys connected to a drogue (2.2) that allows passively following the ocean horizontal flow. They mainly measure sea surface temperature (SST) and air pressure, but can also give information about surface currents if their position is tracked.

Ships

Ships may collect and distribute data in a non operational rate, still they are of great importance in ocean research. Despite being an expensive construction, research vessels are built entirely specialized for oceanography and can be used to collect water samples for biochemical analysis. Furthermore, they are the means of transportation and have mechanisms to launch more complex, bigger and heavier instrumentation. These vessels can also be used as ships of opportunity, mainly to deploy expendable temperature probes (XBT) along their route to their own operations. It is also possible to use merchant vessels as volunteer observing ships (VOS) which allow researchers to launch measurement equipment and to collect underway sampling. However, mostly all of the routes will be

commercial ones and there is always the possibility for the ship operator to change to others not planned.

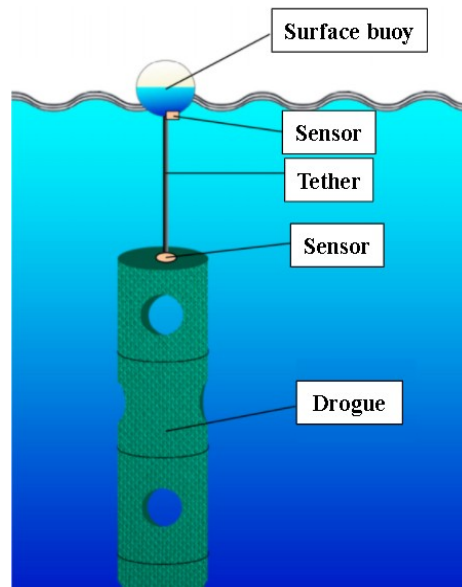


Figure 2.2 - Surface drifter common structure. Adapted from: (Wu et al., 2019)

Moorings and fixed platforms

Normally they are fixed buoys that may be equipped with several different sensors (2.3), that record with a high temporal resolution timeseries of a wide range of properties. It is also possible to get samples along the water column in the position where it is fixed, as well as some atmospheric properties at the surface. Other more robust instruments can be installed to make measurements of radiation, oxygen, carbon dioxide, chlorophyll and other biogeochemical variables. A network of moorings enables the possibility to use a method called acoustic tomography, in which a long distance profile of temperature or currents are obtained as a function of the time it takes an acoustic signal to travel from one instrument to another. In general moorings have a high cost to build and for the maintenance, therefore they are used in low quantity and in places of great interest, where critical ocean processes occurs.

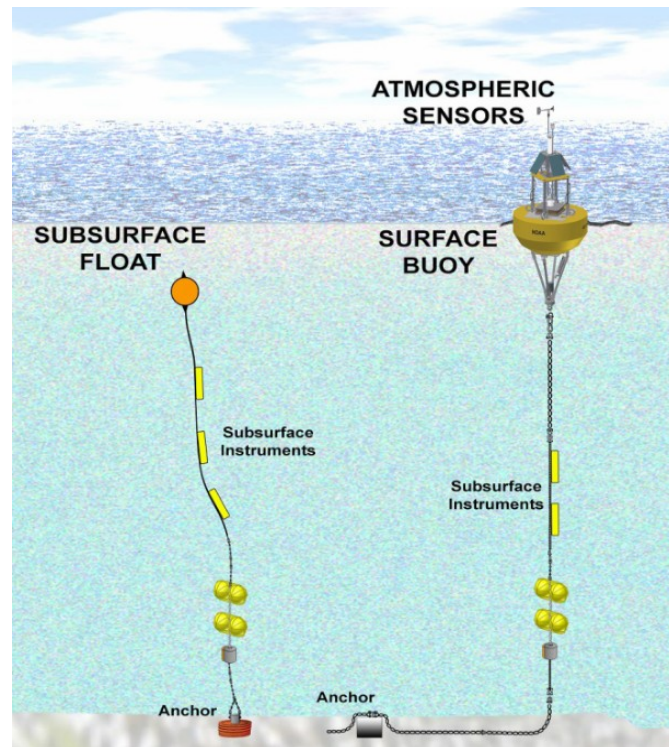


Figure 2.3 - Diagram of coastal moorings. From: (Bailey et al., 2019)

Gliders and autonomous underwater vehicles (AUV)

Vehicles in shape of small submarines that can be programmed to collect samples for a specific mission track. The movement inside water is done actively by propellers in AUV (2.4) and passively by buoyancy changes and wings in gliders. They usually perform an up/down trajectory (sawtooth pattern) to get physical and also some biogeochemical variables in different depths. Their main weakness are regarding total weight, systems energy consumption and limited depth operation.

Radars

High-frequency (HF) radar land-based installations in coastlines can detect sea surface currents providing operational data primarily for ship routing, pollutant transport forecast, algal blooms. Regardless their limited coverage, until 300 km offshore, they have good spatial and temporal resolution. 2.5 shows a diagram of the basic operating principle of the HF radars. The antennas emit electromagnetic signals with a frequency between 6 and 30 MHz on the water surface. The signals scattered back by ocean waves with half the

wavelength of the first are received, and then the surface currents are assessed with the doppler shift of the reflected signals.



Figure 2.4 - Oceanscan light autonomous underwater vehicle used by the research center CIMA at University of Algarve.

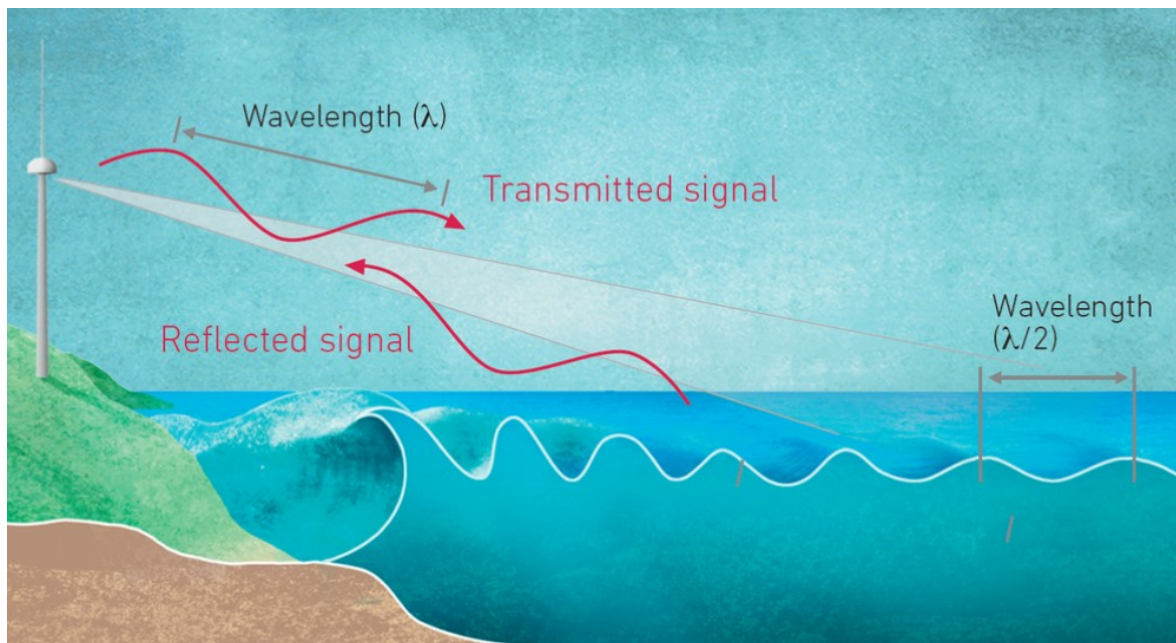


Figure 2.5 - HF radar basic operating principle. From: (Medcliv, 2020)

All the data collected by these *in-situ* methods are applied in countless operational models that use a wide variety of variables (Pouliquen, 2006). In view of that need, observation systems progressively try to increase the spatial and temporal resolution of

their measures. This is the case of the Argo program, which consists of a global array of profiling floats deployed regularly in a large number to provide almost real-time ocean data essential in oceanographic researches (Mittal & Delbridge, 2019). There are over 3900 active free-drifting floats reporting CTD profiles, salinity and velocity of the upper ocean. They have a spacial sampling between 200 and 400 km and can go down to until 2000 m depth. New floats are already under development that can go further than that and even make biogeochemical measurements (Roemmich et al., 2019). For all of its features the project is unique in some aspects and its data is being used to study previously unreachable process such as submarine volcanism (Mittal & Delbridge, 2019) and the Weddell Gyre (Reeve et al., 2019). The black dots in 2.6 represent the position of active Argo floats in May of 2020.

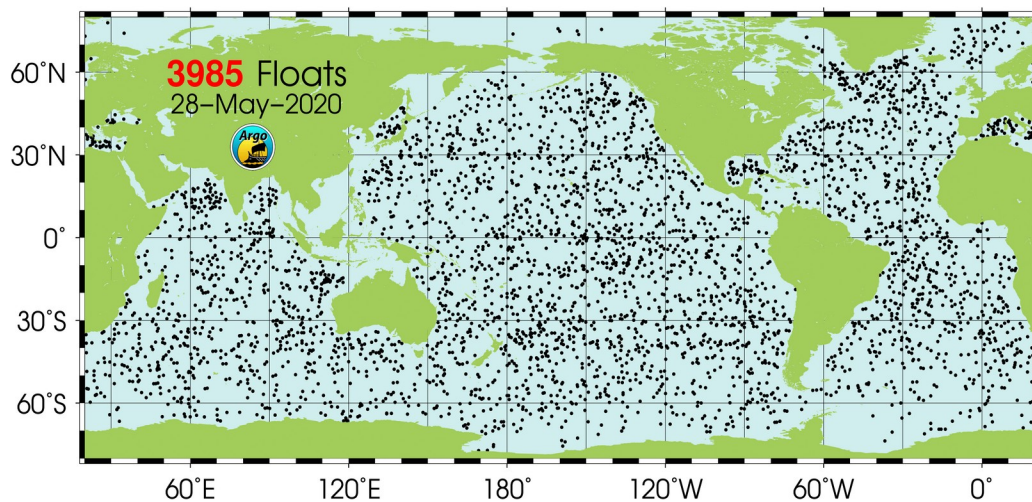


Figure 2.6 - Positions of active Argo floats in May of 2020. From: (Argo Program, 2020)

2.1.1.2 Remote sensing

The other way to collect ocean data is by remote sensing. Mostly done by satellites, it consists of getting, analyze and interpret sea surface electromagnetic signals. Although the sensors are located far away from the data source, this method revealed to be of great importance in operational oceanography considering it gives nearly real time information about valuable ocean properties. Remote acquired data is being used since 1980's and have helped discovering previously unknown aspects of the sea (Robinson, 2010a). The technique is able to cover areas of thousands of kilometers with, however, a spacial resolution capable of providing information from mesoscale to coastal events (Le Traon,

2011). It also has a high temporal resolution and its long-term data accumulation is used to understand ocean variability and to profile the occurrence of extreme events.

Remote instruments are capable of monitor parameters such as SST, sea surface height (SSH), ocean circulation, ice, waves, winds and more recently sea surface salinity. They can also detect ocean colour that gives information about biological properties. Occasionally, to get the desired horizontal distribution and temporal resolution of given surface variable it is necessary a simultaneously operation of several satellites (Le Traon, 2011). The radiation origin received by the sensors classify them into passive and active: the first one collect reflected electromagnetic waves from the sun on the sea, or emitted from the last, and the second emits its own pulses and analyses those reflected back to it from the surface, they operate in the radar frequency band (Robinson, 2006).

When using satellite information, the final user must be aware of the type of data which is provided by the product made available by a system. Essentially they are assorted into levels as follows (Robinson, 2010b):

- Level 0 (L0): raw data in binary form.
- Level 1 (L1): calibrated multi channel signal in scan-line coordinates.
- Level 2 (L2): geolocated ocean data product.
- Level 3 (L3): global gridded data set.
- Level 4 (L4): global analysed gridded data set.

In operational oceanography there is no unique way of collecting data. Remote and *in-situ* sensing are complementary measure techniques between them and inside each one. A variable evaluated in different manners generates calibration and validation tools for sensors and ocean models. Only with the integration of different measurement methods will ocean data have better coverage and resolution. The graphic in 2.7 shows the resolution aspects of *in-situ* and remote observation systems. For operational applications the best quality data will be the one that combines in the most efficient way information from different sources (Send, 2006).

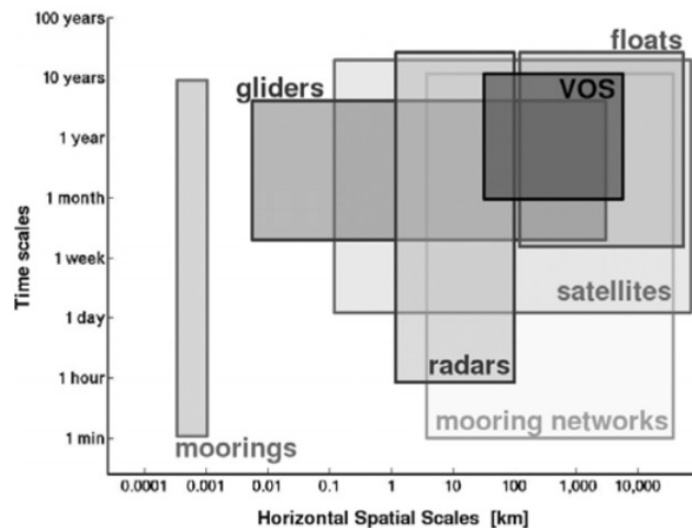


Figure 2.7 - Spacial and temporal resolution at the sea surface of *in-situ* and satellites observation techniques. From: (Send, 2006).

2.1.2 Observation Network

The quantity and quality of data required in operational oceanography is obtained by a well set network of different measurement instruments. Those, however, have high cost and demand a great amount of effort to get them into right spatial coverage. That is why most of the time research centers alone cannot afford them, and for that reason, programs such as The Global Ocean Observing System (GOOS) are created to help build a solid data collection and dissemination framework (Summerhayes, 2002).

When GOOS started, its main activity was the maintenance of the buoys array in Tropical Ocean Global Atmosphere (TOGA) program and of the VOS network. Now, it plays an essential role in operational oceanography, coordinating activities to monitor, describe and predict ocean state on a global scale, including living resources (Malone et al., 2014; Summerhayes, 2002). The program provides ocean observations using an integrated data management and communications system. Furthermore, it works to improve the control of marine and coastal resources, to reduce damage caused by pollution and natural disasters and promote scientific research. The observing system was one of the responsible for creating the concept of Essential Ocean Variables (EOVs) (von Schuckmann et al., 2016), which are physical and biochemical measurable properties, like currents, salinity, temperature, nutrients, phytoplankton biomass, oxygen and others, that give enough information to describe ocean state (Capet et al., 2020).

In order to fulfill all its purpose, the implementation of GOOS is being done by the GRAs (GOOS Regional Alliances). There are already 13 well-defined GRAs and two under development, so that each one consists of an alliance between nations and/or institutions responsible for developing observations systems at regional and coastal scale (Malone et al., 2010). In Europe operational oceanography community is represented by the European Global Ocean Observing System, or EuroGOOS (Capet et al., 2020). Due to that, GOOS is able to constantly improve and design new observation methods that will increase spatial and temporal resolutions of sampling and cover new areas which had no available data yet (Liblik et al., 2016).

Operational oceanography is not made only by a well set observation framework. In its early stages GOOS was trying to establish a global observation network, while there was still a missing link between collected data and assimilation and modelling to generate useful information. This is why the Global Ocean Data Assimilation Experiment (GODAE) was conceived, to evolve oceanography as a research based field to actually an operational one (Smith, 2000). The experiment enabled a routine access to observation data from different sources so they could be used in order to produce timely forecasts. In addition, it established an information sharing environment that assisted the progress of the ocean modelling area (Bell et al., 2009). GODAE was a time determined project that made the ocean prediction process feasible and practical, so that after its end in 2008 a new group was formed to continue its work in a long term, the GODAE OceanView (Bell et al., 2009; Oke et al., 2013).

2.1.3 Operational Oceanography and Blue Growth

Millions of people around the globe depend on the exploitation of marine ecosystem resources. Among the various economic activities that can be developed, oceans are primarily a source of food, can be used for energy generation, exploration of fossil fuels, shipping and also for recreation and tourism. Even though it is already a very valuable business, it is estimated that the ocean economy may still double in the coming years as the population grows in urban centers close to coastal areas (Howard, 2018).

Unfortunately, this exploitation of natural resources often occurs irresponsibly and, therefore, it was necessary to determine an initiative to expand the concept of sustainable development to ocean economy (Bari, 2017). Thus, in 2012, during the United Nations Conference on Sustainable Development, also known as Rio +20, the blue growth term was created to designate practices that aim at economic growth but that also guarantee the preservation of the environment for future generations (Burgess et al., 2018). Therefore, it is a matter of finding the optimum point between exploration and preservation, in order to reduce environmental risks and ecological scarcity (Lee et al., 2020).

Operational oceanography is closely related to the socioeconomic activities of blue growth, as it provides essential marine data to them. This search for sustainable development causes an increasing demand for ocean knowledge, which in turn is one of the agents that drives the improvements of operational oceanography. In this scenario, the continuous expansion process of that area is determining for maintaining the information supply for blue growth (She et al., 2016). For this reason, EuroGOOS has defined four priority development areas within this systematic activity in Europe for the upcoming years, namely ocean observation, modelling and forecasting technology, operational oceanography of coastal areas and operational ecology (She, 2015).

Coastal zones are of great economic importance for society and are key areas of blue growth. European coasts are generally densely populated and are therefore responsible for generating more than 30% of its gross domestic product (GDP) (She, 2015). Because of that, it is essential to understand this environment behavior which is submitted to constant fast changes and, for that, it is still necessary to refine the knowledge of the complex phenomena that takes place in there (She et al., 2016). With this, it will be possible to develop operational coastal models capable of generating value in oceanic data form, which in turn will be used to support sustainable economic growth (Capet et al., 2020).

2.2 Numerical Ocean Modelling

Ocean modelling is one of the most important activities developed within the scope of operational oceanography (Capet et al., 2020), considering that it is precisely through it that it is possible to generate data that demonstrate the aspects and behaviors of this

environment. A model consists in the mathematical representation of phenomena, that in this case are equations to describe the fluid dynamics in oceans. Once a model's governing equations are determined, they are discretized using numerical methods and transformed into computational code to advance or iterate the ocean state in time (Griffies, 2006).

The set of mathematical statements that represent fluid motion forms a system of differential equations, which are constructed respecting fundamental conservation principles of physics (Kämpf, 2009). Ocean models frequently use the Eulerian approach, that consists in the evaluation of the conservation of properties inside a portion of space, or a control volume, while the water flow goes through it. Therefore, in order to solve the equations the first step is to discretize them in space, so that they can be applied individually in each of the control volumes that form the interest region. One of the most commonly used method to do that is finite volumes (Griffies, 2006). Each control volume is an element or cell of the mesh constructed for a domain. Thus, a model with higher spatial resolution has, for the same domain, a mesh formed by smaller elements, but in greater numbers. Despite generating more accurate data, this implies solving a system with more equations and that consequently requires greater computational effort (Greenberg et al., 2007).

The equations that represent the conservation laws are based on the so-called transport equation, which for an infinitesimal cubic control volume delimited in the Cartesian coordinate system, is defined as (Kämpf, 2009; Versteeg & Malalasekera, 2007):

$$\frac{\partial(\rho \phi)}{\partial t} + \frac{\partial(\rho \phi u)}{\partial x} + \frac{\partial(\rho \phi v)}{\partial y} + \frac{\partial(\rho \phi w)}{\partial z} = \sum (Sources - Sinks) \quad (1)$$

In which ϕ corresponds to any intensive property, ρ the density, u , v and w the velocity vector components corresponding to the x , y , z axis, respectively. (1) can also be written as:

$$\frac{\partial(\rho \phi)}{\partial t} + \text{div}(\rho \phi \mathbf{u}) = \sum (Sources - Sinks) \quad (2)$$

On the left side of (2), the first term represents the rate of change in time of ϕ property and the second, the convective term, is the flow rate of ϕ that passes through the surface of

the control volume. On the right side, the sum computes all sources and sinks that can cause production or destruction of ϕ within the fluid element.

The mass conservation principle of an element is expressed by the continuity equation, obtained from (2) by making the sources and sinks equal to zero and ϕ equals to 1, thus:

$$\frac{\partial \rho}{\partial t} + \text{div}(\rho \mathbf{u}) = 0 \quad (3)$$

The conservation of momentum in the control volume is a direct application of Newton's second law, which states that the momentum of a body or particle is only changed through the action of a non null resultant force. In ocean modelling the forces that cause the movement of a fluid particle are due to pressure, viscosity, gravity, centrifugal force, Coriolis and electromagnetic. Therefore, for each axis of the Cartesian coordinate system, (2) leads to:

$$\frac{\partial(\rho u)}{\partial t} + \text{div}(\rho u \mathbf{u}) = \sum F_x \quad (4)$$

$$\frac{\partial(\rho v)}{\partial t} + \text{div}(\rho v \mathbf{u}) = \sum F_y \quad (5)$$

$$\frac{\partial(\rho z)}{\partial t} + \text{div}(\rho z \mathbf{u}) = \sum F_z \quad (6)$$

Equations 3 to 6 are known as the Navier-Stokes equations. They form the governing equations that describe the movement of the fluid, together with the energy balance and state equations, which correlate thermodynamic properties and link temperature and salinity with density. The governing equations are then discretized by a numerical method in order to be solved after specifying the initial and boundary conditions. Each iteration of the solution of this system expresses the conservation of properties in each control volume (Versteeg & Malalasekera, 2007).

2.2.1 MOHID Modelling System

The modelling system Modelo Hidrodinâmico, or simply MOHID, is a program designed to solve the governing equations in order to model marine environments. It has a modular architecture and features to reproduce several physical, chemical and biological processes of that ecosystem (Janeiro et al., 2014). The model was conceived and is supported by the Marine and Environmental Technology Research Center (MARETEC) of the Instituto Superior Técnico (IST) in University of Lisbon, and it is a working tool in many projects of the research center's environmental modeling group (Neves, 2007). Its beginning took place in the 1980s, when numerical modelling in operational oceanography was driven by the emergence of more powerful computer systems, and up until today new versions are released regularly.

The first version of MOHID was developed as a two-dimensional tidal model, programmed in ANSI FORTRAN 77, and discretized the governing equations using the finite difference method. It was commonly used in the study of coastal areas and estuaries. Eventually the system evolved into a three-dimensional version, in which Eulerian and Lagrangian transport models were introduced (Braunschweig et al., 2004), being the reference of the latter a fluid parcel that moves along with the flow instead of a control volume (Kämpf, 2009). As for the vertical discretization, the first 3D version of MOHID system has implemented a double sigma coordinates to better representing the topography. However, it was not possible for a single type of vertical mesh to be suitable to the entire domain and to all influences on water flow, which also vary at each point. In this way, generic coordinates soon replaced the predecessor, which combines different types to better accommodate the heterogeneous regions of the domain (2.8) (F. Martins et al., 2001).

Given the application of the generic coordinates, the model became able to solve the equations for any type of geometry. Therefore, the finite volume approach was introduced, in which the discrete form of the equations is applied macroscopically to the control volume of each cell. Then, the model solves three-dimensional primitive equations for incompressible and compressible flows, in which hydrostatic equilibrium and Boussinesq approximation are assumed. In temporal discretization, a semi-implicit ADI algorithm with two levels of time per iteration is used. A more detailed description of model's discretization can be found in (F. Martins et al., 1998; F. Martins, 1999).

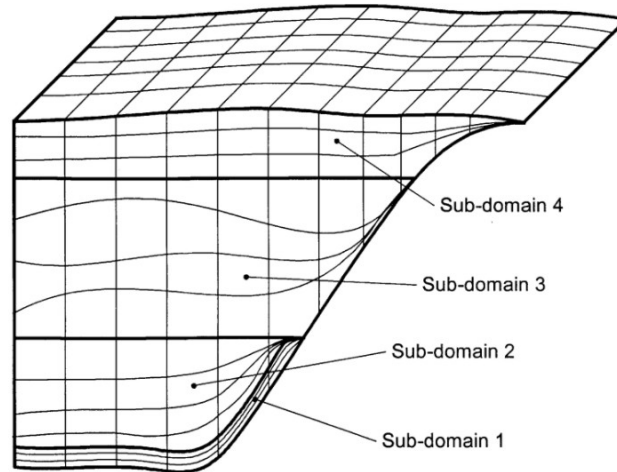


Figure 2.8 - Vertical grid division into four different sub-domains. From: (F. Martins et al., 2001).

Due to the growing number of users and, specially, the limitations of the FORTRAN language used in its algorithm, the model had to go through an update process. In this manner, MOHID code was updated to ANSI FORTRAN 95, which despite not being an object-oriented language, allowed programming in this paradigm (Miranda et al., 2000). This is the moment at which the model assumes the configuration of its current structure: the modular architecture, so that each module having the functionality of a class object. In comparison with the previous version, the model took two to three times more CPU time in the simulations, however the code was much better structured and easier to understand, becoming much more reliable and protected against errors (Braunschweig et al., 2004). Another great advantage that the use of this paradigm brought was to allow the observation of processes in a greater level of detail by enabling the simultaneous simulation of nested models, each one being a new instance of individual classes. These models can then be used in the downscaling methodology, so that the parent model generates the boundary conditions for the child models with more refined grids (Leitão et al., 2005).

The modular design is the basis of MOHID modelling system. Each module of the FORTRAN code represents a class responsible for managing a specific type of information (Neves, 2007). Spacial discretization is established in Geometry module, it stores and updates in each iteration the shape of the finite volume. It is also responsible for the division of the water column in sub-domains. Then, the elements geometry information are

passed to the Hydrodynamic module that actually solves a system of discrete forms of the equations 3 to 6. For a finite volume element as shown in 2.9 and mentioned approximations, in MOHID Navier-Stokes equations become:

$$\frac{\partial u_i}{\partial x_i} = 0 \quad (7)$$

$$\frac{\partial u_1}{\partial t} + \frac{\partial u_j u_1}{\partial x_j} = f u_2 - g \frac{\rho_\eta \partial \eta}{\rho_0 \partial x_1} - \frac{1}{\rho_0} \frac{\partial p_s}{\partial x_1} - \frac{g}{\rho_0} \int_z^\eta \frac{\partial \rho'}{\partial x_1} dx_3 + \frac{\partial}{\partial x_j} (A_j \frac{\partial u_1}{\partial x_j}) \quad (8)$$

$$\frac{\partial u_2}{\partial t} + \frac{\partial u_j u_2}{\partial x_j} = -f u_1 - g \frac{\rho_\eta \partial \eta}{\rho_0 \partial x_2} - \frac{1}{\rho_0} \frac{\partial p_s}{\partial x_2} - \frac{g}{\rho_0} \int_z^\eta \frac{\partial \rho'}{\partial x_2} dx_3 + \frac{\partial}{\partial x_j} (A_j \frac{\partial u_2}{\partial x_j}) \quad (9)$$

$$\frac{\partial p}{\partial x_3} + \rho g = 0 \quad (10)$$

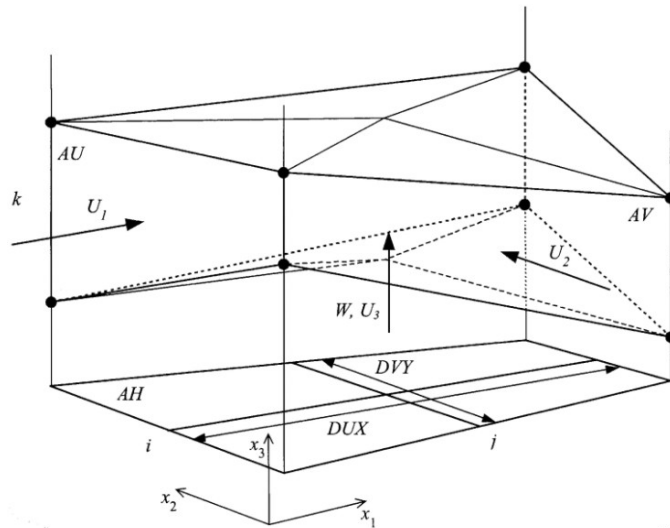


Figure 2.9 - MOHID generic finite volume element geometry. From: (F. Martins et al., 2001).

In the equations from (7) to (10) u_i indicates the velocity vector components referenced by the axis in 2.9 and η , the free surface elevation. In the right hand side of (8) and (9) the first term computes the force with Coriolis parameter f ; the second term represents the barotropic force produced by gradients in the water height, being ρ the density; third term is the barotropic force produced by gradients in atmospheric pressure (p_s); fourth term is

the baroclinic force produced by the vertical integral of horizontal density gradients, where ρ' is the density's anomaly; and the fifth term stands for the diffusive forces, where A_j is the turbulent viscosity coefficients (F. Martins et al., 2001).

Still using the transport equation, the MOHID system manages to coordinate the evolution of properties values in the water column in the WaterProperties module (Neves, 2007). The concentration of a parameter is balanced through advective and diffusive flows, fresh water discharges, sediment flows, surface heat and oxygen exchanges, among others internal sources and sinks. With this module it is possible to simulate up to 24 different properties, that in addition to the physical as temperature and salinity, it also does for biochemical ones such as phytoplankton, zooplankton, ciliate bacteria, nitrogen, phosphorus and oxygen. Due to the modular design that brings the features of an object-orientated programming, new properties can be easily added. The WaterProperties module uses the Eulerian approach to solve the equations, however there also is the Lagrangian module, which can compute the evolution of the same properties with the Lagrangian approach.

Processes involving non-conservative properties are handled by other modules, as is the case of oil dispersion, ecology and turbulence. This last module uses the formulation from the General Ocean Turbulence Model (GOTM). The system consists, at this moment, of more than 70 FORTRAN modules, adding up to more than 300 thousand lines of written code. Some of them require extra information to be executed correctly, they are parameters passed through a keyword system written in ASCII text format in a specific file for each module. The set of those files constitute the model setup which is simulated in the MOHID environment.

The MOHID system has evolved from a simple two-dimensional hydrodynamic model to a complex three-dimensional support tool for research and studies. In the scientific community, it has already been used to simulate, in several estuaries in Portugal, including transport processes, sediment dynamics and water quality (Trancoso et al., 2009), in addition to also being used for modelling ocean circulation and oil spills (Janeiro et al., 2017). The complexity of the numerical tool led to the development of a graphical user interface (MOHID GUI), in which it is possible to create and edit the data and directory

structure necessary to configure a set of simulations, and a tool for visualizing georeferenced data and results (MOHID GIS) (Braunschweig et al., 2004).

2.3 Python Programming Language

Developed at the Centrum Wiskunde & Informatica (CWI) in Netherlands, by Guido van Rossum, who remains one of its main authors, Python is a multi platform general-purpose programming language suitable for lots of different applications. Since its first launch in the early 90s, all releases are free license and open source (Sáenz et al., 2002). Its code consists of numerous versatile modules and presents a clean structure, being consequently easy to understand and to learn, making it very attractive to new programmers. This is one of the reasons why Python has an active and fast increasing community of users and also is one of the most adopted programming languages, being widely used for artificial intelligence, data science and machine learning (Casanova-Arenillas et al., 2020).

Python is a high-level language with dynamic typing, that is, variables declaration at the beginning of the script are not necessary. Also, it uses a tab indentation to delimit blocks of code, thus all tasks performed within a repetition cycle or decision command for example are typed with forward indentation. The code group is then completed when the algorithm returns to original tab without the need of an end statement (Van Rossum & Drake Jr, 1995). Furthermore, Python is a language built based on the object-oriented approach, using the most common model of this paradigm, the class-based one (Perez-Schofield & Ortin, 2019). One of the great advantages of object-oriented programming (OOP) is the possibility to create reusable entities by specifying sharing relationships, hence objects are instances of classes that determine which information they contain, their attributes, and what operations they can perform (their methods) (Ungar et al., 1991). According to Chambers et al. (1991), two of the OOP's peculiar features are:

Inheritance

The most basic feature of OOP enables code sharing avoiding unnecessary programming effort, in which a subclass inherits from a super-class or ancestral class any of its attributes and methods.

Encapsulation

Restriction of the direct access to some of the object's components, some implementation details of a class are kept unassessible to the user. In this way it is possible to use a method without knowing how it internally works, or to make updates to the object without any other components that use it being affected.

All the data introduced in Python corresponds to classes. A simple variable that contains text information is an object or an instance of the class *str*, that transforms its value into a string, which in turn has its attributes to define and its methods to modify it (J. P. Martins, 2012).

Regarding data structures, there are three basic types available in Python: tuples, lists and dictionaries. The first one is an immutable sequence of elements separated with comma and inside parentheses that resemble the vector idea in math. Among other applications, tuples can be quite handy when the user wants to keep a set of values that will be used several times along the script, since they consume less memory space than the others. Lists are very similar to the last, but are defined between brackets, have more methods available to modify their elements and are commonly used as matrices. The last type are the dictionaries, created between curly brackets, and unlike lists and tuples its elements order are not important since they are organized in pairs. They can commonly be found in other programming languages in what called associative arrays or memories. The pairs, separated by commas, are formatted as *key:value* in which keys cannot be repeated and values may assume any other data structure or variable. A certain value is accessed when the corresponding key is requested.

Python features a library of built in versatile modules that allow quickly creation of sophisticated algorithms for all kinds of tasks. With the *import* statement plus the name of the desired module it is possible to summon tools for files edition and handling, operative system commands, time and date manipulation, parallel script execution and a whole lot more (Oliphant, 2007). Through additional download of external modules, Python computational power can be intensely increased, such as with SciPy project, short for Scientific Python, which contains a collection of open-source packages for science, engineering and mathematics (Oliphant, 2006). One of them is an elegant and efficient way

to work with large datasets in matrix of any given dimensions, instead of using lists, widely used in academia, the NumPy N-dimensional array (van der Walt et al., 2011).

To help develop their applications, programmers often rely on the growing community of Python users, recognized for their fast and useful responses (Oliphant, 2007). This network allied with its characteristics make Python one of the most widely used languages today, especially in the scientific domain in which most users have decided to adopt it as the main tool in replace of other major commonly used software (Casanova-Arenillas et al., 2020). Therefore, Python is already a recognized instrument in environmental modelling and is being used as a method in many published researches, such as: a pre- and post-processing tool for the Precipitation-Runoff Modeling System (PRMS) (Volk & Turner, 2019), a networked resource system simulators support library (Knox et al., 2018), a planning and analysis library for water resource systems (Tomlinson et al., 2020), a greenhouse gas emissions visualization toolkit (Wohlstadter et al., 2016), an application to obtain shoreline position time-series (Vos et al., 2019), uncertainty analysis for environmental models (White et al., 2016) and several others.

Similar to what is proposed in this work, Marta-Almeida et al. (2011) used Python to develop an engine to automate forecast simulations for the Regional Ocean Modelling System (ROMS). ROMS has its code written in Fortran like MOHID. The programming language chosen has the necessary tools to perform the most essential task during the forecast cycles, which is the control of model inputs and outputs. Thus, according to the authors, in a single cycle the program must perform the following fundamental steps:

1. Verify the availability of required external forcing data, that could be information regarding tides, river or other fresh water discharges, atmospheric surface heat and momentum fluxes.
2. Interpolate data to model grid.
3. Check for the initial conditions which are obtained from the restart files of the previous day cycle.
4. Write model input files.
5. Run simulation.

6. Manage model outputs and prepare restart files to the next forecast cycle.

Since daily simulations can easily produce many gigabytes of data, it is important to create a cleaning module that will erase not necessary input or output files between forecast cycles. In this way, the computer that performs the simulations will always have free disk space, hence the model database must be built in another location or storage devices.

According to Marta-Almeida et al. (2011), keeping forecast cycles in constant operation involves dealing with several different tasks, which can cause the code to have low performance. Despite this, Python still is a good option for ocean modelling, because, besides all its built-in properties, it has tools to handle common data formats in ocean atmospheric sciences such as HDF5, GRIB and NetCDF, support for OPeNDAP, tidal harmonic analysis software and scientific visualization resources.

3 METHODOLOGY

Coastal regions are places of quick changes and are subject to most of the dynamic changes caused by winds, waves, tides, sediment transport, human activity. Therefore, it is necessary to generate new knowledge to integrate these and other interactions of coastal hydrodynamics in order to produce operational models capable of providing reliable predictions of the ocean state (She et al., 2016). Due to the high variability of the ocean processes, coastal models generally run simulations for not too many days at a time and aim to represent detailed information in a limited domain but with a high resolution grid (Send, 2006).

The Algarve Operational Modelling and Monitoring System (SOMA) arose from the need to produce a high-resolution operational model, with the purpose of providing predictions of the sea state and the trajectory of oil spills on the Algarve coast (Janeiro et al., 2017). The authors wanted to investigate the efficiency of downscaling methods in determining the sources of oil leakage by combining backtracking simulation with vessel trajectories and using lagrangian particles. SOMA is a validated model and is enabled to make predictions of the hydrodynamic behavior and water properties of the implemented region. In addition to traditional calibration and validation, an operational model must be continuously assessed dynamically and this is also being done for SOMA under the OCASO project (Lorente et al., 2019; IP, 2006). For those reasons, it was used as the basis for the creation of the simulation management tool proposed in this work.

SOMA is built within the environment of MOHID modelling system, which due to its architecture is a suitable and robust tool for downscaling methods (Janeiro et al., 2017). The first version of the operational model consisted of two grid levels: the first one had a 3 km resolution grid and hybrid vertical spatial discretization, being 11 layers of sigma coordinates at the first 20 m of depth and 35 in unequally spaced Z coordinates to the bottom; the second level presented the same vertical profile of the first, but with horizontal resolution of 1 km. Simulation step time was of 30 and 15 seconds for the first and second levels respectively. The communication between the levels was performed by the flow relaxation scheme (FRS) method. In 3.1 the geographical location of the two grid areas is

shown, as well as of the *in-situ* observation systems used for the model validation and calibration.

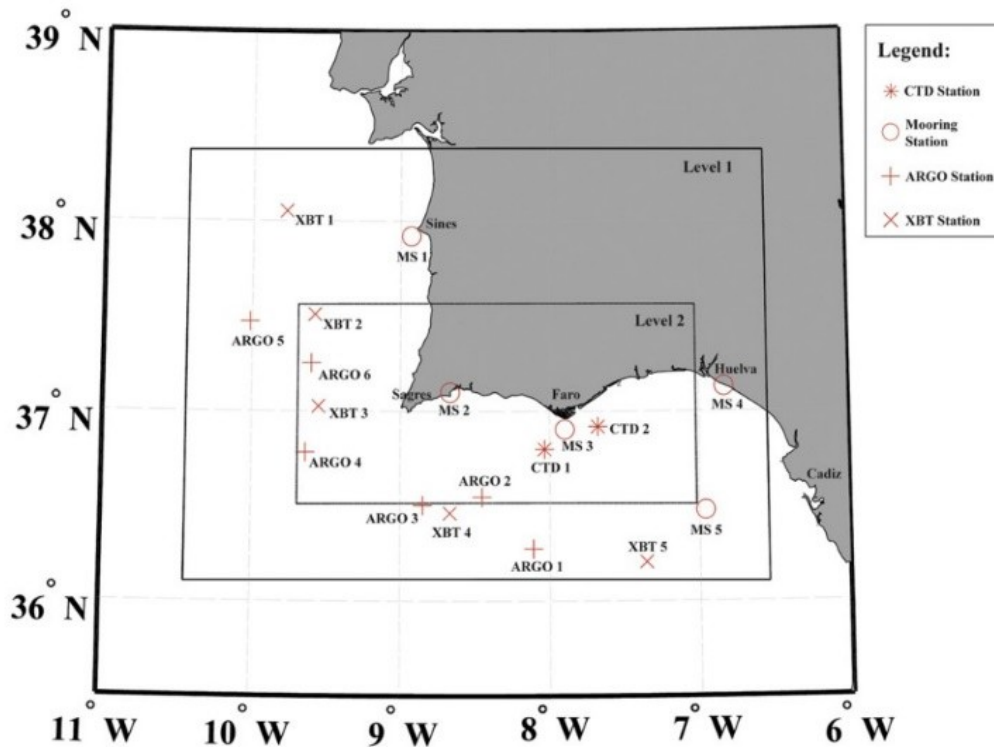


Figure 3.1 - SOMA levels and location of in-situ observation systems. From: (Janeiro et al., 2017).

A different version of the SOMA system was used to build the SMS-Coastal. The main change presented by the current version of the operational model are the inclusion of an additional mesh level and an increase of resolution.

The original SOMA model had only two nested levels, both 3D where the first level was forced at its boundaries by a combination of hydrodynamic and tidal forcings. This was possible because the data provider was the IBI model that also solves the tide explicitly. In this version the data provider was changed to the Mercator Ocean analysis and forecast provided by Copernicus Marine Environment Monitoring Service (CMEMS). This new model do not solve the tide explicitly. Due to that, an additional grid level was added above the previous two levels. This new level is a simple 2D hydrodynamic model, forced by the FES2012 global tidal solution and has as its single purpose supply the tidal conditions to the lower levels.

The increase in resolution was applied in the first 3D level to improve the quality of the results. Numerical experiments have shown that an increase from the previous 3 km space step to 2 km leads to significant improvements in the solution. Evolution of computational capacity now allows the solution of this 2 km grid in an acceptable amount of time. 3.2 shows the representation of the present bathymetry for levels 2 and 3 of the model.

The operationalization of SOMA consists in two types of simulation cycles, daily and weekly runs. The forecasts are obtained from the execution of the first type, which in addition to the external forcing data, also need the initial condition files generated in the previous day's cycle. As initials conditions start to degrade due to the sequence of daily runs, a weekly run is done to start a new solution of the model to provide fresh restart files. Until the model is able to run at full speed, weekly cycles are divided into two stages with gradual increase of step time to prevent simulation instability.

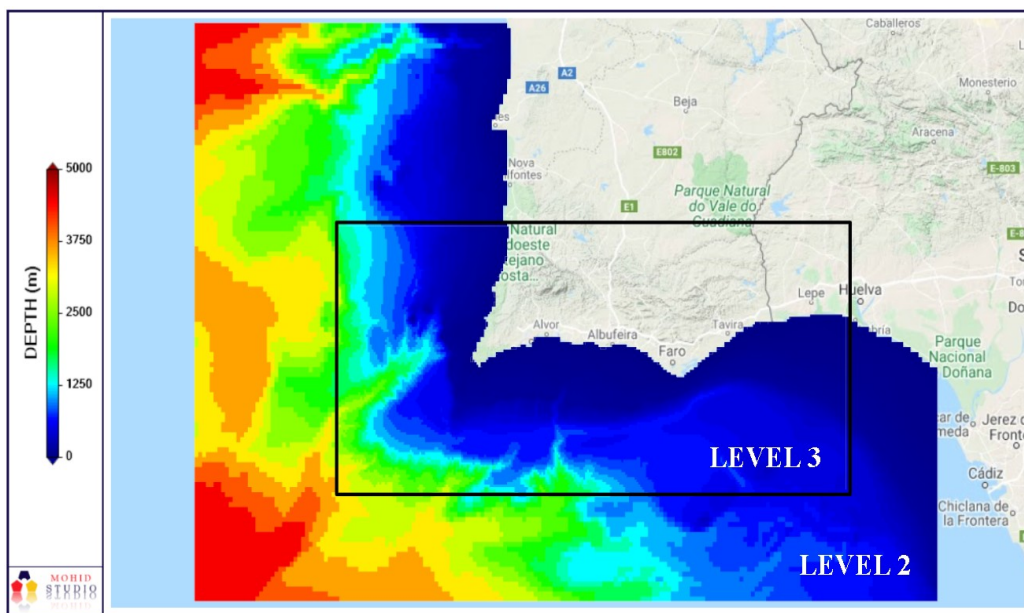


Figure 3.2 - SOMA current bathymetry.

The diagram shown in 3.3 demonstrate the simulation cycles processes. Assuming a four-day forecast, a new daily run starts (blue ribbon in the figure) and, after a day of simulation time, the model data has instructions for MOHID to write the initial condition files for the next day simulation (red ribbon). After seven daily cycles, a weekly run (green ribbon) is executed in hindcast mode to provide the initial conditions for the eighth day simulation. In

that eighth day the weekly cycle restart files should always take priority over the daily simulation files that runs in parallel.

For external forcing the model uses daily mean physical properties of CMEMS Mercator, and as for atmospheric data, from Skiron forecast system provided by Atmospheric Modeling and Weather Forecasting Group of The National and Kapodistrian University of Athens. These sources will constitute the first external data that SMS-Coastal will be able to process, which in turn will have continuous updates to add other sources to its library. Thus, the system downloads necessary files and conduct conversion and interpolation operations performed by the MOHID supporting tool ConvertToHDF. In this way, SMS-Coastal main body will be divided into two basic layers, one to download and process data from external sources and the other to manage simulations.

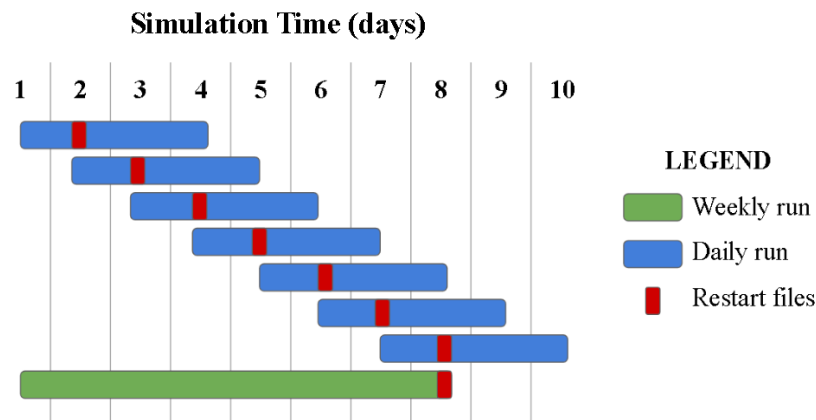


Figure 3.3 - Simulation cycles diagram.

SMS-Coastal was designed to read a specific set of folders, which was based on the construction of a project in MOHID GUI, as shown in 3.4 for SOMA. Each level of the model contains the MOHID modules reading data files for each simulation cycle, so that coldstart and hotstart in the figure represent the two stages of weekly run mentioned before.

In order to load a simulation with MOHID system, its executable needs to know the hierarchical structure of the nested models and which modules to activate. SMS-Coastal will then give the information regarding levels arrangement by writing the "Tree.dat" file (3.5) in the working directory of the father domain in the beginning of a simulation process. The chosen MOHID modules and its correspondent data files are specific for one

model and are defined in “Nomfich.dat” file (3.6) with keywords method. However, SMS-Coastal must ensure that the correct one for each run is placed in the working directory of each level. MOHID must be launched when the operating system current working directory coincides with that of the father domain, and this is why all computational paths, as indicated in the figures, are written using relative paths, based on the "father" directory. Finally, SMS-Coastal will be the one in charge to generate the “Model.dat” file (3.7) for each run and level, which specify to MOHID information related to time, such as simulation initial and final dates, iteration seconds and the time zone.

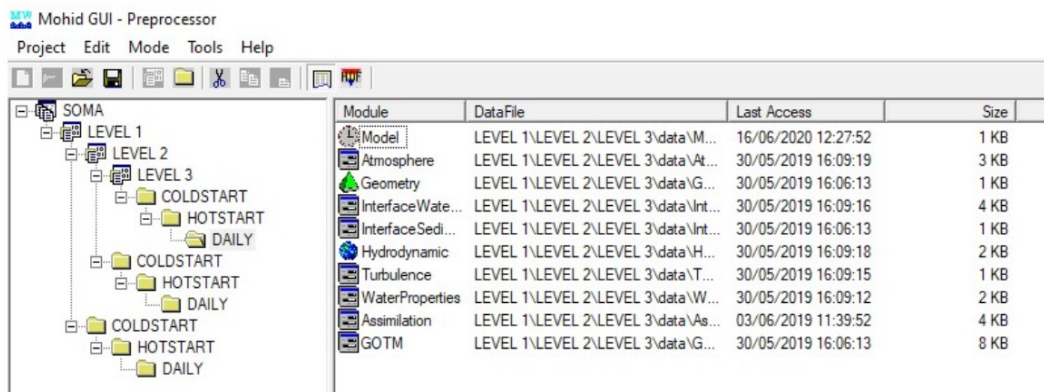


Figure 3.4 - SOMA's built in MOHID GUI.

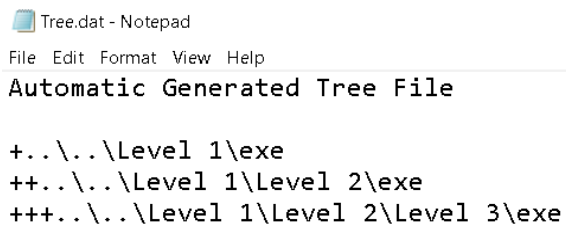


Figure 3.5 - Hierarchical structure instructions for three nested models.

The most basic operations performed in each cycle type by the SMS-Coastal were delimited as shown in the diagram of the 3.8, based on the essential tasks of a simulation manager defined in chapter 2.3. Firstly, it shall download and process external oceanic and atmospheric data independently, so that when there is more than one source defined for the same type of data, they will serve as redundancy in the event of failure in the operation of one of them. Secondly, it will generate and control the simulations through the handling of files and folders. For the both layers the program should be able to send e-mails to report

their status. Lastly, it should conduct operations to format the simulation results. 3.1 specifies the vital Python modules used to build SMS-Coastal code so it could accomplish all its functions.

```

Nomfich.dat - Notepad
File Edit Format View Help
IN_BATIM      : ..\..\General Data\Digital Terrain\BATIM_LV1.dat
ROOT         : ..\res\
ROOT_SRT     : ..\res\Run1\
IN_MODEL     : ..\data\Model_1.dat
SURF_DAT     : ..\data\Atmosphere_1.dat
SURF_HDF     : ..\res\Atmosphere_1.hdf
DOMAIN      : ..\data\Geometry_1.dat
AIRW_DAT     : ..\data\InterfaceWaterAir_1.dat
AIRW_HDF     : ..\res\InterfaceWaterAir_1.hdf
AIRW_FIN     : ..\res\InterfaceWaterAir_1.fin
BOT_DAT     : ..\data\InterfaceSedimentWater_1.dat
BOT_HDF     : ..\res\InterfaceSedimentWater_1.hdf
BOT_FIN     : ..\res\InterfaceSedimentWater_1.fin
IN_DAD3D    : ..\data\Hydrodynamic_1.dat
OUT_DESF    : ..\res\Hydrodynamic_1.hdf
OUT_FIN     : ..\res\Hydrodynamic_1.fin
IN_TURB     : ..\data\Turbulence_1.dat
TURB_HDF    : ..\res\Turbulence_1.hdf
DISQUAL     : ..\data\WaterProperties_1.dat
EUL_HDF     : ..\res\WaterProperties_1.hdf
EUL_FIN     : ..\res\WaterProperties_1.fin
OUTWATCH    : ..\res\Outwatch_1.txt
DT_LOG      : ..\res\DTLog_1.txt
IN_TIDES    : ..\data\Tide_1.dat
AIRW_INI    : ..\res\InterfaceWaterAir_0.fin
BOT_INI     : ..\res\InterfaceSedimentWater_0.fin
EUL_INI     : ..\res\WaterProperties_0.fin
IN_CNDI     : ..\res\Hydrodynamic_0.fin

```

Figure 3.6 - MOHID modules instructions for the father domain of SOMA system.

```

Model_1.dat - Notepad
File Edit Format View Help
START       : 2020 06 11 12 00 00
END         : 2020 06 15 12 00 00
DT          : 60
VARIABLEDT : 0
GMTREFERENCE : 0

```

Figure 3.7 - Simulation time instructions.

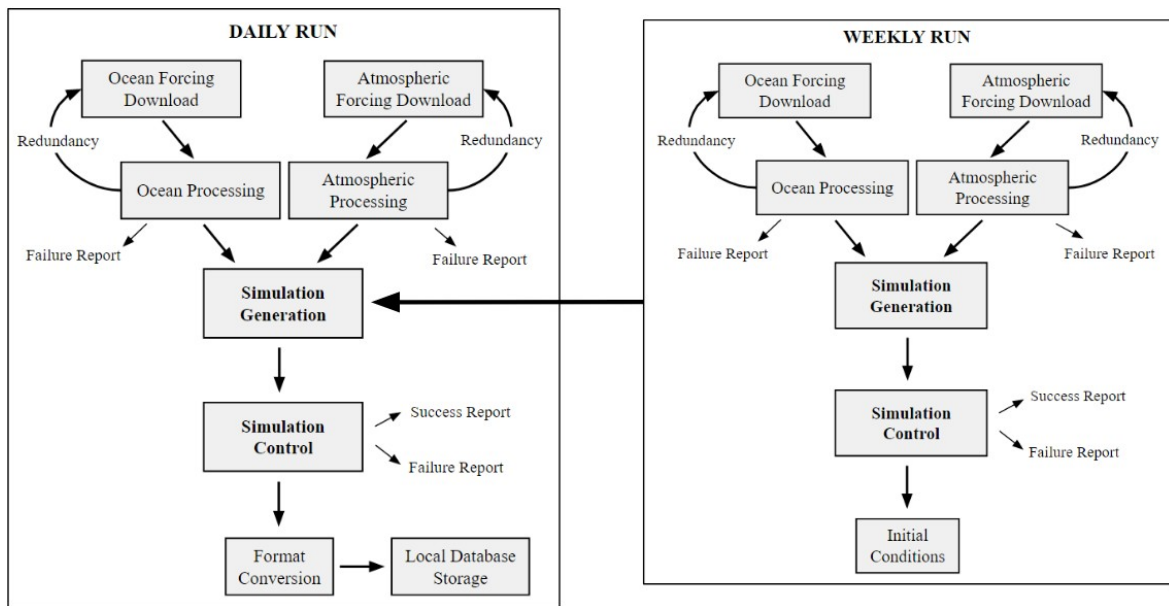


Figure 3.8 - Simulation cycles essential operations sequence.

Table 3.1 - Essential Python modules in simulation manager.

Python module	Library type	Function
os	Bult-in	operating system dependent functionality
Shutil	Bult-in	high-level file and directory handling
datetime	Bult-in	manipulate dates
glob	Bult-in	work with path names
subprocess	Bult-in	spawn new processes
threading	Bult-in	open modules in different threads
numpy	External	operations with multidimensional array
h5py	External	reading and writing HDF files
netCDF4	External	reading and writing NETCDF files
gdal	External	reading GRIB files

4 RESULTS

4.1 Main data structure

This section presents the structure of directories and files created specifically for the SMS-Coastal operation. In order to be managed by the system, a model must attend the specifications of this structure. The model data files, those of external forces, SMS-Coastal and MOHID executables and all others necessary to run a simulation must be placed in a single folder identified as the project directory. The name of this folder should be conveniently the same as the project, since SMS-Coastal may use it in some processes to address files and reports. During operations, several paths are treated relatively to the project directory. For that reason, all others inside the main project folder must have a very well-defined standard format. Thus, the project folder contains the following set of folders and files:

- FORC: stores hydrodynamic and atmospheric external forcing data;
- MOHID: location for MOHID executable, its libraries in dll format and the supporting tools Convert2Hdf5 and HDF5Extractor.
- Sim_Daily: daily simulations directory;
- Sim_Weekly: weekly simulations directory;
- SIM_Manager.exe: SMS-Coastal executable for windows operating system;
- init.dat: data file with user inputs for the system.

There are two simulations folder, one for each cycle defined in the methodology, weekly and daily runs, and both of them have the same internal structure. Furthermore, SMS-Coastal will create the "FORC" folder if it does not exist, as well as its entire internal content in the forcing layer.

As this structure was based in the same one created by MOHID GUI, each simulation directory contains the folder of the first level and "General Data" one in the model set up. This folder is used to store files for bathymetry, tide data, initial conditions and time series location. Bathymetry files must be named in the pattern "BATIM_LVi", in which "i" is an

integer corresponding to the level. Moreover, SMS-Coastal will store here processed external forcing data. It also creates here a third folder, “Operations”, to place all simulations outputs such as HDF5 and log files, as well as formatted results to build the model database, files of a failed run and, in case of SOMA, results converted to NetCDF4 format.

Still following the structure logic of MOHID GUI, each level folder contains three items plus the directory of the subsequent level, if any. Two of the items are for exclusive use of the MOHID executable and of the SMS-Coastal operations, therefore do not require any action on the part of the user. They are the “exe” folder, which is the working directory of a domain and holds files containing information of computational paths to conduct the simulation, and “res” for MOHID outputs of that level. The last one, “data”, is the folder in which the set of data files with the user-defined parameters for the resolution of the governing equations by MOHID is located.

The scheme in 4.1 shows the configuration of folders and files for a generic project. Any information regarding paths must be written, in each data set, relative to the "exe" folder of the model’s first level. In addition, in order for MOHID to find the forcing data files during a simulation, the following naming standard must be followed:

- HYDFORC_LVi.hdf5 and ATMFORC_LVi.hdf5: respectively ocean and atmospheric data for the whole grid, in which “i” is an integer of the corresponding level;
- HYDFORC.dat and ATMFORC.dat: respectively ocean and atmospheric data in time series for a single point of the grid.

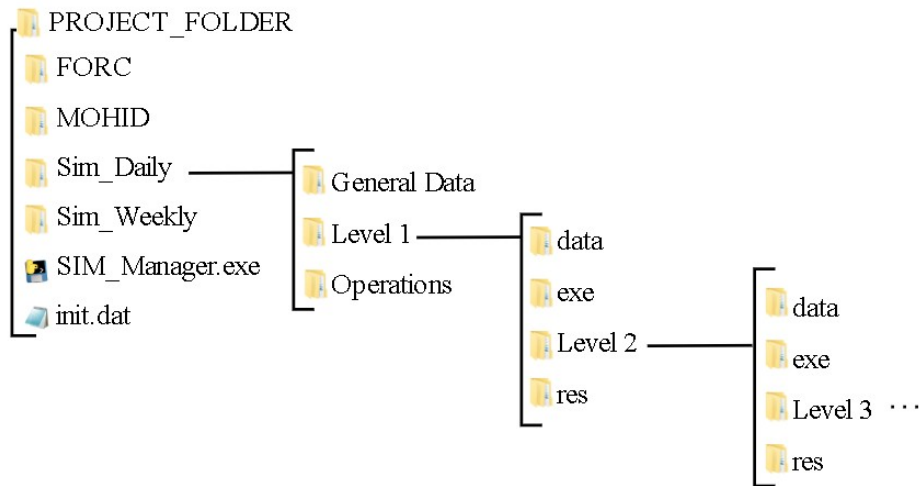


Figure 4.1 - Scheme of a generic project structure.

4.2 User inputs

In the same way that MOHID reads parameters for the resolution of the fluid flow equations, SMS-Coastal reads user's inputs through a system of keywords written in `init.dat` file at project directory. Some of those are indispensable, so the first task performed by the program is to check if all of them have been correctly entered, otherwise the execution is interrupted. The keywords can be classified into three types: mandatory (4.1), optional (4.2), and dependent mandatory (4.3). The incorrect typing or the lack of mandatory words aborts the progress of the program and may also cause improperly stops.

In 4.1, "OPTYPE" must be "1" so SMS-Coastal conduct a simulation management operation. Since in the optics of the system there are two types of models, one in which the first level is part of the forecast data set and another in which it only serves to generate the hydrodynamics for the interior levels, "MODSET" can assume only two values: "1" and "2" respectively. The keyword "RESTART" takes on the value in Python that corresponds to the day of the week desired to start a new cycle of solutions, so that Monday is "0", Tuesday is "1" and so on. The "i" in "DTLVi" is the integer to the corresponding level. It is defined by an ordered list of integers which are the time range in days for each stage in the weekly run cycle and the last one for daily run. SMS-Coastal validates if every "DTLVi" entered have the same length and if there is one for each level as in "LEVELS".

Table 4.1 - Mandatory keywords.

Keyword	Description	Format	Accepted Values	Example
OPTYPE	select operation type for the tool to run	integer	1	1
MODSET	model configuration	integer	1 or 2	2
LEVELS	number of levels of the model	integer	≥ 0	3
RESTART	week day to start a new solution cycle	integer	from 0 to 6	5
GMTREF	GMF reference of the model	integer	--	-1
TRANGE	range in days ordered for each stage	list of integers	> 0	2 4 5
DTLVi	step time in seconds for the iterations of each stage of each level	list of integers	> 0	5 10 15

Table 4.2 - Optional keywords.

Keyword	Description	Format	Accepted Values	Default
OPDATE	operation date	date YYYY MM DD	2020 07 30	today's date
FMT	format output results to populate database	logic	1 or 0	0
PDE	format output results to Puertos del Estado	logic	1 or 0	0
MAILTO	e-mail address to send reports	string	--	None
HYDSRC	external sources for ocean data	list of strings	--	None
ATMSRC	external sources for atmospheric data	list of strings	--	None
HYDTS	set external ocean data output as time series	logic	1 or 0	0
ATMTS	set external atmospheric data output as time series	logic	1 or 0	0

Except for “HYDSRC” and “ATMSRC” in 4.2, all optional keywords have a standard value if they are not inputted by the user. The ones with logic format are for switching on and off subroutines inside the program, with “1” and “0” respectively. When “OPDATE” is

not passed, the tool assumes that operation date is today. The program will run even if the user does not specify sources for external data, it will not process the forcing layer or search for forcing files in the simulation one.

Whatever sources for external data are specified, it is necessary to input the grid limits in “LATLIM” and “LONLIM” (4.3). In this way, the forcing layer is going to download and interpolate data for the area formed by those limits. The format of these keywords is a list of two real numbers, or floats, that can be out of order and have decimal separator of “,” or “.”. If one of the strings in “HYDSRC” is “Mercator”, it must be indicated in “MERC_CRED” the user and password to login into CMEMS database so the tool is able to download the files. Last but not least, ocean and atmospheric external data standard output format is a HDF5 file, nonetheless this can be changed by enabling "HYDTS" or "ATMTS". By doing so, the location of time series must be defined in "TSLOC", first the latitude value and then the longitude.

Table 4.3 - Dependent mandatory keywords.

Keyword	Description	Format	Example	Default
LATLIM	grid latitude limits for external forcing process	list of floats	40.0 35.5	None
LONLIM	grid longitude limits for external forcing process	list of floats	-5.0 -12.0	None
TSLOC	time series location for external forcing process	list of floats	38.8 -7.5	None

At the end, all the inputs read from init.dat file are converted into a Python dictionary. 4.2 shows an example of a keywords set up to manage SOMA simulations.

```
init.dat - Notepad
File Edit Format View Help
OPTYPE      : 1

LEVELS     : 3
RESTART    : 3
GMTREF     : 0

TRANGE     : 2 4 4
DTLV2     : 10 20 30
DTLV1     : 60 60 60
DTLV3     : 5 10 15

MODSET     : 2
FMT       : 1
PDE       : 1

HYDSRC    : Mercator
ATMSRC    : Skiron
LATLIM    : 40.0 35,5
LONLIM    : -5,0 -12,0

MERC_CRED : cmems_user cmems_password
```






Figure 4.2 - SOMA init.dat set up.

4.3 Simulation operation

This section presents the diagrams that represent the programming logic of the algorithm implemented in Python to execute the procedures related to a simulation management. These diagrams are simplification of the code, in a way that only the most relevant processes were described. The full code can be consulted in the Appendices section. The patterns of the shapes used in the flowcharts can be interpreted as indicated in the 4.4.

The first task that SMS-Coastal performs is to read the inputs from the init.dat file, both of them located inside the project directory. After identifying that the keyword "OPTYPE" has a value of "1", it initiate the sequence of operations that controls the execution of the forcing and simulation layers. When SMS-Coastal starts a new solution cycle, by the day indicated in the keyword "RESTART", it divides itself into two threads, one to operate weekly run and the other for the daily one, which are carried out concurrently. On the other days of the week, the simulation layer consists only of the daily cycle. Despite being different operations, within the simulation generation and control blocks of the 3.8, the two cycles make use of the same methods, available when each one creates an instance of the class "Simrun" (4.3).

Table 4.4 - Description of the shapes adopted.

Shape	Description
	start/end of a module or a sequence of operations
	flow direction
	process or operation
	decision
T / F	“T” for a true statement and “F” for false
	beginning of a fixed repetition cycle

As for the forcing layer, it will be carried out in two stages. The first one is executed before any simulation cycle, to process at least one of the sources inserted in "HYDSRC" and in "ATMSRC". Therefore, as soon as there is available data, the program moves to the simulations layer. The second stage starts only after the end of the daily run, just to download unprocessed sources in the first stage. That sequence was represented in the diagram in the 4.3.

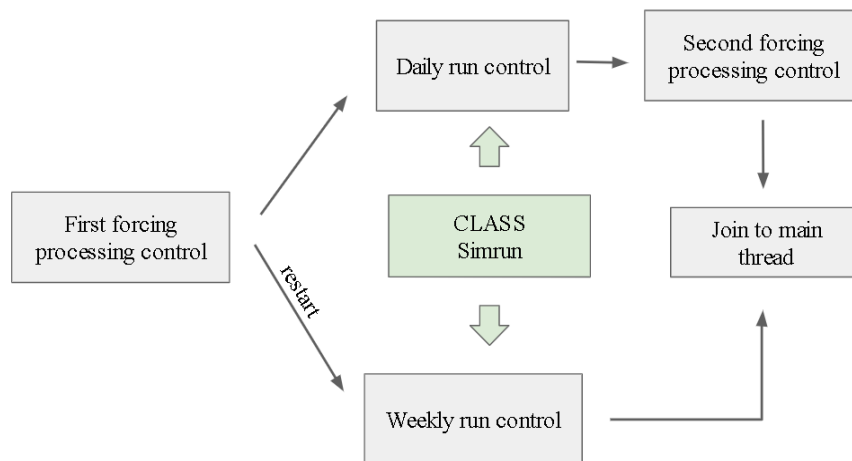


Figure 4.3 - Main processes of the simulation management.

The diagram in 4.4 shows the implementing logic of the Python algorithm for the simulation operation. The abbreviations WR and DR respectively indicate the weekly run

and daily run cycles. The acquisition of hydrodynamic and atmosphere data are independent procedures between them, thus to avoid delay of the simulation cycles the program splits into more than one thread in the forcing layer first stage. Formatting results for database and their conversion to NetCDF are represented by outputs conversion block.

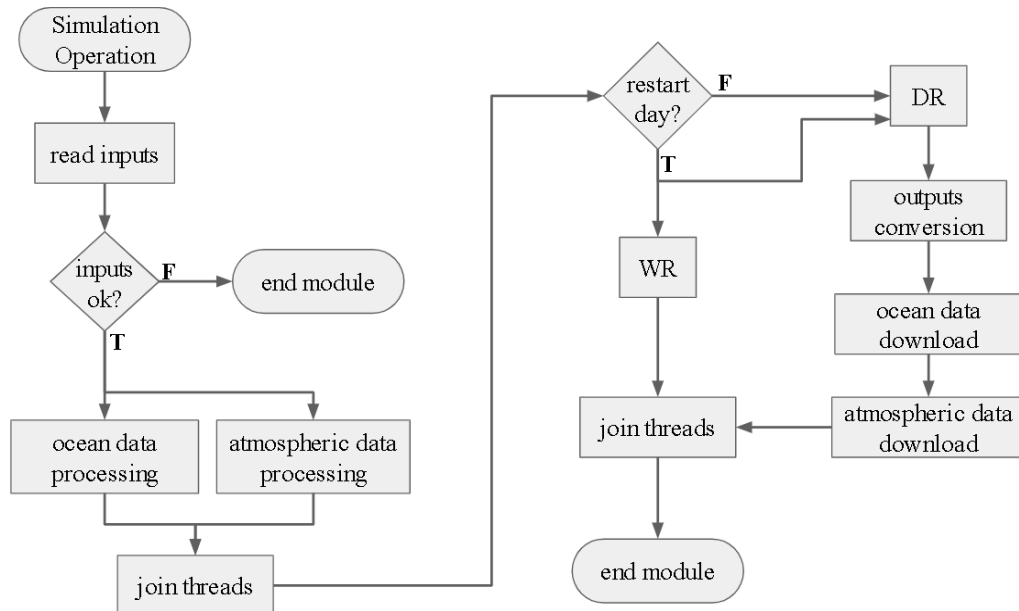


Figure 4.4 - Simulation operation algorithm logic.

4.3.1 Daily run cycle

The operations conducted in a daily run cycle are summarized in the diagrams of figures 4.5 and 4.6. SMS-Coastal first action is to create an instance of the class "Simrun", hence all the green blocks in the figures represent a series of procedures performed by that class methods, which are, in the order they appear, to:

- Remove previous simulation external forcing data and all contents from “res” and “exe” folders for each model level.
- Create or verify “Operations” directory and its inside structure, as well as SMS-Coastal log file.
- Write “Model.dat” file in “data” folder of each level based on the information about forecast range and iteration step time, “Tree.dat” in “exe” of the first level, copy “Nomfich.dat” to that folder, and then run MOHID executable.

- Check if the cycle has finished correctly and if so, copy forecast outputs to "Operations". Simulation success is determined by the content of MOHID log run file and the availability of restart files generated for the next day cycle.

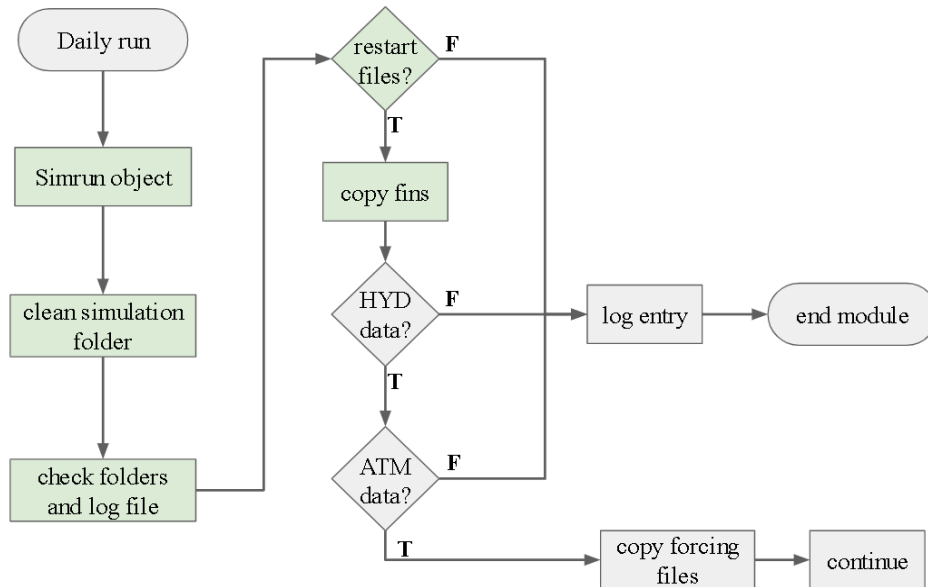


Figure 4.5 - Daily run cycle operations part 1.

In 4.5, “HYD data” and “ATM data” blocks denote the instant when SMS -oastal searches respectively for external oceanic and atmospheric data in “FORC” directory, given the information passed in "HYDSRC" and "ATMSRC" keywords. For each of the sources in one input, the algorithm chooses to use the file that has the most recent data, giving priority to the order listed by the user. Soon after that, there is an adjustment of the forecast days range, as shown by the first block of 4.6, by selecting the minimum value between the one entered in “TRANGE” and the maximum for each of the external forcing files chosen.

SMS-Coastal entries a record in its log file inside "Operations" folder before the daily module ends. It writes the information about whether the simulation succeeded or not. That entry consists of the simulation date, the success or error code plus the time and date of the registration, e.g. “2019-05-12 ERR03 2019-05-13_00:12”, which means that the simulation of may 12th failed at 00:12 of the next day. 4.5 shows the codes used by the management system. Additionally, all “log entry” blocks in figures 4.5 and 4.6, in which

this operation takes place, SMS-Coastal attempts to send a notice if an email is defined in “MAILTO” keyword, which may also contain MOHID execution logs attached.

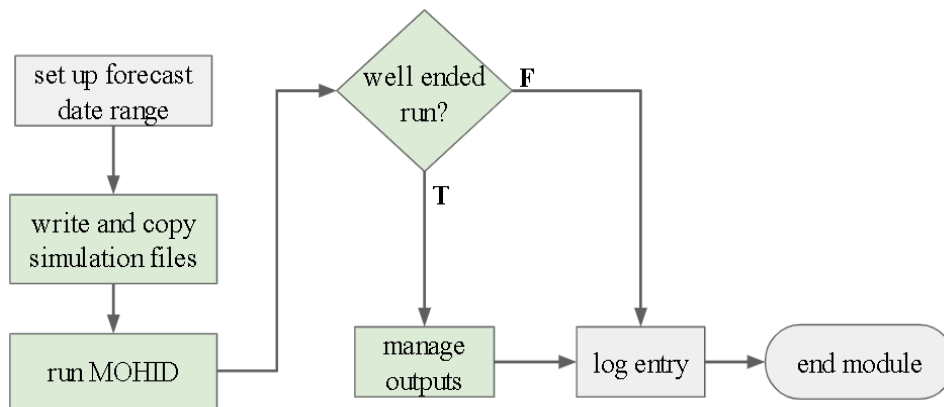


Figure 4.6 - Daily run cycle operations part 2.

Table 4.5 - Simulation layer log codes.

Code	Description
1	simulation success
ERR01	restart files not found
ERR02	external ocean data not found
ERR03	external atmospheric data not found
ERR04	MOHID log not found
ERR05	simulation stopped unexpectedly
ERR06	failed to generate restart files to the next run
WARN01	used outdated external ocean and atmospheric data
WARN02	used outdated external ocean data
WARN03	used outdated atmospheric data

4.3.2 Weekly run cycle

The processes sequence performed in the weekly simulation cycles has some similarities in relation to the daily run, as can be seen in the diagrams in figures 4.7 to 4.9. As

mentioned earlier, the weekly cycle also instantiates an object of the class “Simrun” to have access to its methods. However, there is no need to search for restart files before external forcing data check (4.7), considering that in weekly run a new solution cycle starts from initial conditions determined by the user in the model data files inside “data” folders of each level.

It is also noted in 4.8 that in weekly run, shortened to “WR”, SMS-Coastal commands the execution of more than one simulation, which are the stages of the model startup. Except for the last element in the keyword “TRANGE”, which is the value of days range for the forecast simulation, the number of stages is determined by the quantity of components in that input. Therefore, before the beginning of simulation cycles, hindcast dates ranges are calculated based on the value indicated by each “TRANGE” element. For every successfully completed stage the tool attempts to send a notice to the e-mail defined in “MAILTO” and, in case of a failure of one of them, the whole execution is aborted.

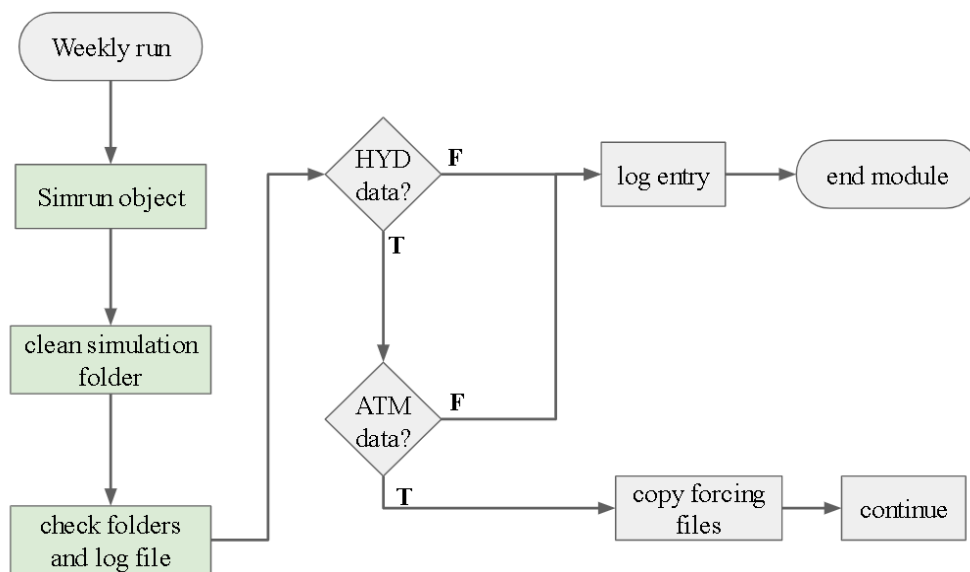


Figure 4.7 - Weekly run cycle operations part 1.

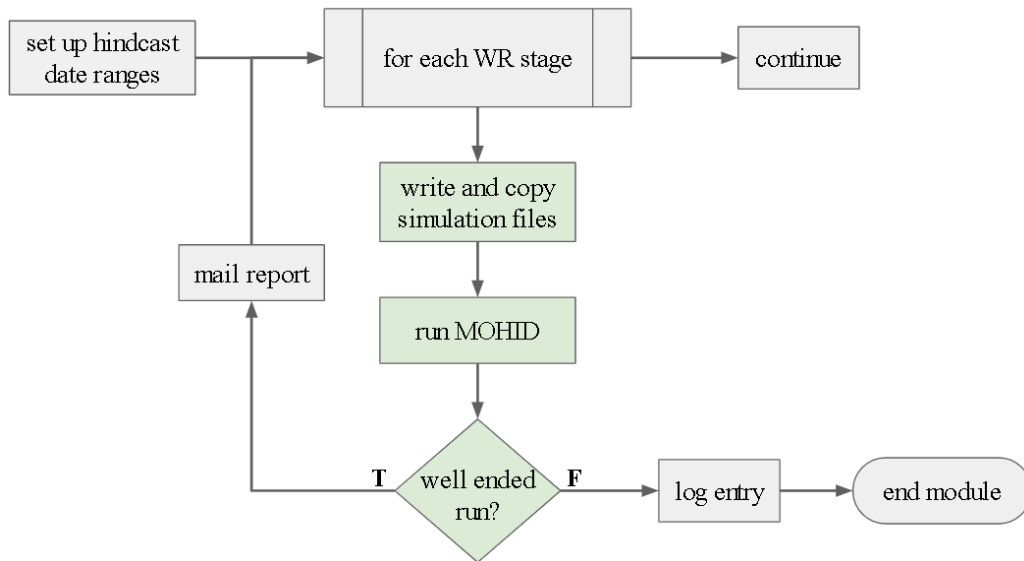


Figure 4.8 - Weekly run cycle operations part 2.

Considering that in a weekly run a hindcast of several days is performed (3.3), the execution of all stages may take more than one day to complete. This means that the last stage restart files can be produced with a delay that will make it impossible to use them in the daily simulation cycle. For this reason, the program generates an additional simulation stage (4.9), with the same data and configuration of a forecast run, but with an adjustment in days range, spanning from the date in which the previous stage has finished to the next daily cycle, in a way that the fin files should be available before the next daily cycle begins.

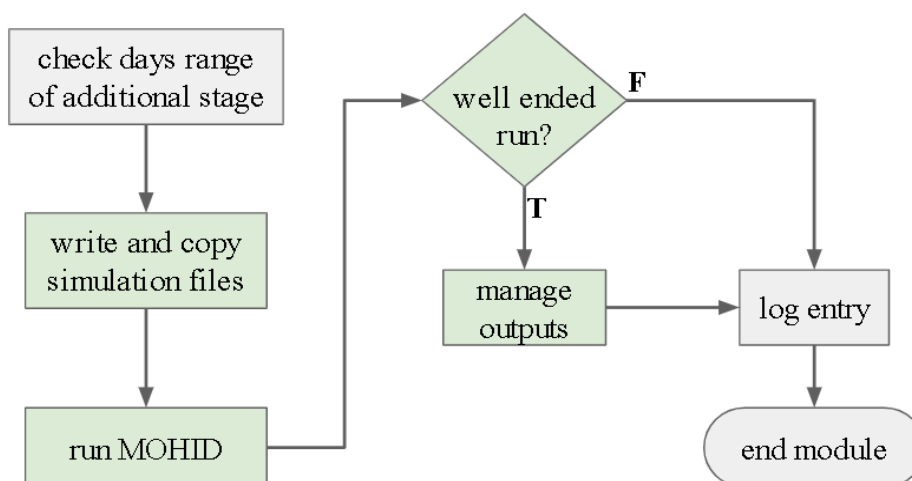


Figure 4.9 - Weekly run cycle operations part 3.

4.3.3 Forcing layer

The forcing layer has the function of coordinating the execution of the specific operations of each source entered in the keywords “HYDSRC” and “ATMSRC”, developing actions in two phases as shown in 4.3. In the first one, the list of oceanic or atmospheric data sources is scanned according to the diagram in the 4.10. The specific processing module for each source is executed considering the order inputted in each keyword. For a single data type, only one of the specific processes needs to be successful for the first stage to end, that is, SMS-Coastal will process the first data found for the listed sources. It will then returns to the simulation operation course (4.4) a list of the sources from which it was not necessary to try downloading. If none of the specific operations finish successfully, the first stage will return an empty list and the error corresponding to the lack of data will be identified in the simulation layer, as explained in item 4.3.1.

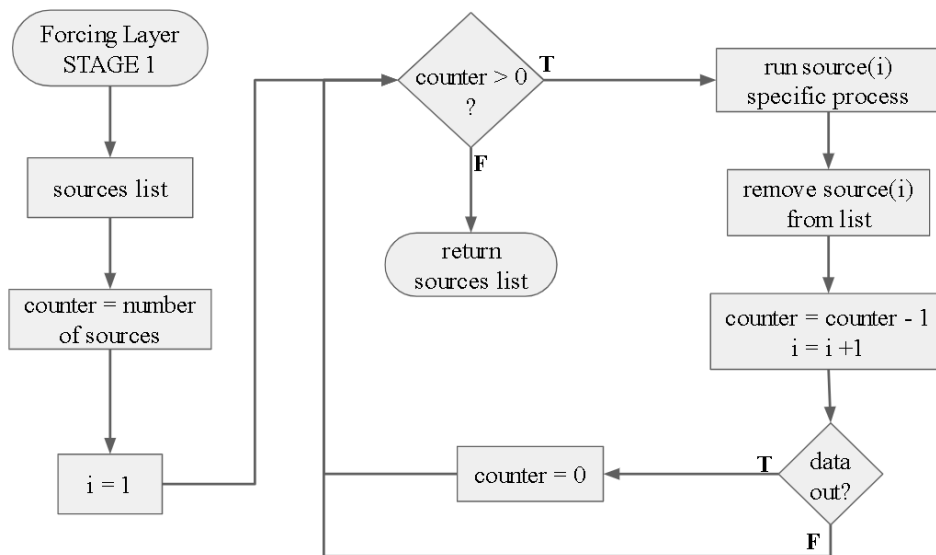


Figure 4.10 - Forcing layer first stage.

The second stage of the forcing layer (4.11) is launched after the execution of a daily run cycle and of any outputs formatting operation when applicable. Simpler than the first, it receives the list of not downloaded oceanic and atmospheric sources from the first stage and downloads the remaining data. In addition, as there is no need to perform conversion and interpolation operations, SMS-Coastal performs an update in the inputs that each specific module receives to disable these features. Both phases are executed if a source list

has no elements, but each one will be finalized before entering in their respective repetition cycle.

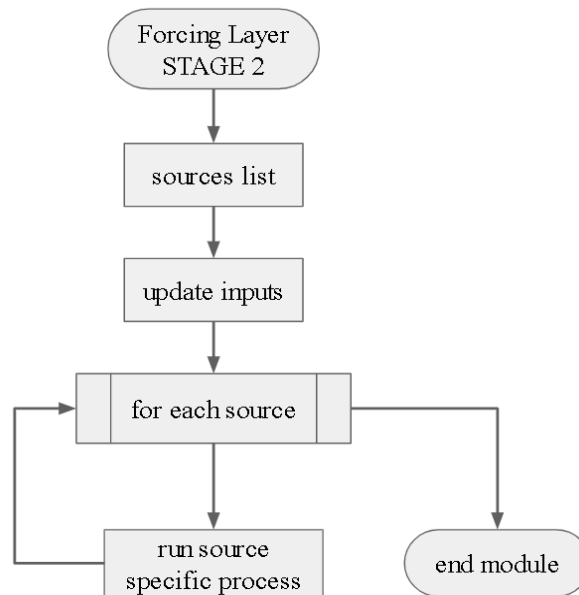


Figure 4.11 - Forcing layer second stage.

Due to the particularities of each external data source, it was necessary to build a module with the set of specific operations for each one. However, these modules have a similar processing logic, which can be delimited by the following items:

1. Check structure

Each module checks folders and files structure for the respective source, that will be located inside "FORC" in the project directory. When necessary, SMS-Coastal creates the set of folders that will be used in the subsequent processes, which are the folder for download, conversion, interpolation, simulation data storage and for downloaded files backup.

2. Data cleaning and backup

SMS-Coastal removes the outputs from the previous run, namely from the conversion and interpolation folders. For sources whose provider has long-term databases, the files in the download folder are also deleted, otherwise they are automatically moved to the backup folder. Within the simulation data storage, SMS-Coastal maintains a file database of one week for HDF5 formats and one month for time series.

3. Wait data availability

SMS-Coastal pauses if launched before the availability of data from the sources selected in "HYDSRC" or "ATMSRC". Each supplier has a specific schedule for uploading their data, and that is why this information is included in each source specific module.

4. Download

The download operation has the most specific processes for each source. In SOMA's case data from Mercator and Skiron are used. For the first one, it is necessary to have the "motuclient" Python module installed so that it can be called by the program to extract the information from the CMEMS database. As for Skiron data, they are downloaded from an FTP repository, by using built-in Python libraries.

5. Conversion

The conversion is carried out to transform the downloaded files into the HDF format, which is the native format of MOHID. SMS-Coastal conducts a conversion operation using the Convert2Hdf5 support tool for Mercator data in NetCDF files. As for Skiron GRIB files, it uses the "gdal" external module to read the datasets and then writes them directly to an HDF output using the methods available in "h5py" module.

6. Interpolation

After the conversion, the outputs go through the interpolation process done again by the Convert2Hdf5 tool, but controlled by the each specific source module. Then, the interpolated files are copied to another folder with a name pattern of "HYDSRC_LVi_YYMMDD.hdf5" for oceanic data and "ATMSRC_LVi_YYMMDD.hdf5" for atmospheric, where "i" is the number of the level to which the data was interpolated, "YY" the year in two digits, "MM" the month and "DD" the day, that is, the processing date of the datasets recorded in that file. In the simulation layer, the forcing data is searched according to the date recorded in the file name, which is copied to "General Data" folder changing the name to the standard specified in item 4.1

If the forcing format is changed to time series, by attributing "1" to "HYDTS" or "ATMTS" keywords, the program do not go through conversion and interpolation steps and writes the datasets information, obtained directly from the downloads reading, in a "dat"

file. The simulation files name pattern is changed to “HYDSRC_YYMMDD.dat” for oceanic data and “ATMSRC_YYMMDD.dat” for atmospheric.

4.3.4 Formatting outputs

At the end of a successful daily simulation cycle, if the user had defined “1” for the keyword “FMT”, SMS-Coastal initiates the results formatting module, which will generate the files to populate the local database. The implementing logic of this module algorithm is expressed by the diagram of 4.12.

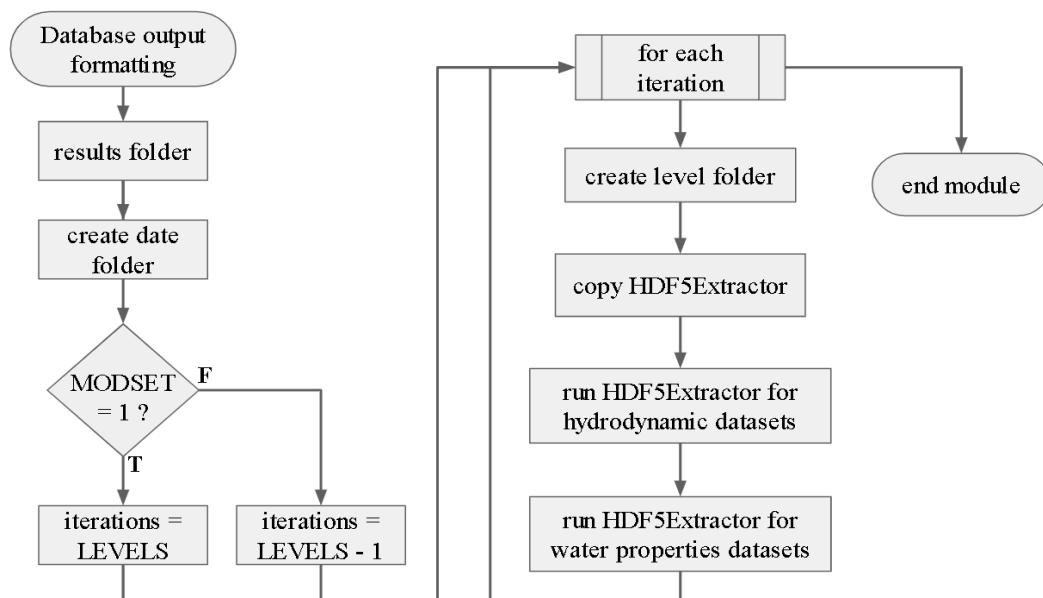


Figure 4.12 - Simulation output formatting module operations for model database.

One of the inputs of the formatting module is the folder in “Operations” directory that holds the daily run results to be reshaped. Then, the first thing it does is to create the date folder, inside the module standard output directory also located in “Operations”.

The number of iterations is based on the information given in “MODSET” and “LEVELS” keywords. If the first level of the model is configured just to generate the necessary hydrodynamics for child models, this module will process only the outputs from the second level and further ones.

Next, within the repetition cycle of 4.12, the program creates a folder for the current level inside of the date one and copies HDF5Extractor to it. Then SMS-Coastal executes the MOHID supporting tool for gathering a group of datasets of the same output time and write them in a single HDF5 file, doing this until there is data for only the first day of forecast. This is done firstly for the hydrodynamic file and then to the water properties one.

In the specific case of SOMA operationalization, it is also possible to convert the forecast results to the NetCDF format, which are one of the products of OCASO project. The algorithm of the module responsible for this operation, represented in the diagram of the 4.13, is activated by assigning the value “1” to the keyword “PDE” in the SMS-Coastal initialization file. This module also uses as input the folder containing the daily run outputs and excludes from the conversion the files of first level, if “MODSET” is equals to “2”.

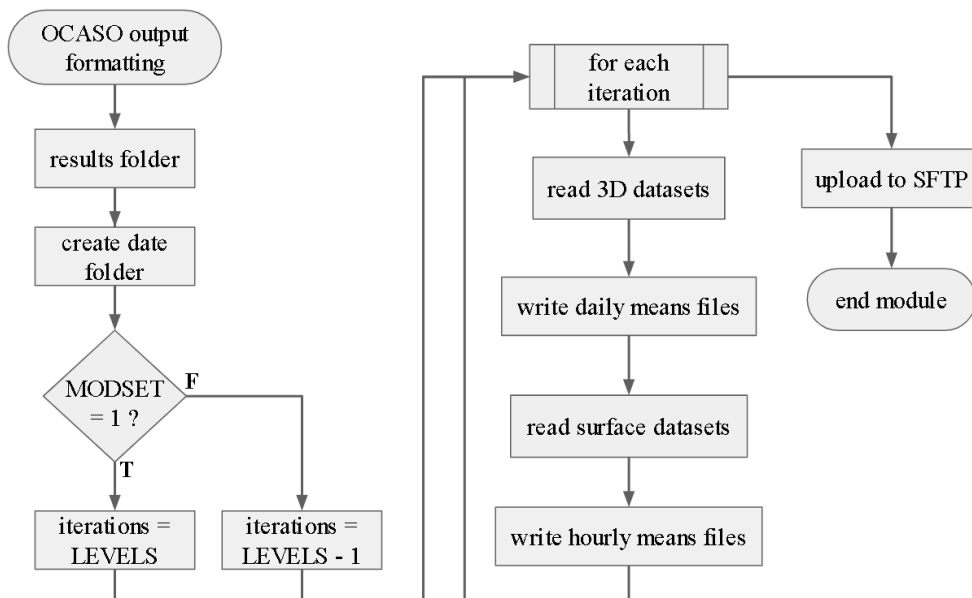


Figure 4.13 - Simulation output formatting module operations for OCASO project.

After the setting up, OCASO formatting module iterates through the outputs of each level. In the repetition cycle, it reads the datasets from hydrodynamic and water properties HDF5 files for the whole bathymetry, 3D datasets in 4.13. Then, it calculates the daily means and write one NetCDF file for each forecasted day. Secondly, it reads bathymetry top layer datasets only, from surface HDF5 files, calculates its hourly means and writes one NetCDF for each forecasted hour.

Each type of data sets, surface and three-dimensional, had to be handled in a unique way. For this reason, an auxiliary class was created to instantiate a tool-object in each iteration of the 4.13, which contains methods suitable to both types and others specific to deal with each one. As the diagram in the figure is a simplification of the algorithm, that class architecture can be consulted in the codes of the Appendices section. Lastly, when all files in the NetCDF format are generated, they are uploaded to an SFTP repository, so that they can be accessed by other collaborators of the OCASO project.

4.4 Failures statistics

SMS-Coastal first version was launched on July 7th 2019 and since then kept SOMA operational. Until June 30th 2020, the system coordinated 68 weekly runs and 371 daily runs, totaling 439 executions of the simulation layer. The simulations are running on a server computer, within Windows 10 Enterprise 64-bit operating system, with availability of 10 out of the 20 cores of the Intel Xeon Gold 6138 processor, 9.76 GB of RAM and an exclusive 450 GB data storage space. 4.6 shows the failures computed in the program log file for those executions and 4.7 the proportion of these failures related to total runs.

Table 4.6 - Failures in simulation layer.

Failures	Successful runs	Weekly run	Daily run	Total failures
Code errors	Yes	3	8	11
Manual emergency stops	Yes	2	7	9
Executions terminated by MOHID	No	8	11	19
Insufficient virtual memory	No	3	4	7
Restart files not found	No	0	2	2
Totals		16	32	48

Table 4.7 - Simulation layer failures proportion.

Failures	% failures	% total runs	% unsuccessful runs
Code errors	22.9 %	2.51%	0.00 %
Manual emergency stops	18.7 %	2.05 %	0.00 %
Executions terminated by MOHID	39.6 %	4.33 %	4.33 %
Insufficient virtual memory	14.6 %	1.59 %	1.59 %

Restart files not found	4.2 %	0.46 %	0.46 %
Totals	100 %	10.9 %	6.38 %

As for the forcing layer, SMS-Coastal did a total of 834 executions, so that it was 434 runs of the Mercator's specific module and 400 of Skiron's. 4.8 shows the failures of these modules that the management system was able to point out in the log files of each source.

Table 4.8 - Failures in forcing layer.

Failures	Mercator	Skiron	Total Failures	% failures	% total runs
Code errors	11	9	20	0	0
Download error	82	37	119	0	0
Interpolation error	4	11	15	0	0
Reading files error	0	4	4	0	0
Totals	97	61	158	100 %	0

The failures evaluated in the last two tables correspond to:

- Code error: SMS-Coastal crash caused by poor programming of a new module or in a code update.
- Manual emergency stops: execution manually aborted by the user.
- Executions terminated by MOHID: SMS-Coastal recognizes that the MOHID executable unsuccessfully ended, due to model instability, lack of external forces, or anything else presented in MOHID execution log.
- Insufficient virtual memory: specific case in which MOHID does not find enough space in the computer's memory to continue its execution.
- Restart files not found: SMS-Coastal was unable to prompt a daily run cycle, in simulation layer, due to the lack of restart files.
- Download error: failure encountered during specific download processes for each external source of oceanic and atmospheric data.

- Interpolation error: failure computed by Convert2Hdf5 supporting tool when performing an interpolation process of external forcing data.
- Reading files error: failure caused when opening external force file in read mode.

4.5 SOMA Outputs

At the end of each daily simulation cycle, SMS-Coastal copies the files containing the forecast data to the “RES” folder within “Operations”, becoming this way available to be viewed and analyzed. The following figures, constructed by MOHID Studio software from Action Modulers, illustrate some of the data available in that folder that were obtained during SOMA management by SMS-Coastal. In all of them the region with the highest resolution, which is the model level 3, is delimited by a rectangle and the black arrows indicates velocity vectors.

4.14 shows the map containing the predicted water velocity at the surface on July 18th 2020. For the same day, 4.15 and 4.16 show the temperature forecast for the surface and at 100 meters deep respectively. As a way of comparison, 4.17 contains the velocity, 4.18 and 4.19 the temperature information, but for June 19th of the same year. It is noticeable through the figures the change in water behavior between the indicated months. In June it is possible to observe a periodic phenomenon that occurs on the Portugal west coast and extends to the Algarve, which is the resurgence of deep and cold waters from the bottom of the ocean to the surface, also known as upwelling (Relvas, 2002).

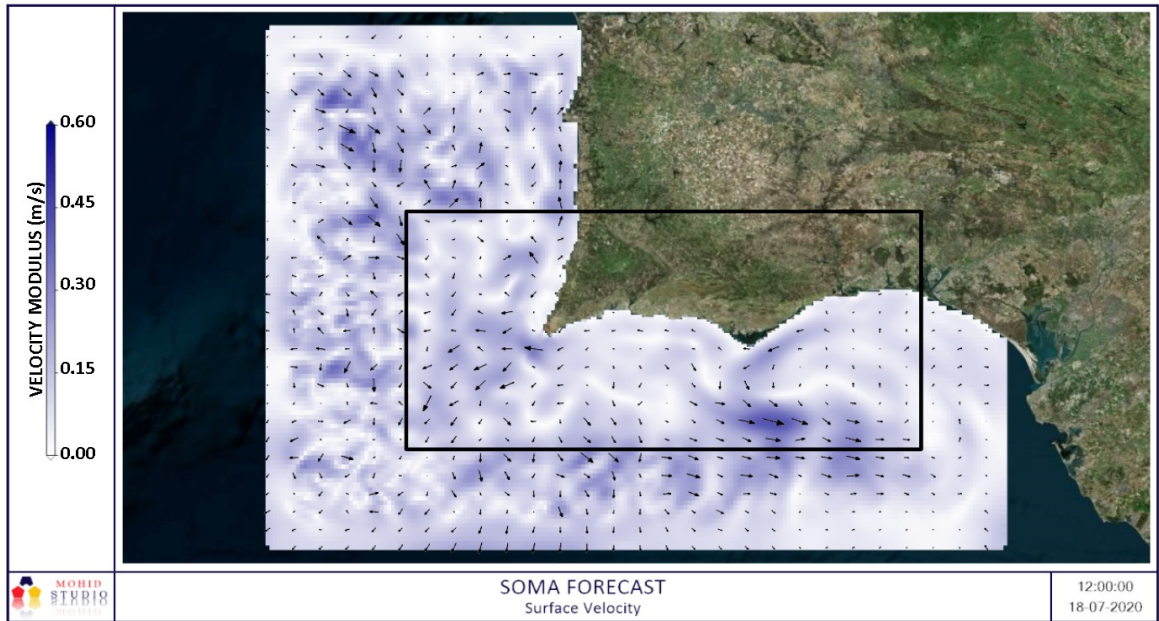


Figure 4.14 - SOMA surface velocity forecast in July 18th.

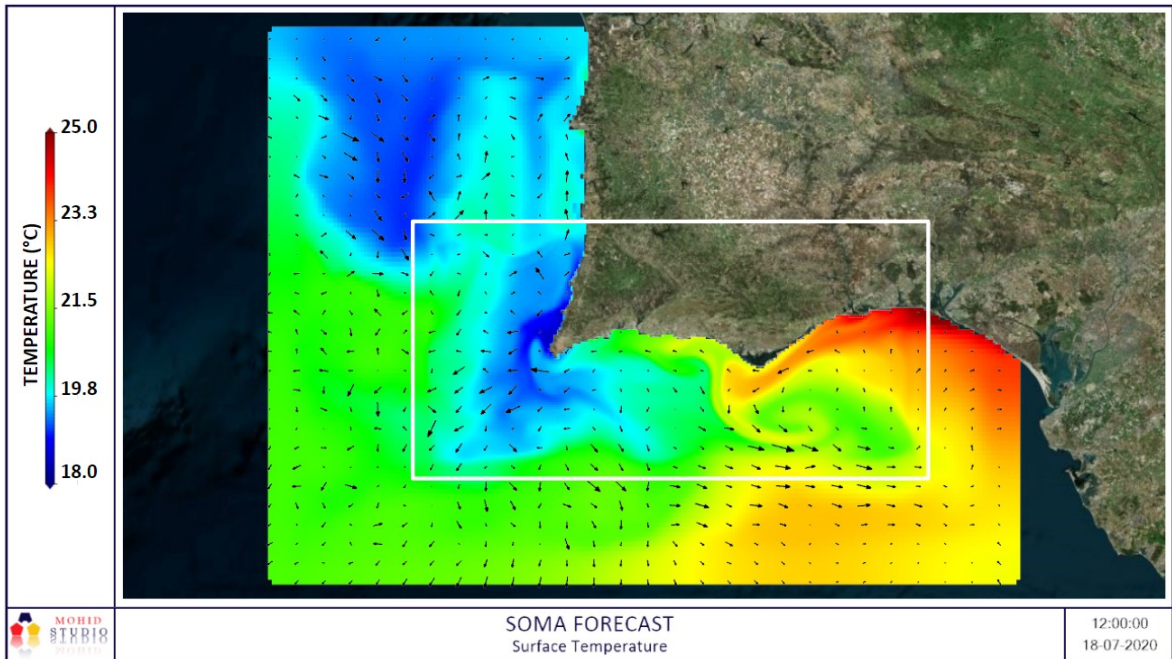


Figure 4.15 - SOMA surface temperature forecast in July 18th.

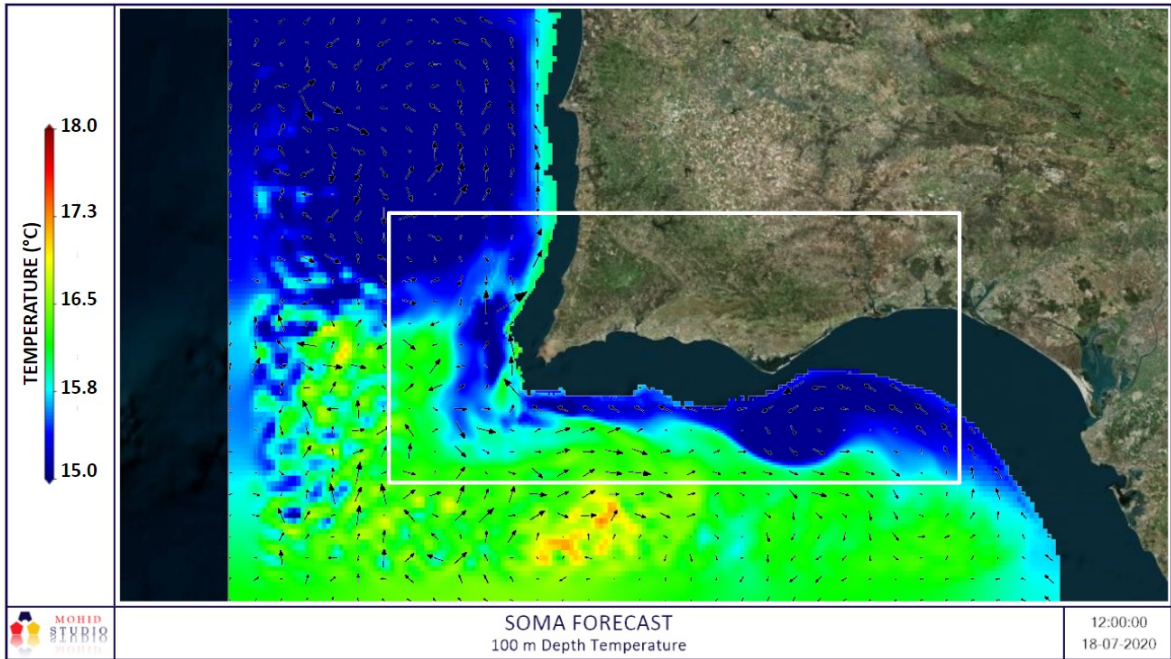


Figure 4.16 - SOMA 100 meters depth temperature forecast in July 18th.

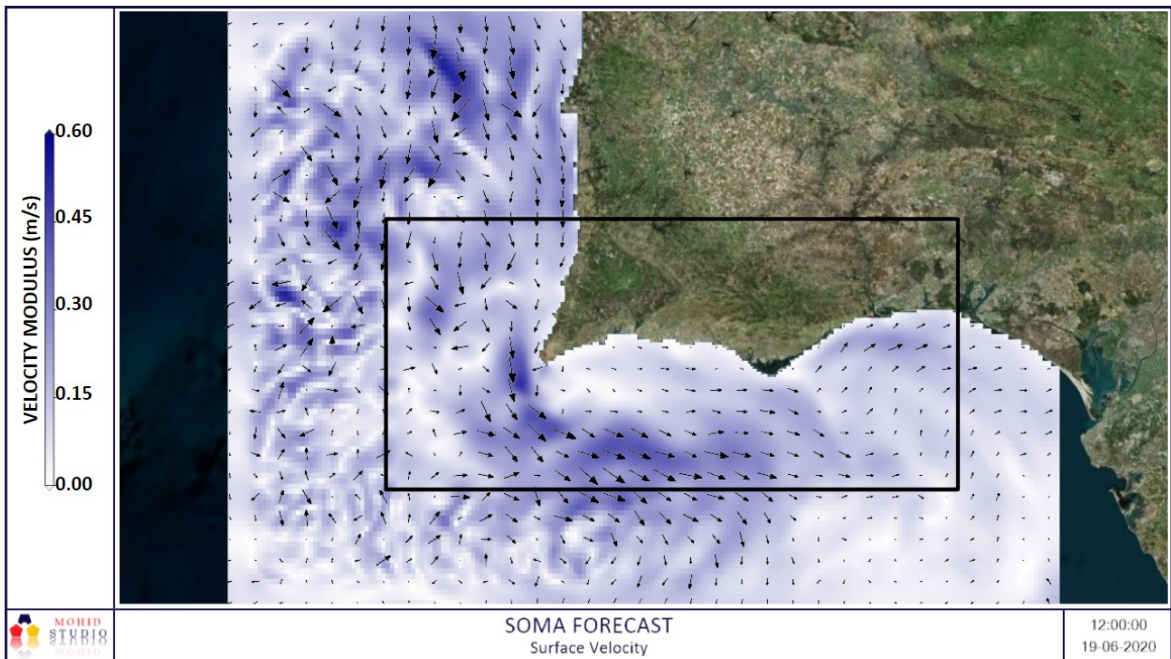


Figure 4.17 - SOMA surface velocity forecast in June 19th.

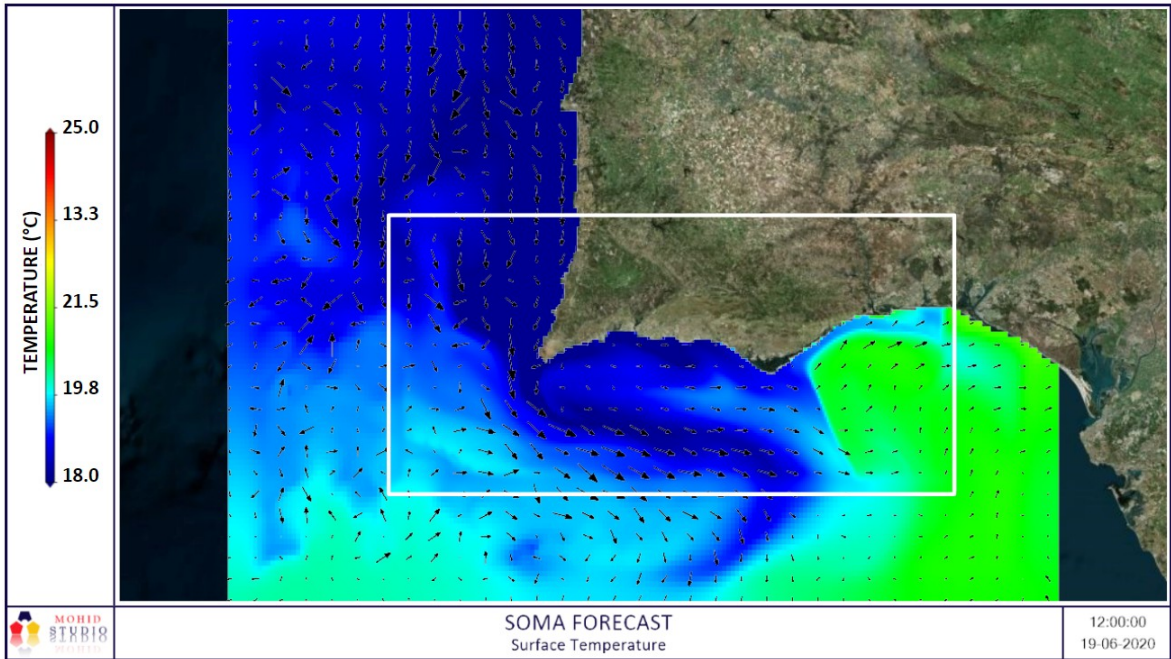


Figure 4.18 - SOMA surface temperature forecast in June 19th.

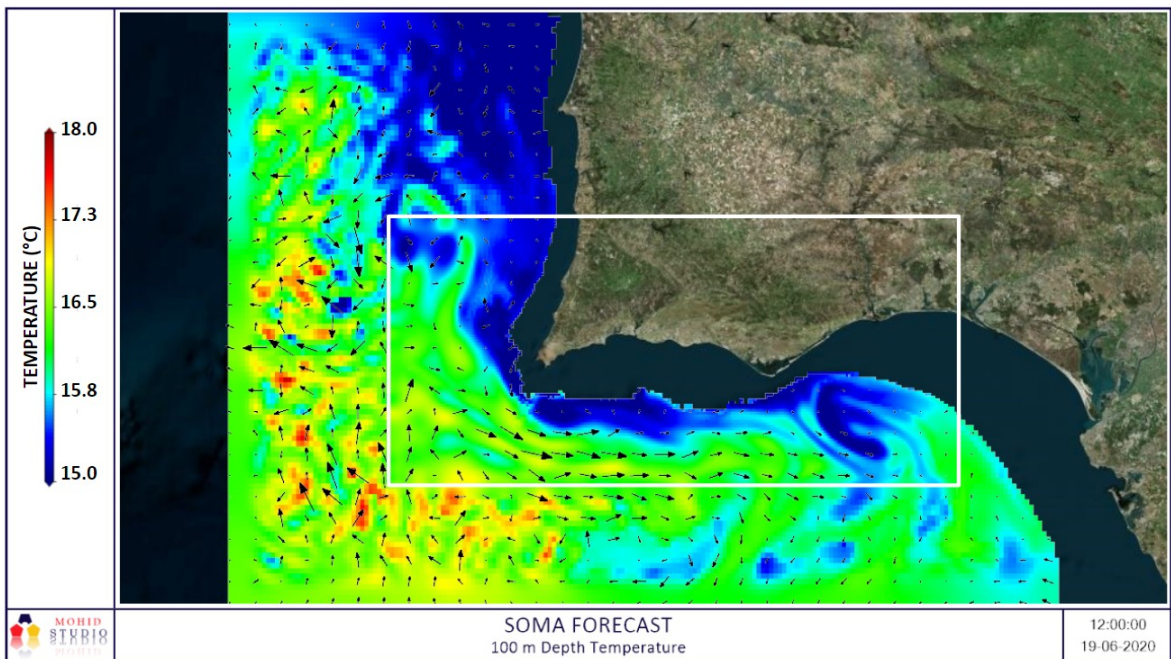


Figure 4.19 - SOMA 100 meters depth temperature forecast in June 19th.

5 DISCUSSION

Since the launch of SMS-Coastal first version for the operationalization of SOMA system, several improvements have been implemented in its code, new modules have been added and others discarded. In this way, new versions of the management system were released as updates were made, so that each one is the compilation of the set of modules that form it into a single executable file to be used on computers with Windows operating system. That file incorporates all necessary modules and libraries so that it can run independently of a Python installation. Nevertheless, an interpreter of this programming language is essential when downloading information from CMEMS, since to retrieve Mercator files the module “motuclient” must be called inside a Python environment. It is also important to note that SMS-Coastal does not have a self awakening mechanism, thus it is necessary to schedule its executable as one of the Windows tasks to make it fully automatic.

The imposed structure of files and folders must be respected so that the generic code identifies the correct computational paths during simulation handling. Therefore, in order to run a simulation operation the model must be set as shown by the project structure in 4.1, its data files must be inserted in each level “data” folder, the required inputs must be written in the initial file “ini.dat” and run SMS-Coastal application. Some missing directories will be generated by the management system in their absence, such as “Operations” of each simulation cycle, “FORC” and their contents, plus “res” and “exe” for each model level. Furthermore, since SMS-Coastal manipulates external forcing and bathymetry files, they have standard names which must be used in the instructions written in the model data files, otherwise MOHID will not be able to find them.

As for the input files, the user must be aware of which keywords to use and what values to attribute them. One of the SMS-Coastal first tasks is to check them all and interrupt the execution if something incorrect is found. After the keywords are defined for a model they will hardly be changed when the continuous forecast cycles begin. Inside SMS-Coastal’s program code those inputs are read as a Python dictionary, making the order they are typed in “init.dat” not relevant.

One of the major problems of the continuous simulation cycles is the availability of local storage space for the outputs. In a few days of execution a very large amount of data can be easily generated. In SOMA specific case, which has three levels, a four-day forecast occupies around 8.25 GB on disk, neglecting Mercator and Skiron interpolated data files used in that run. For that reason, SMS-Coastal has automatic data removal processes, since it was observed that low space disk can affect itself performance, making daily and/or weekly runs longer to complete, or even be interrupted. Therefore, in each simulation folder, SMS-Coastal maintains a maximum of 20 days results in a local database inside "Operations" and for each source of oceanic and atmospheric data, a week of interpolated HDF5 files or 30 days time series recorded in dat files.

At some moments during the execution of the code, SMS-Coastal splits into more than one thread to launch simultaneous processes. In the simulation layer, this happens only on the scheduled day to start a new solution cycle of the model. This is necessary because it is not possible to execute a daily run sequentially to a weekly run without interrupting the daily forecasts provision, considering that the latter may take more than a day to complete. Therefore, as the weekly and daily simulations are handled concomitantly, they were placed in separate folders inside the project directory. That is also quite useful to avoid conflict when they have to use the same forcing data files. Despite all this, weekly run still causes a reduction in the performance of the forecast simulation, which in the case of SOMA, MOHID execution logs indicated an increase of more than three hours, in most cases, in the simulation elapsed time. Obtaining external oceanic and atmospheric data are independent processes, so in the first stage of the forcing layer they are also carried out in different threads, so that simulation cycles may start as soon as possible. SMS-Coastal division into multiple threads, carried out by the Python “threading” module, however, did not have any efficiency analysis, so it is still possible to implement other more sophisticated programming techniques and even use other tools available for that language to optimize the management system in this aspect.

For each source available to be inputted in the keywords “HYDSRC” and “ATMSRC”, a specific module was built to process its data. Thus, SMS-Coastal has a library that will be constantly expanded as new sources are demanded for other coastal models. Hence, its forcing layer basically coordinates the moment when these modules are triggered in the

code operations sequence. For that reason, it was not required to generate its activity log, instead the procedures status of each specific module are individually registered in each source log file.

As indicated in the previous chapter, the forcing layer was separated into two stages, the first of one with the possibility to run more than one task at the same time. The second, on the other hand, only executes data download from unprocessed sources at the previous stage and the specific modules are no longer required to be run simultaneously, as the daily simulation cycle and other outputs conversion operations will be already completed. The objective here is to build a local database of forcing files that may be used in the simulation cycles. In this way, if a source processing fails, the next one will already have part of the data available to avoid delaying the start of the simulation layer, especially on the weekly run days when the amount of data corresponds the ranges defined in the keyword "TRANGE". This shows that the sources listed by the user serve only as a redundancy of each other and in the end only one will be used.

The same procedures are performed in the forcing sources specific modules, even though in different ways. Final interpolated or time series files always contain data from the first day of hindcast to the last day of forecast, calculated based on the inputs of "TRANGE" and the operation date. In this manner, the forcing files will always be prepared to run a restart of the model, regardless of the week day. Then, all the user have to do in order to change the weekly run cycle day is to update the value given to "RESTART" keyword. As for storing, for sources that has no long-term data availability, the module will automatically backup the downloads in a local database. Those, in turn, must be managed by the user so the disk will not run out of memory, erasing or moving unnecessary files to another storage device. External forcing files ready for the simulation layer are kept for one week to cover the possible scenario in which the processing of all sources fails, therefore, even if it uses outdated oceanic and/or atmospheric data, the model remains operational. In this case the forecast days range are adjusted according to the file used.

The SMS-Coastal version presented in this work has two optional processes for formatting daily cycle simulations results. The first one divides the outputs of one day into smaller HDF5 files, one for each datasets instant, until there is information for the first 24 hours of the forecast. The next run will give the subsequent 24 hours information and so

on. This procedure is activated by the keyword “FMT” and was designed to generate files that will become the model’s historical database. Due to that, the program does not have a cleaning operation for those files, and just like for the forcing data backup, they should be manually transferred to another disk.

The second formatting process essentially consists in converting the simulation results from HDF5 to NetCDF, which, however, was programmed exclusively for the SOMA system results. The module built to accomplish this task meets a demand from OCASO project by making Algarve’s coast forecast data available to the other partners. For this reason, the output files of this module have writing standards and a very well defined structure. Because of that, the code is not generic in this point and shall not be used for any other model. Nevertheless, a valuable knowledge about NetCDF handling with Python was developed and it might be used as basis to build other applications.

Some of the SMS-Coastal modules could be used independently, that is, outside the simulations context. This is the case of the intrinsic operations to obtain forcing data and to format HDF5 MOHID output files. Thus, from the management system itself new features are being generated and they will be available to be selected only by changing the value of the keyword “OPTYPE”. The other inputs will be specific for each operation type, so that it will be required to set up the initial file “init.dat” for each one. The functions are: (1) coastal models forecast simulations management; (2) forcing data processing according to the library available in SMS-Coastal; (3) MOHID outputs formatting for model’s database; (4) MOHID outputs conversion for OCASO project; and (5) a more generic application of the last, MOHID outputs conversion to NetCDF for model’s database. Except for (5), the main objective of this work was operation (1) integrated with the others, so that this is already programmed and is currently being used to manage SOMA system forecast cycles.

Regarding the operationalization of the SOMA system, SMS-Coastal has been managing its forecast simulations for over one year. During this period, the management system had been updated many times, as new tasks were added and programming failures were identified. The tables in the section 4.4 indicates the problems encountered while handling the model runs, from the launch of the first version until the last day of June 2020. Most of the failures caused by poor coding happened in the first days of operation and they were used exactly as a basis to make the necessary corrections in the algorithm, which in turn

was only implemented after exhaustive tests to prevent simulation cycles interruption. In the forcing layer, errors during download represented more than 75% of the total failures. Little can be done about them except to rerun the program to make a new download attempt. This is mainly caused by lost of connection between the computer and the supplier's repository, but also by a possible delay in data availability.

Finally, in the simulation layer, the failure caused by insufficient virtual memory is a specific case of the stops made by the MOHID system, which however was counted separated because it represents a situation that is related to SMS-Coastal. At the end of an execution, the prompt window remains open so the user is able to consult the operations history. Thus, it was observed that when those processes are still open, even having more than enough disk space to run the model, the computer is left with low random-access memory (RAM). This type of failure is even more likely to happen when there is also two simulations running simultaneously. Therefore, SMS-Coastal is not scheduled to be automatically launched on SOMA restart day, instead, all remaining windows are manually closed and the computer is rebooted. After this procedure has been implemented no simulation has failed for that reason yet.

6 CONCLUSION

In this work a program as developed, written with the object-oriented programming language Python, to manage forecast simulations of operational hydrodynamic coastal models, the SMS-Coastal. The sequence of most basic operations to be developed by it in a management process was based on the design of a similar one with the same purpose found in the literature. However, the program was conceived to run models only built within the MOHID modelling system environment, which is the one responsible to perform the numerical solution of the governing equations and to carry out other process such as interpolation and conversion forcing files to HDF5 format.

SMS-Coastal was developed using SOMA's operationalization as a background, which is a MOHID based and validated high resolution hydrodynamic model of the Algarve coast. Therefore, it conducts two distinct simulation cycles: the daily runs, which correspond to the continuous forecast simulations, and the weekly runs, which starts the model from the most recent conditions given by CMEMS to generate new restart files with less deteriorated initial conditions. The model became the means to test the SMS-Coastal versions and consequently became the first one to be controlled by it. This is also the reason why modules were created within the management system's forcing library to process Mercator and Skiron data. The operationalization of SOMA remains active and producing daily forecast data since the launch of the program first version in July 2019.

Simulation management basically consists in coordinating the execution of different operations and file handling. For that reason, the simulation process carried out by MOHID system, as well as the generic folder structure and file naming presented in this work, for which SOMA had to be adapted, were the means to identify the routine patterns that were used to build SMS-Coastal's algorithm in a way that it would be suitable for any other model. The method of reading keywords was adopted so that the user can specify the parameters to run each model. In this way, it is expected that other projects can be put in operational mode by the tool so that their data may support activities developed within the blue economy.

The use of an object-oriented programming language helped to simplify the program's code, since the same methods could be used more than once in different situations,

avoiding extensive writing of the same parts of the code. Moreover, modules that develop tasks outside the context of simulations, such as formatting results or forcing data processing, have given rise to other features that shall be independently accessed. However, in the way it was designed and together with the imposed folder structure, SMS-Coastal major limitation is that it does not yet allow the operationalization of models that have more than one sub-domain defined for the same level. In addition, when the code splits its execution into more than one thread to perform simultaneous tasks, the printed messages of each running module are shuffled in the prompt window, becoming confusing to the user.

Despite these and other minor code problems that might be identified, the simulation management system presented in this work is prepared to make coastal models operational. Even so, SMS-Coastal's construction is a continuous process and so updates and improvements shall be regularly implemented. Because of that, the options for running independent operations such as results formatting and forcing data processing are already being programmed and soon will be integrated into the program. Besides that, and considering that Python has a vast built-in library and several other external tools, which offer a wide range of resources, also as future works, it is recommended to optimize the parallelization of tasks and even to build a graphical interface for better viewing the messages. Furthermore, SMS-Coastal still has room to grow with new modules for other forcing sources and even with the development of new features.

BIBLIOGRAPHY

- Argo Program*. (2020, May 28). Argo - Part of the Integrated Global Observation Strategy. <http://www.argo.ucsd.edu/>
- Bailey, K., Steinberg, C., Davies, C., Galibert, G., Hidas, M., McManus, M. A., Murphy, T., Newton, J., Roughan, M., & Schaeffer, A. (2019). Coastal Mooring Observing Networks and Their Data Products: Recommendations for the Next Decade. *Frontiers in Marine Science*, 6. <https://doi.org/10.3389/fmars.2019.00180>
- Bari, A. (2017). Our Oceans and the Blue Economy: Opportunities and Challenges. *Procedia Engineering*, 194, 5–11. <https://doi.org/10.1016/j.proeng.2017.08.109>
- Bell, M., Lefèbvre, M., Le Traon, P.-Y., Smith, N., & Wilmer-Becker, K. (2009). GODAE: The Global Ocean Data Assimilation Experiment. *Oceanography*, 22(3), 14–21. <https://doi.org/10.5670/oceanog.2009.62>
- Böning, C. W., & Semtner, A. J. (2001). Chapter 2.2 High-resolution modelling of the thermohaline and wind-driven circulation. In *International Geophysics* (Vol. 77, pp. 59–XII). Elsevier. [https://doi.org/10.1016/S0074-6142\(01\)80112-1](https://doi.org/10.1016/S0074-6142(01)80112-1)
- Braunschweig, F., Leitao, P. C., Fernandes, L., Pina, P., & Neves, R. J. J. (2004). The object-oriented design of the integrated water modelling system MOHID. In *Developments in Water Science* (Vol. 55, pp. 1079–1090). Elsevier. [https://doi.org/10.1016/S0167-5648\(04\)80126-6](https://doi.org/10.1016/S0167-5648(04)80126-6)
- Burgess, M. G., Clemence, M., McDermott, G. R., Costello, C., & Gaines, S. D. (2018). Five rules for pragmatic blue growth. *Marine Policy*, 87, 331–339. <https://doi.org/10.1016/j.marpol.2016.12.005>
- Capet, A., Fernández, V., She, J., Dabrowski, T., Umgiesser, G., Staneva, J., Mészáros, L., Campuzano, F., Ursella, L., Nolan, G., & El Serafy, G. (2020). Operational Modeling Capacity in European Seas—An EuroGOOS Perspective and Recommendations for Improvement. *Frontiers in Marine Science*, 7, 129. <https://doi.org/10.3389/fmars.2020.00129>
- Casanova-Arenillas, S., Rodríguez-Tovar, F. J., & Martínez-Ruiz, F. (2020). Applied ichnology in sedimentary geology: Python scripts as a method to automatize ichnofabric analysis in marine core images. *Computers & Geosciences*, 136, 104407. <https://doi.org/10.1016/j.cageo.2020.104407>

- Chambers, C., Ungar, D., Chang, B.-W., & Hölzle, U. (1991). Parents are shared parts of objects: Inheritance and encapsulation in SELF. *Lisp and Symbolic Computation*, 4(3), 207–222. <https://doi.org/10.1007/BF01806106>
- Dombrowsky, E. (2011). Overview Global Operational Oceanography Systems. In A. Schiller & G. B. Brassington (Eds.), *Operational Oceanography in the 21st Century* (pp. 397–411). Springer Netherlands. https://doi.org/10.1007/978-94-007-0332-2_16
- Greenberg, D. A., Dupont, F., Lyard, F. H., Lynch, D. R., & Werner, F. E. (2007). Resolution issues in numerical models of oceanic and coastal circulation. *Continental Shelf Research*, 27(9), 1317–1343. <https://doi.org/10.1016/j.csr.2007.01.023>
- Griffies, S. M. (2006). Some Ocean Model Fundamentals. In E. P. Chassignet & J. Verron (Eds.), *Ocean Weather Forecasting* (pp. 19–73). Springer-Verlag. https://doi.org/10.1007/1-4020-4028-8_2
- Howard, B. C. (2018). Blue growth: Stakeholder perspectives. *Marine Policy*, 87, 375–377. <https://doi.org/10.1016/j.marpol.2017.11.002>
- IP, M. (2006). *List of internal metrics for the MERSEA-GODAE Global Ocean: Specification for implementation*.
- Janeiro, J., Neves, A., Martins, F., & Relvas, P. (2017). Integrating technologies for oil spill response in the SW Iberian coast. *Journal of Marine Systems*, 173, 31–42. <https://doi.org/10.1016/j.jmarsys.2017.04.005>
- Janeiro, J., Zacharioudaki, A., Sarhadi, E., Neves, A., & Martins, F. (2014). Enhancing the management response to oil spills in the Tuscany Archipelago through operational modelling. *Marine Pollution Bulletin*, 85(2), 574–589. <https://doi.org/10.1016/j.marpolbul.2014.03.021>
- Kämpf, J. (2009). *Ocean Modelling for Beginners*. Springer Berlin Heidelberg. <https://doi.org/10.1007/978-3-642-00820-7>
- Knox, S., Meier, P., Yoon, J., & Harou, J. J. (2018). A python framework for multi-agent simulation of networked resource systems. *Environmental Modelling & Software*, 103, 16–28. <https://doi.org/10.1016/j.envsoft.2018.01.019>
- Le Traon, P.-Y. (2011). Satellites and Operational Oceanography. In A. Schiller & G. B. Brassington (Eds.), *Operational Oceanography in the 21st Century* (pp. 29–54). Springer Netherlands. https://doi.org/10.1007/978-94-007-0332-2_2

- Lee, K.-H., Noh, J., & Khim, J. S. (2020). The Blue Economy and the United Nations' sustainable development goals: Challenges and opportunities. *Environment International*, *137*, 105528. <https://doi.org/10.1016/j.envint.2020.105528>
- Leitão, P., Coelho, H., Santos, A., & Neves, R. (2005). Modelling the main features of the Algarve coastal circulation during July 2004: A downscaling approach. *Journal of Atmospheric & Ocean Science*, *10*(4), 421–462. <https://doi.org/10.1080/17417530601127704>
- Liblik, T., Karstensen, J., Testor, P., Alenius, P., Hayes, D., Ruiz, S., Heywood, K. J., Pouliquen, S., Mortier, L., & Mauri, E. (2016). Potential for an underwater glider component as part of the Global Ocean Observing System. *Methods in Oceanography*, *17*, 50–82. <https://doi.org/10.1016/j.mio.2016.05.001>
- Lorente, P., Aznar, R., Alvarez Fanjul, E., Pascual, Á., Toledano Lozano, C., Amo, A., Dabrowski, T., Levier, B., Reffray, G., Dalphinnet, A., & Aouf, L. (2019). *The NARVAL software toolbox in support of ocean model skill assessment at regional and coastal scales*.
- Malone, T., Davidson, M., DiGiacomo, P., Gonçalves, E., Knap, T., Muelbert, J., Parslow, J., Sweijd, N., Yanagai, T., & Yap, H. (2010). Climate Change, Sustainable Development and Coastal Ocean Information Needs. *Procedia Environmental Sciences*, *1*, 324–341. <https://doi.org/10.1016/j.proenv.2010.09.021>
- Malone, T., DiGiacomo, P. M., Gonçalves, E., Knap, A. H., Talaue-McManus, L., & de Mora, S. (2014). A global ocean observing system framework for sustainable development. *Marine Policy*, *43*, 262–272. <https://doi.org/10.1016/j.marpol.2013.06.008>
- Marta-Almeida, M., Ruiz-Villarreal, M., Otero, P., Cobas, M., Peliz, A., Nolasco, R., Cirano, M., & Pereira, J. (2011). OOFe: A Python engine for automating regional and coastal ocean forecasts☆. *Environmental Modelling & Software*, *26*(5), 680–682. <https://doi.org/10.1016/j.envsoft.2010.11.015>
- Martins, F. (1999). *Modelação matemática tridimensional de escoamentos costeiros e estuarinos usando uma abordagem de coordenada vertical genérica*. Universidade Técnica de Lisboa.

- Martins, F., Leitão, P., Silva, A., & Neves, R. (2001). 3D modelling in the Sado estuary using a new generic vertical discretization approach. *Oceanologica Acta*, 24, 51–62. [https://doi.org/10.1016/S0399-1784\(01\)00092-5](https://doi.org/10.1016/S0399-1784(01)00092-5)
- Martins, F., Neves, R. J., & Leitão, P. C. (1998). *A three-dimensional hydrodynamic model with generic vertical coordinate*.
- Martins, J. P. (2012). *Programação em Python: Introdução a Programação Utilizando Múltiplos Paradigmas*.
- Medcliv. (2020, September 22). Coastal HF Radar. <http://medcliv.es/en/instrumentos/radar-costero-hf/>
- Miranda, R., Braunschweig, F., Leitao, P., Neves, R., Martins, F., & Santos, A. (2000). MOHID 2000-A coastal integrated object oriented model. *WIT Transactions on Ecology and the Environment*, 40.
- Mittal, T., & Delbridge, B. (2019). Detection of the 2012 Havre submarine eruption plume using Argo floats and its implications for ocean dynamics. *Earth and Planetary Science Letters*, 511, 105–116. <https://doi.org/10.1016/j.epsl.2019.01.035>
- Neves, R. (2007). NUMERICAL MODELS AS DECISION SUPPORT TOOLS IN COASTAL AREAS. In I. E. Gonenc, V. G. Koutitonsky, B. Rashleigh, R. B. Ambrose, & J. P. Wolflin (Eds.), *Assessment of the Fate and Effects of Toxic Agents on Water Resources* (pp. 171–195). Springer Netherlands. https://doi.org/10.1007/978-1-4020-5528-7_8
- Oke, P. R., Sakov, P., Cahill, M. L., Dunn, J. R., Fiedler, R., Griffin, D. A., Mansbridge, J. V., Ridgway, K. R., & Schiller, A. (2013). Towards a dynamically balanced eddy-resolving ocean reanalysis: BRAN3. *Ocean Modelling*, 67, 52–70. <https://doi.org/10.1016/j.ocemod.2013.03.008>
- Oliphant, T. E. (2006). *Guide to NumPy*. USA: Trelgol Publishing.
- Oliphant, T. E. (2007). Python for Scientific Computing. *Computing in Science & Engineering*, 9(3), 10–20. <https://doi.org/10.1109/MCSE.2007.58>
- Perez-Schofield, B. G., & Ortin, F. (2019). A didactic object-oriented, prototype-based visual programming environment. *Science of Computer Programming*, 176, 1–13. <https://doi.org/10.1016/j.scico.2019.02.004>

- Pouliquen, S. (2006). In-Situ Observations: Operational Systems and Data Management. In E. P. Chassignet & J. Verron (Eds.), *Ocean Weather Forecasting* (pp. 207–227). Springer-Verlag. https://doi.org/10.1007/1-4020-4028-8_8
- Prandle, D. (2000). Introduction. *Coastal Engineering*, 41(1–3), 3–12. [https://doi.org/10.1016/S0378-3839\(00\)00024-7](https://doi.org/10.1016/S0378-3839(00)00024-7)
- Ravichandran, M. (2011). In-Situ Ocean Observing System. In A. Schiller & G. B. Brassington (Eds.), *Operational Oceanography in the 21st Century* (pp. 55–90). Springer Netherlands. https://doi.org/10.1007/978-94-007-0332-2_3
- Reeve, K. A., Boebel, O., Strass, V., Kanzow, T., & Gerdes, R. (2019). Horizontal circulation and volume transports in the Weddell Gyre derived from Argo float data. *Progress in Oceanography*, 175, 263–283. <https://doi.org/10.1016/j.pocean.2019.04.006>
- Relvas, P. (2002). Mesoscale patterns in the Cape São Vicente (Iberian Peninsula) upwelling region. *Journal of Geophysical Research*, 107(C10), 3164. <https://doi.org/10.1029/2000JC000456>
- Robinson, I. S. (2006). Satellite Measurements for Operational Ocean Models. In E. P. Chassignet & J. Verron (Eds.), *Ocean Weather Forecasting* (pp. 147–189). Springer-Verlag. https://doi.org/10.1007/1-4020-4028-8_6
- Robinson, I. S. (2010a). Introduction. In I. S. Robinson, *Discovering the Ocean from Space* (pp. 1–6). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68322-3_1
- Robinson, I. S. (2010b). The methods of satellite oceanography. In I. S. Robinson, *Discovering the Ocean from Space* (pp. 7–67). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68322-3_2
- Roemmich, D., Alford, M. H., Claustre, H., Johnson, K., King, B., Moum, J., Oke, P., Owens, W. B., Pouliquen, S., Purkey, S., Scanderbeg, M., Suga, T., Wijffels, S., Zilberman, N., Bakker, D., Baringer, M., Belbeoch, M., Bittig, H. C., Boss, E., ... Yasuda, I. (2019). On the Future of Argo: A Global, Full-Depth, Multi-Disciplinary Array. *Frontiers in Marine Science*, 6. <https://doi.org/10.3389/fmars.2019.00439>
- Sáenz, J., Zubillaga, J., & Fernández, J. (2002). Geophysical data analysis using Python. *Computers & Geosciences*, 28(4), 457–465. [https://doi.org/10.1016/S0098-3004\(01\)00086-3](https://doi.org/10.1016/S0098-3004(01)00086-3)

- Schiller, A. (2011). Ocean Forecasting in the 21st Century. In A. Schiller & G. B. Brassington (Eds.), *Operational Oceanography in the 21st Century* (pp. 3–26). Springer Netherlands. https://doi.org/10.1007/978-94-007-0332-2_1
- Send, U. (2006). In-Situ Observations: Platforms and Techniques. In E. P. Chassignet & J. Verron (Eds.), *Ocean Weather Forecasting* (pp. 191–206). Springer-Verlag. https://doi.org/10.1007/1-4020-4028-8_7
- She, J. (2015). *Analysis on research priorities for European operational oceanography*.
- She, J., Allen, I., Buch, E., Crise, A., Johannessen, J. A., Le Traon, P.-Y., Lips, U., Nolan, G., Pinardi, N., Reißmann, J. H., Siddorn, J., Stanev, E., & Wehde, H. (2016). Developing European operational oceanography for Blue Growth, climate change adaptation and mitigation, and ecosystem-based management. *Ocean Science*, *12*(4), 953–976. <https://doi.org/10.5194/os-12-953-2016>
- Smith, N. R. (2000). The Global Ocean Data Assimilation Experiment. *Advances in Space Research*, *25*(5), 1089–1098. [https://doi.org/10.1016/S0273-1177\(99\)00868-6](https://doi.org/10.1016/S0273-1177(99)00868-6)
- Summerhayes, C. (2002). Technical tools for regional seas management: The role of the Global Ocean Observing System (GOOS). *Ocean & Coastal Management*, *45*(11–12), 777–796. [https://doi.org/10.1016/S0964-5691\(02\)00106-0](https://doi.org/10.1016/S0964-5691(02)00106-0)
- Tomlinson, J. E., Arnott, J. H., & Harou, J. J. (2020). A water resource simulator in Python. *Environmental Modelling & Software*, *126*, 104635. <https://doi.org/10.1016/j.envsoft.2020.104635>
- Trancoso, A. R., Braunschweig, F., Chambel Leitão, P., Obermann, M., & Neves, R. (2009). An advanced modelling tool for simulating complex river systems. *Science of The Total Environment*, *407*(8), 3004–3016. <https://doi.org/10.1016/j.scitotenv.2009.01.015>
- Ungar, D., Chambers, C., Chang, B.-W., & Hölzle, U. (1991). Organizing programs without classes. *Lisp and Symbolic Computation*, *4*(3), 223–242. <https://doi.org/10.1007/BF01806107>
- van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, *13*(2), 22–30. <https://doi.org/10.1109/MCSE.2011.37>
- Van Rossum, G., & Drake Jr, F. L. (1995). *Python tutorial* (Vol. 620). Amsterdam: Centrum voor Wiskunde en Informatica.

- Versteeg, H. K., & Malalasekera, W. (2007). *An introduction to computational fluid dynamics: The finite volume method*. Pearson Education.
- Volk, J. M., & Turner, M. A. (2019). PRMS-Python: A Python framework for programmatic PRMS modeling and access to its data structures. *Environmental Modelling & Software*, *114*, 152–165. <https://doi.org/10.1016/j.envsoft.2019.01.006>
- von Schuckmann, K., Le Traon, P.-Y., Alvarez-Fanjul, E., Axell, L., Balmaseda, M., Breivik, L.-A., Brewin, R. J. W., Bricaud, C., Drevillon, M., Drillet, Y., Dubois, C., Embury, O., Etienne, H., Sotillo, M. G., Garric, G., Gasparin, F., Gutknecht, E., Guinehut, S., Hernandez, F., ... Verbrugge, N. (2016). The Copernicus Marine Environment Monitoring Service Ocean State Report. *Journal of Operational Oceanography*, *9*(sup2), s235–s320. <https://doi.org/10.1080/1755876X.2016.1273446>
- Vos, K., Splinter, K. D., Harley, M. D., Simmons, J. A., & Turner, I. L. (2019). CoastSat: A Google Earth Engine-enabled Python toolkit to extract shorelines from publicly available satellite imagery. *Environmental Modelling & Software*, *122*, 104528. <https://doi.org/10.1016/j.envsoft.2019.104528>
- White, J. T., Fienen, M. N., & Doherty, J. E. (2016). A python framework for environmental model uncertainty analysis. *Environmental Modelling & Software*, *85*, 217–228. <https://doi.org/10.1016/j.envsoft.2016.08.017>
- Wohlstadter, M., Shoaib, L., Posey, J., Welsh, J., & Fishman, J. (2016). A Python toolkit for visualizing greenhouse gas emissions at sub-county scales. *Environmental Modelling & Software*, *83*, 237–244. <https://doi.org/10.1016/j.envsoft.2016.05.016>
- Wu, G., Lu, Z., Luo, Z., Shang, J., Sun, C., & Zhu, Y. (2019). Experimental Analysis of a Novel Adaptively Counter-Rotating Wave Energy Converter for Powering Drifters. *Journal of Marine Science and Engineering*, *7*(6), 171. <https://doi.org/10.3390/jmse7060171>

APPENDICES

This section presents the codes of the modules built in the Python programming language, which together form the simulation management system for coastal hydrodynamic models, the SMS-Coastal. Each sub-chapter corresponds to one of the modules.

Appendix A – Operation selector

Module to select the operation to perform.

```
import os
from subprocess import run
from datetime import datetime

import inputsread
import sim_control
from forcontrol import forcontrol

rootdir = os.getcwd()
if not os.path.isfile(rootdir + "\\init.dat"):
    print("ERROR\n\ninit file not found\n")
    run("pause", shell=True)
    raise SystemExit

init = inputsread.initread(("OPTYPE", "OPDATE"), rootdir)
optype = init.get("OPTYPE")
opdate = init.get("OPDATE")

if not opdate:
    opdate = datetime.today().date()
else:
    try:
        opdate = datetime.strptime(opdate, "%Y %m %d").date()
    except ValueError:
        print("ERROR\n\nNot a valid date format\n")
        run("pause", shell=True)
        raise SystemExit

if optype == "1":
    sim_control.simcontrol(rootdir, opdate)
elif optype == "2":
    print("FORCING testing")
elif optype == "3":
    print("FMT develop")
elif optype == "4":
    print("PDE develop")
elif optype == "5":
```

```

print("NetCDF project")
else:
    print("ERROR\n\nSelect a valid operation\n")

os.chdir(rootdir)
os.chdir("../..\\")
run("pause", shell=True)

```

Appendix B – Initial data file reader

Module to read initial data file “init.dat” and return a dictionary with the inputs.

```

import os

def initread(keywords, initpath):
    initdat = initpath + "\\init.dat"
    lines = tuple([line for line in open(initdat, "r")])

    init_keys = tuple([line[:line.find(":")].strip() for line in lines])
    init_vlue = tuple([line[line.find(":") + 1:].strip() for line in lines])

    vlues = []
    for key in keywords:
        vlue = None

        if key in init_keys:
            vlue = init_vlue[init_keys.index(key)]

        if vlue == ":":
            vlue = None

        vlues.append(vlue)

    return dict(zip(keywords, vlues))

def initsim(initpath):
    inputs = initread(("MODSET", "LEVELS", "RESTART", "GMTREF", "TRANGE", "FMT", "PDE",
"MAILTO", "HYDSRC", "ATMSRC",
"LATLIM", "LONLIM", "HYDTS", "ATMTS", "TSLOC"), initpath)
    out_keys = {}

    if None in (inputs.get("MODSET"), inputs.get("LEVELS"), inputs.get("RESTART"),
inputs.get("GMTREF"),
inputs.get("TRANGE")):
        return

    levels = inputs.get("LEVELS")
    restart = inputs.get("RESTART")
    gmtref = inputs.get("GMTREF")
    trange = inputs.get("TRANGE")

    try:

```

```

levels = int(levels)
int(gmtref)
restart = int(restart)
trange = [int(val) for val in trange.split()]
except ValueError:
    return

if levels < 1 or 0 in trange:
    return

out_keys["LEVELS"] = levels
out_keys["RESTART"] = restart
out_keys["GMTREF"] = gmtref
out_keys["TRANGE"] = tuple(trange)

dtlvi = initread([f"DTLV{level + 1}" for level in range(levels)], initpath)
dtrun = []

for level in range(levels):
    vals = dtlvi.get(f"DTLV{level + 1}")
    if not vals:
        return

    vals = vals.split()

    try:
        vals = [int(val) for val in vals]
    except (ValueError, TypeError):
        return

    if len(vals) != len(trange):
        return

    dtrun.append(vals)

dtrun = tuple(zip(*dtrun))
out_keys["DTRUN"] = dtrun

val = inputs.get("MODSET")
if not val or val not in ("1", "2"):
    return
out_keys["MODSET"] = val

vals = "FMT", "PDE", "HYDTS", "ATMTS"
for key in vals:
    val = inputs.get(key)
    if val not in ("0", "1"):
        out_keys[key] = "0"
    else:
        out_keys[key] = val

out_keys["MAILTO"] = inputs.get("MAILTO")

hydsrc = inputs.get("HYDSRC")
atmsrc = inputs.get("ATMSRC")

if not hydsrc and not atmsrc:
    out_keys["HYDSRC"] = ()

```

```

out_keys["ATMSRC"] = ()
out_keys["GRID"] = None
out_keys["TSLOC"] = None
return out_keys

out_keys["HYDSRC"] = tuple(hydsrc.split())
out_keys["ATMSRC"] = tuple(atmsrc.split())
lat = inputs.get("LATLIM")
lon = inputs.get("LONLIM")
tsl = inputs.get("TSLOC")

try:
    lat = [float(val.replace(",",".")) for val in lat.split()]
    lon = [float(val.replace(",",".")) for val in lon.split()]
except (ValueError, TypeError, AttributeError):
    return

if len(lat) != 2 or len(lon) != 2:
    return

lon.sort()
lat.sort()
out_keys["GRID"] = tuple(lon + lat)

if "1" in (out_keys.get("ATMTS"), out_keys.get("HYDTS")):
    try:
        tsl = [float(val.replace(",",".")) for val in tsl.split()]
    except (ValueError, TypeError, AttributeError):
        return

    if len(tsl) != 2:
        return

    out_keys["TSLOC"] = tuple(tsl)
else:
    out_keys["TSLOC"] = None

return out_keys

```

Appendix C – Common operations

Module containing methods to write in log files and send reporting e-mails.

```

import os
import smtplib

from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders

def mailreport(email_send, reportsubject, body, filename):
    if not email_send or email_send == " or "@" not in email_send:

```

```

return
try:
    email_user = '*****@ualg.pt'
    email_password = '*****'
    server = smtplib.SMTP('smtp.office365.com',587)

    msg = MIMEMultipart()
    msg['From'] = email_user
    msg['To'] = email_send
    msg['Subject'] = reportsubject

    msg.attach(MIMEText(body,'plain'))

    for file in filename:
        if not os.path.isfile(file):
            filename = ()

    for file in filename:
        attachment = open(file,'rb')

        part = MIMEBase('application','octet-stream')
        part.set_payload(attachment.read())
        encoders.encode_base64(part)
        part.add_header('Content-Disposition', "attachment; filename= "+file)
        msg.attach(part)

    text = msg.as_string()

    server.starttls()
    server.login(email_user,email_password)

    server.sendmail(email_user,email_send,text)
    server.quit()

except Exception as err:
    print('\n' + ' Failed to send report e-mail '.center(65, '-') + f'\n{err}\n')

def logentry(logfile, entry, mode="a"):
    with open(logfile, mode) as log:
        log.write(entry)

```

Appendix D – Simulation control

Module to control the simulation operation.

```

import os
import concurrent.futures
from inputsread import initsim
from threading import Thread
from time import sleep
from glob import glob

```

```

from forcmercator import mercator
from forc_skiron import skiron
from forcnam import nam
from forcgfs import gfs
from sim_weekly import weeklyrun
from sim_daily import dailyrun
from post_fmtres import fmtres
from post_pde_netcdf import convertpde

def first_forc(srcs, outdir, opdate, inputs):
    srclslib = {"Mercator": mercator, "Skiron": skiron, "NAM": nam, "GFS": gfs}
    redund = list(srcs)

    if "AVGWP" in redund:
        redund.remove("AVGWP")

    counter = len(redund)

    while counter > 0:
        status = srclslib.get(redund[0])(outdir, opdate, inputs)
        redund.remove(redund[0])
        counter -= 1

        if status == 1:
            counter = 0

    return redund

def scnd_forc(srcs, outdir, opdate, inputs):
    srclslib = {"Mercator": mercator, "Skiron": skiron, "NAM": nam, "GFS": gfs}
    for src in srcs:
        srclslib.get(src)(outdir, opdate, inputs)

def simcontrol(projdir, opdate):
    inputs = initsim(projdir)
    if not inputs:
        print("ERROR\n\nInvalid/missing inputs\n")
        return

    inputs["PROJDIR"] = projdir
    inputs["MOHID"] = projdir + "\\MOHID"

    inputs["FORCDIR"] = projdir + "\\FORC"
    if not os.path.isdir(inputs.get("FORCDIR")):
        os.mkdir(inputs.get("FORCDIR"))

    # ----- FORCING LAYER
    hyd_inputs = inputs.copy()
    hyd_inputs["HNDCAST"] = sum(inputs.get("TRANGE")[:-1])
    hyd_inputs["BKUP"] = "1"
    if inputs.get("HYDTS") == "0":
        hyd_inputs["HDFOUT"] = "1"
        hyd_inputs["TSOUT"] = "0"
    else:

```

```

hyd_inputs["HDFOUT"] = "0"
hyd_inputs["TSOUT"] = "1"

atm_inputs = hyd_inputs.copy()
if inputs.get("ATMTS") == "0":
    atm_inputs["HDFOUT"] = "1"
    atm_inputs["TSOUT"] = "0"
else:
    atm_inputs["HDFOUT"] = "0"
    atm_inputs["TSOUT"] = "1"

with concurrent.futures.ThreadPoolExecutor() as executor:
    hydpro = executor.submit(first_forc, inputs.get("HYDSRC"), inputs.get("FORCDIR"), opdate,
hyd_inputs)
    sleep(30)
    atmpro = executor.submit(first_forc, inputs.get("ATMSRC"), inputs.get("FORCDIR"), opdate,
atm_inputs)
    hydpost = hydpro.result()
    atmpost = atmpro.result()

atm_inputs["HDFOUT"] = "0"
atm_inputs["TSOUT"] = "0"
hyd_inputs["HDFOUT"] = "0"
hyd_inputs["TSOUT"] = "0"

# ----- SIMULATION LAYER
weekpro = Thread(target=weeklyrun, args=(opdate, inputs))
if opdate.weekday() == inputs.get("RESTART"):
    print("WR + DR")
    weekpro.start()
    sleep(30)

dailyrun(opdate, inputs)

# ----- POST OPERATIONS
resdir = glob(projdir + opdate.strftime("\\Sim_Daily\\Operations\\RES\\SIM%y%m%d_END*"))
resdir.sort()
try:
    resdir = resdir[-1]
except IndexError:
    resdir = None

if resdir and inputs.get("FMT") == "1":
    fmtres(resdir, opdate, projdir + "\\Sim_Daily\\Operations", inputs.get("LEVELS"),
inputs.get("MODSET"),
    inputs.get("MOHID"))

if resdir and inputs.get("PDE") == "1":
    convertpde(resdir, opdate, projdir + "\\Sim_Daily\\Operations", inputs.get("MODSET"),
inputs.get("LEVELS"))

# ----- REDUNDANCY
scond_forc(hydpost, inputs.get("FORCDIR"), opdate, hyd_inputs)
scond_forc(atmpost, inputs.get("FORCDIR"), opdate, atm_inputs)

if weekpro.is_alive():
    weekpro.join()

```


Appendix E – Weekly run cycle control

Module to control weekly run cycles.

```
import os
from datetime import datetime, timedelta
from math import ceil
from threading import Thread

from sim_runs import Simrun
from sim_forcing import select_source, check_range
from opcommon import mailreport

def weeklyrun(simdate, inputs):
    projdir = inputs.get("PROJDIR")
    projid = os.path.basename(projdir)
    simdir = projdir + "\\Sim_Weekly"

    print("-" * 80 + "\n" + "SIMULATION MANAGMENT MODULE".center(80) + "\n" + "-" * 80)
    print("PROJECT   :", projdir)
    print("DATE       :", simdate.strftime("%Y %m %d"))
    print("TYPE        : Weekly run")
    print("MODEL SET   :", inputs.get("MODSET"))
    print("LEVELS     :", inputs.get("LEVELS"))
    print("-"*80)

    # ----- LOAD SIMULATION OBJECT
    runsim = Simrun(projdir, inputs.get("LEVELS"), simdate, "Weekly", inputs.get("MAILTO"))
    print("Wiping simulation folder", end="\n\n")
    runsim.wipesim()
    print("Setting up environment", end="\n\n")
    runsim.chkdirs()
    runsim.logentry(str(simdate))

    # ----- LOAD EXTERNAL FORCINGS DATA
    print("Searching forcing data")
    hyd_status = select_source(inputs.get("HYDSRC"), inputs.get("FORCDIR"), simdir, simdate,
inputs.get("HYDTS"))
    print(hyd_status)
    if hyd_status[0] == "NOT_FOUND":
        print("-"*80 + "\nERROR - Hydrodynamic forcing data not found\n" + "-"*80)
        runsim.logentry(datetime.today().strftime(" ERR02 %Y-%m-%d_%H:%M\n"))
        body = simdate.strftime("%Y-%m-%d Hydrodynamic forcing data not found")
        mailreport(inputs.get("MAILTO"), projid + " Daily run ERROR", body, ())
        return

    atm_status = select_source(inputs.get("ATMSRC"), inputs.get("FORCDIR"), simdir, simdate,
inputs.get("ATMTS"))
    print(atm_status)
    if atm_status[0] == "NOT_FOUND":
        print("-"*80 + "\nERROR - Atmospheric forcing data not found\n" + "-"*80)
        runsim.logentry(datetime.today().strftime(" ERR03 %Y-%m-%d_%H:%M\n"))
        body = simdate.strftime("%Y-%m-%d Atmospheric forcing data not found")
        mailreport(inputs.get("MAILTO"), projid + " Daily run ERROR", body, ())
        return
```

```

print()

forctime = check_range(hyd_status, atm_status)
simtime = datetime(simdate.year, simdate.month, simdate.day) + timedelta(hours=forctime[1])

timerun = []
trange = list(inputs.get("TRANGE"))[:-1]

for _ in range(len(trange)):
    timerun.append(simtime - timedelta(sum(trange)))
    trange = trange[1:]

timerun.append(simtime)

# ----- RUN SIMULATION
threads = []
runid = 1

for stage in range(len(timerun)-1):
    runid += stage
    print("Running STAGE", runid)
    runsim.modeldat(runid, timerun[stage], timerun[stage + 1], inputs.get("DTRUN")[stage],
inputs.get("GMTREF"))
    status = runsim.execrun(runid)
    if status < 1:
        return
    print()

    subject = projid + " Weekly run STAGE " + str(runid) + " COMPLETED"
    body = simdate.strftime("%Y-%m-%d") + ' SIMULATION HDF5 READY'
    logs = simdir + "\\Operations\\LOGS\\Mohid_log_"
    logs = logs + f"run{runid}.txt", logs + f"err{runid}.txt"
    threads.append(Thread(target=mailreport, args=(inputs.get("MAILTO"), subject, body, logs)))
    threads[-1].start()

delta_time = ceil((datetime.today() - simtime).total_seconds() / (24 * 3600))

if delta_time <= 2:
    print("Running Interface")
    timerun = simtime, simtime + timedelta(delta_time)
    runid += 1

    runsim.modeldat(runid, timerun[0], timerun[1], inputs.get("DTRUN")[-1], inputs.get("GMTREF"))
    status = runsim.execrun(runid)
    if status < 1:
        return
    print()

    body = simdate.strftime("%Y-%m-%d") + ' SIMULATION HDF5 READY'
    logs = simdir + "\\Operations\\LOGS\\Mohid_log_"
    logs = logs + f"run{runid}.txt", logs + f"err{runid}.txt"
    mailreport(inputs.get("MAILTO"), projid + " Interface run COMPLETED", body, logs)

else:
    delta_time = 0

# ----- FINISH SIMULATION AND COPY FILES
if hyd_status[1] and atm_status[1] > 0:

```

```

    runsim.logentry(datetime.today().strftime(" WARN1 %Y-%m-%d_%H:%M\n")) # hyd and atm
forecast data
elif hyd_status[1] > 0:
    runsim.logentry(datetime.today().strftime(" WARN2 %Y-%m-%d_%H:%M\n")) # hyd forecast data
elif atm_status[1] > 0:
    runsim.logentry(datetime.today().strftime(" WARN3 %Y-%m-%d_%H:%M\n")) # atm forecast data
else:
    runsim.logentry(datetime.today().strftime(" 1 %Y-%m-%d_%H:%M\n"))

runsim.saveres()
runsim.savefin(runid, delta_time, inputs.get("MODSET"))

for thread in threads:
    thread.join()
print("-"*80)
print("WEEKLY RUN COMPLETED")
print("-"*80)

```

Appendix F – Daily run cycle control

Module to control daily run cycles.

```

import os
from datetime import datetime, timedelta

from sim_runs import Simrun
from sim_forcing import select_source, check_range
from opcommon import mailreport

def dailyrun(simdate, inputs):
    projdir = inputs.get("PROJDIR")
    projid = os.path.basename(projdir)
    simdir = projdir + "\\Sim_Daily"

    print("-" * 80 + "\n" + "SIMULATION MANAGMENT MODULE".center(80) + "\n" + "-" * 80)
    print("PROJECT   :", projdir)
    print("DATE      :", simdate.strftime("%Y %m %d"))
    print("TYPE      : Daily run")
    print("MODEL SET :", inputs.get("MODSET"))
    print("LEVELS   :", inputs.get("LEVELS"))
    print("-"*80)

    # ----- LOAD SIMULATION OBJECT
    runsim = Simrun(projdir, inputs.get("LEVELS"), simdate, "Daily", inputs.get("MAILTO"))
    print("Wiping simulation folder", end="\n\n")
    runsim.wipesim()
    print("Setting up environment", end="\n\n")
    runsim.chkdirs()
    runsim.logentry(str(simdate))

    print("Checking FINS")
    status = runsim.getfins()

```

```

if status < 1:
    return

# ----- LOAD EXTERNAL FORCINGS DATA
print("Searching forcing data")
hyd_status = select_source(inputs.get("HYDSRC"), inputs.get("FORCDIR"), simdir, simdate,
inputs.get("HYDTS"))
print(hyd_status)
if hyd_status[0] == "NOT_FOUND":
    print("-"*80 + "\nERROR - Hydrodynamic forcing data not found\n" + "-"*80)
    runsim.logentry(datetime.today().strftime(" ERR02 %Y-%m-%d_%H:%M\n"))
    body = simdate.strftime("%Y-%m-%d Hydrodynamic forcing data not found")
    mailreport(inputs.get("MAILTO"), projid + " Daily run ERROR", body, ())
    return

atm_status = select_source(inputs.get("ATMSRC"), inputs.get("FORCDIR"), simdir, simdate,
inputs.get("ATMTS"))
print(atm_status)
if atm_status[0] == "NOT_FOUND":
    print("-"*80 + "\nERROR - Atmospheric forcing data not found\n" + "-"*80)
    runsim.logentry(datetime.today().strftime(" ERR03 %Y-%m-%d_%H:%M\n"))
    body = simdate.strftime("%Y-%m-%d Atmospheric forcing data not found")
    mailreport(inputs.get("MAILTO"), projid + " Daily run ERROR", body, ())
    return
print()

forctime = check_range(hyd_status, atm_status)
simtime = datetime(simdate.year, simdate.month, simdate.day) + timedelta(hours=forctime[1])
frange = min(inputs.get("TRANGE")[-1], forctime[0])

# ----- RUN SIMULATION
print("Running Forecast")
runsim.modeldat(1, simtime, simtime + timedelta(frange), inputs.get("DTRUN")[-1],
inputs.get("GMTREF"))
status = runsim.execrun(1)
if status < 1:
    return

# ----- FINISH SIMULATION AND COPY FILES
if hyd_status[1] and atm_status[1] > 0:
    runsim.logentry(datetime.today().strftime(" WARN1 %Y-%m-%d_%H:%M\n")) # hyd and atm
forecast data
elif hyd_status[1] > 0:
    runsim.logentry(datetime.today().strftime(" WARN2 %Y-%m-%d_%H:%M\n")) # hyd forecast data
elif atm_status[1] > 0:
    runsim.logentry(datetime.today().strftime(" WARN3 %Y-%m-%d_%H:%M\n")) # atm forecast data
else:
    runsim.logentry(datetime.today().strftime(" 1 %Y-%m-%d_%H:%M\n"))

runsim.saveres()
runsim.savefin(1, 1, inputs.get("MODSET"))

print("-"*80)
print("DAILY RUN COMPLETED")
print("-"*80)

body = simdate.strftime("%Y-%m-%d") + ' SIMULATION HDF5 READY'

```

```
logs = simdir + "\\Operations\\LOGS\\Mohid_log_run1.txt", simdir + "\\Operations\\LOGS\\
Mohid_log_err1.txt"
mailreport(inputs.get("MAILTO"), projid + " Daily run COMPLETED", body, logs)
```

Appendix G – Simulation class

Class with all the common methods to any kind of simulation cycle.

```
import os
from glob import glob
from shutil import rmtree, copyfile, copytree
from subprocess import run
from datetime import datetime, timedelta

from opcommon import mailreport

class Simrun:
    def __init__(self, projdir, levels, simdate, simtype, mailto):
        self.projdir = projdir
        self.rootdir = projdir + "\\Sim_" + simtype
        self.levels = levels
        self.simdate = simdate
        self.simtype = simtype
        self.mailto = mailto

    def wipesim(self):
        boundaries = self.rootdir + "\\General Data\\Boundary Conditions\\*FORC*"
        initials = self.rootdir + "\\General Data\\Initial Conditions\\AVGWP*"
        logs = self.rootdir + "\\Operations\\LOGS\\Mohid*.txt"
        files = glob(boundaries) + glob(initials) + glob(logs)
        for file in files:
            os.unlink(file)

        lvpath = ""
        for level in range(self.levels):
            lvpath += "\\Level " + str(level + 1)
            if os.path.isdir(self.rootdir + lvpath + "\\res"):
                rmtree(self.rootdir + lvpath + "\\res")
            if os.path.isdir(self.rootdir + lvpath + "\\exe"):
                rmtree(self.rootdir + lvpath + "\\exe")
            os.mkdir(self.rootdir + lvpath + "\\res")
            os.mkdir(self.rootdir + lvpath + "\\exe")

    def chkdirs(self):
        if not os.path.isdir(self.rootdir + "\\Operations"):
            os.mkdir(self.rootdir + "\\Operations")
        if not os.path.isdir(self.rootdir + "\\Operations\\LOGS"):
            os.mkdir(self.rootdir + "\\Operations\\LOGS")
        if not os.path.isfile(self.rootdir + "\\Operations\\LOGS\\" + self.simtype + "_run.log"):
            Simrun.logentry(self, 'Date Status Time\n', mode="w")

    def logentry(self, entry, mode="a"):
```

```

with open(self.rootdir + "\\Operations\\LOGS\\" + self.simtype + "_run.log", mode) as log:
    log.write(entry)

def getfins(self):
    if os.path.isdir(self.projdir + "\\Sim_Weekly\\Operations\\FINS\\" + self.simdate.strftime("%y%m%d")):
        finspath = self.projdir + "\\Sim_Weekly\\Operations\\FINS\\" + self.simdate.strftime("%y%m%d")
    elif os.path.isdir(self.projdir + "\\Sim_Daily\\Operations\\FINS\\" + self.simdate.strftime("%y%m%d")):
        finspath = self.projdir + "\\Sim_Daily\\Operations\\FINS\\" + self.simdate.strftime("%y%m%d")
    else:
        print("-"*80 + f"\n{self.simtype} run ERROR - No FINS available\n" + "-"*80)
        Simrun.logentry(self, datetime.today().strftime(" ERR01 %Y-%m-%d_%H:%M\n"))
        mailreport(self.mailto, os.path.basename(self.projdir) + " Daily run ERROR", "No FINS available", ())
        return 0

    print(" FINS  :", finspath.replace(self.projdir, "."))
    lvpath = self.projdir + "\\Sim_Daily"
    for level in range(self.levels):
        lvpath += '\\Level ' + str(level + 1)
        fins = glob(finspath + f'\\LV{level + 1:02d}*.fin*')

        for fin in fins:
            prefix = os.path.basename(fin).split('_')[1]
            suffix = os.path.basename(fin).split('.')[-1]
            copyfile(fin, lvpath + '\\res\\' + prefix + '_0.' + suffix)
    print()
    return 1

def modeldat(self, runid, ini, end, dtstep, gmtrf):
    print(ini.strftime(" START : %Y %m %d %H %M %S"))
    print(end.strftime(" END   : %Y %m %d %H %M %S"))
    print(" DTs  :", str(dtstep).replace("(", "").replace(")", "").replace(",", ""))
    lvpath = self.rootdir
    for level in range(self.levels):
        lvpath += f'\\Level {level + 1}'

        with open(lvpath + f'\\data\\Model_{runid}.dat', 'w') as model:
            model.write(f"START           : {ini:%Y %m %d %H %M %S}
END             : {end:%Y %m %d %H %M %S}
DT              : {dtstep[level]}
VARIABLEDT     : 0
GMTREFERENCE   : {gmtrf}\n")

        copyfile(lvpath + f'\\data\\Nomfich_{runid}.dat', lvpath + "\\exe\\Nomfich.dat")

def execrun(self, runid):
    with open(self.rootdir + "\\Level 1\\exe\\Tree.dat", 'w') as tree:
        tree.write('Automatic Generated Tree File\nby FERNANDOs AWESOME PYTHON BASED PROGRAM\n')

    lvpath = self.rootdir
    for level in range(self.levels):
        lvpath += f'\\Level {level + 1}'

    with open(self.rootdir + "\\Level 1\\exe\\Tree.dat", 'a') as tree:

```

```

tree.write('+ ' * (level + 1) + lvpath + '\\exe\\n')

if not os.path.exists(lvpath + '\\res\\Run' + str(runid)):
    os.mkdir(lvpath + '\\res\\Run' + str(runid))

# ----- Run simulation
logrun = self.rootdir + "\\Operations\\LOGS\\Mohid_log_run" + str(runid) + ".txt"
logerr = self.rootdir + "\\Operations\\LOGS\\Mohid_log_err" + str(runid) + ".txt"

os.chdir(self.rootdir + '\\Level 1\\exe')
run(f'"{self.projdir}\\MOHID\\MOHIDWater.exe" > {logrun} 2> {logerr}', shell=True)
run('exit', shell=True)
# -----

subject = os.path.basename(self.projdir)+" "+self.simtype+" run "

if not os.path.isfile(logrun):
    print("-"*80 + f"\n{self.simtype} run ERROR - MOHID log not found\n" + "-"*80)
    Simrun.logentry(self, datetime.today().strftime(" ERR04 %Y-%m-%d_%H:%M\n"))
    mailreport(self.mailto, subject+"ERROR", "MOHID log not found", ())
    Simrun.savefail(self)
    return 0

loglines = [line.strip() for line in open(logrun)]
if 'Program Mohid Water successfully terminated' not in loglines:
    print("-" * 80 + f"\n{self.simtype} run ERROR - Simulation stopped unexpectedly\n" + "-" * 80)
    Simrun.logentry(self, datetime.today().strftime(" ERR05 %Y-%m-%d_%H:%M\n"))
    mailreport(self.mailto, subject+"ERROR", "Simulation stopped unexpectedly", (logrun, logerr))
    Simrun.savefail(self)
    return 0

lvpath = self.rootdir
for level in range(self.levels):
    lvpath += f'\\Level {level + 1}'
if not os.path.isfile(lvpath + '\\res\\Hydrodynamic_' + str(runid) + '.fin'):
    print("-" * 80 + f"\n{self.simtype} run ERROR - Hydrodynamic FIN not found\n" + "-" * 80)
    Simrun.logentry(self, datetime.today().strftime(" ERR06 %Y-%m-%d_%H:%M\n"))
    mailreport(self.mailto, subject+"ERROR", "Hydrodynamic FIN not found", (logrun, logerr))
    Simrun.savefail(self)
    return 0

return 1

def savefail(self):
    if not os.path.isdir(self.rootdir + "\\Operations\\FAIL"):
        os.mkdir(self.rootdir + "\\Operations\\FAIL")

    faildir = self.rootdir + self.simdate.strftime("\\Operations\\FAIL\\SIM%y%m%d_END")
    faildir += datetime.today().strftime("%y%m%d_%H%M")
    os.mkdir(faildir)

    boundarys = self.rootdir + "\\General Data\\Boundary Conditions\\*FORC*"
    initials = self.rootdir + "\\General Data\\Initial Conditions\\AVG*"
    logs = self.rootdir + "\\Operations\\LOGS\\Mohid*.txt"

    files = glob(boundarys)+glob(initials)+glob(logs)
    for file in files:
        copyfile(file, faildir + "\\" + os.path.basename(file))

```

```

lvpath = self.rootdir
for level in range(self.levels):
    lvpath += "\\Level " + str(level + 1)
    copytree(lvpath + "\\res", faildir + f"\\LV{level + 1:02d}_res")

def saveres(self):
    outdir = self.rootdir + "\\Operations\\RES"
    if not os.path.isdir(outdir):
        os.mkdir(outdir)

    folders = glob(outdir + "\\*")
    while len(folders) > 20:
        rmtree(folders[0])
        folders = glob(outdir + "\\*")

    outdir += self.simdate.strftime("\\SIM%y%m%d_END") + datetime.today().strftime("%y%m%d_
%H%M")
    os.mkdir(outdir)

    lvpath = self.rootdir
    for level in range(self.levels):
        lvpath += "\\Level " + str(level + 1)

        hdfs = glob(lvpath + "\\res\\Hydrodynamic*.hdf5") + glob(lvpath + "\\res\\WaterProperties*.hdf5")
        for hdf in hdfs:
            copyfile(hdf, outdir + f"\\LV{level + 1:02d}_ " + os.path.basename(hdf))

        timeseries = glob(lvpath + "\\res\\Run[1-3]")
        for folder in timeseries:
            copytree(folder, outdir + f"\\LV{level + 1:02d}_TimeSeries_" + os.path.basename(folder))

def savefin(self, runid, fin_day, modset):
    outdir = self.rootdir + "\\Operations\\FINS"
    if not os.path.isdir(outdir):
        os.mkdir(outdir)

    folders = glob(outdir + "\\*")
    while len(folders) > 20:
        rmtree(folders[0])
        folders = glob(outdir + "\\*")

    findate = self.simdate + timedelta(fin_day)
    outdir += findate.strftime("\\%y%m%d")
    if os.path.isdir(outdir):
        rmtree(outdir)
    os.mkdir(outdir)

    lvpath = self.rootdir
    for level in range(self.levels):
        lvpath += "\\Level " + str(level + 1)

        if modset == "2" and level == 0:
            fins_size = 1
        else:
            fins_size = 3

        fins = glob(lvpath + findate.strftime("\\res\\*%Y%m%d*.fin*"))

```



```

if not fins or len(fins) < fins_size:
    fins = glob(lvpath + f"\\res\\*_{{runid}}.fin*")

for fin in fins:
    copyfile(fin, outdir + f"\\LV{{level + 1:02d}}_" + os.path.basename(fin))

```

Appendix H – Forcing data selector

Module to select the most convenient forcing data from the sources listed.

```

import os
from datetime import timedelta
from glob import glob
from shutil import copyfile

def select_source(srcs, forcdir, simdir, opdate, tsout):
    file_path = []
    for src in srcs:
        if src == "AVGWP" and tsout == "0":
            file_path.append(forcdir + "\\AVGWP\\*_%.hdf5")
        elif src == "AVGWP" and tsout == "1":
            file_path.append(forcdir + "\\AVGWP\\*_%.dat")
        elif tsout == "0":
            file_path.append(forcdir + "\\" + src + "\\Data HDF\\*_%y%m%.hdf5")
        else:
            file_path.append(forcdir + "\\" + src + "\\Data TS\\*_%y%m%.dat")

    file_day = []
    for path in file_path:
        file_day.append(file_availability(path, opdate))

    dtday = min(file_day)
    pos = file_day.index(dtday)
    if dtday > 90:
        return "NOT_FOUND", 99

    src = srcs[pos]
    path = file_path[pos]
    files = glob((opdate - timedelta(dtday)).strftime(path))
    if src == "AVGWP":
        outdir = simdir + "\\General Data\\Initial Conditions\\"
    else:
        outdir = simdir + "\\General Data\\Boundary Conditions\\"

    for file in files:
        file_name = str(os.path.basename(file))
        file_pre = file_name.split("_")
        file_ext = file_name.split(".")[-1]

        if file_ext == "hdf5" or "AVGWP" in file_name:
            file_pre = file_pre[0] + "_" + file_pre[1]
        else:

```

```

    file_pre = file_pre[0]

    file_name = file_pre + "." + file_ext
    print("", file)
    copyfile(file, outdir + file_name)

    return src, dtday

def file_availability(path, opdate):
    max_file = {'Mercator': 6, 'Skiron': 3, 'AVGWP': 6, 'NAM': 3, 'GFS': 3} # maximum number of days
    scanned

    if 'AVGWP' in path:
        max_days = max_file.get(os.path.basename(os.path.dirname(path)))
    else:
        max_days = max_file.get(os.path.basename(os.path.dirname(os.path.dirname(path))))

    if not max_days:
        return 99

    end_while = 0
    file_day = 0

    while file_day <= max_days and end_while < 1:
        file_date = opdate - timedelta(file_day)
        if len(glob(file_date.strftime(path))) > 0:
            end_while = 1
        else:
            file_day += 1

    if file_day > max_days:
        return 99
    return file_day

def check_range(hyd_status, atm_status):
    srcs = {'Mercator': (7, 12.), 'Skiron': (5, 0.), 'AVGWP': (31, 0.), 'NAM': (3, 0.), 'GFS': (5, 0.)}

    hyd_loss = hyd_status[1]
    atm_loss = atm_status[1]

    if srcs.get(hyd_status[0])[1] > srcs.get(atm_status[0])[1]:
        atm_loss += 1
    elif srcs.get(hyd_status[0])[1] < srcs.get(atm_status[0])[1]:
        hyd_loss += 1

    frange = min(srcs.get(hyd_status[0])[0] - hyd_loss, srcs.get(atm_status[0])[0] - atm_loss)
    simhour = max(srcs.get(hyd_status[0])[1], srcs.get(atm_status[0])[1])
    return frange, simhour

```

Appendix I – MOHID outputs database formatting

Module to conduct the formatting of MOHID outputs to populate the local database.

```
import os
from shutil import copyfile, rmtree, copytree
from datetime import datetime, timedelta
from subprocess import run
from glob import glob

import h5py
import numpy as np

def fmtres(resdir, opdate, outdir, levels, modset, mohid):
    return_dir = os.getcwd()

    def mkdir(path):
        if not os.path.isdir(path):
            os.mkdir(path)

    print('-' * 80 + '\n' + "FORMAT RESULTS MODULE".center(80) + '\n' + '-' * 80)
    if not os.path.isdir(resdir) or not os.path.isdir(outdir):
        print("ERROR\nIn/out directory not found\n" + "-" * 80)
        return
    print("INDIR :", resdir)

    outdir += "\\FMT"
    mkdir(outdir)

    incrm = 1
    if modset == "2":
        levels -= 1
        incrm += 1

    for level in range(levels):
        lvd = outdir + "\\Level " + str(level+1)
        mkdir(lvd)

        lvd += opdate.strftime("\\%y%m%d")
        if os.path.isdir(lvd):
            rmtree(lvd)
            mkdir(lvd)

        mohid_files = "HDF5Extractor.exe", "szlibdll.dll", "zlib1.dll"
        for mohid_file in mohid_files:
            try:
                copyfile(mohid + "\\" + mohid_file, lvd + "\\" + mohid_file)
            except FileNotFoundError:
                print("-" * 80 + "\nERROR\nMOHID files not found\n" + "-" * 80)
                return

        timeseries = glob(resdir + f"\\LV{level+incrm:02d}_TimeSeries*")[0]
        try:
            copytree(timeseries, lvd + "\\TimeSeries")
        except FileNotFoundError:
```

```

    print("-"*80 + "\nERROR\nTimeSeries folder not found\n" + "-" * 80)
    return

with open(lvdir + "\\Nomfich.dat", "w") as dat:
    dat.write(f"IN_MODEL : Extractor.dat\nROOT_SRT : {lvdir}\n")

os.chdir(lvdir)
hdf = resdir + f"\\LV{level+incrm:02d}_Hydrodynamic_1.hdf5"
print("HDFIN :", hdf.replace(resdir, "."))
if os.path.isfile(hdf):
    status = extractdata(hdf, "HD")
else:
    print("-"*80 + f"\nERROR\nMissing file: {hdf}\n" + "-" * 80)
    return

if status < 1:
    os.chdir(return_dir)
    print("-"*80 + f"\nERROR\nExtraction failed for\n{hdf}\n" + "-" * 80)
    return

hdf = resdir + f"\\LV{level+incrm:02d}_WaterProperties_1.hdf5"
print("HDFIN :", hdf.replace(resdir, "."))
if os.path.isfile(hdf):
    status = extractdata(hdf, "WP")
else:
    print("-"*80 + f"\nERROR\nMissing file: {hdf}\n" + "-" * 80)
    return
os.chdir(return_dir)

if status < 1:
    print("-"*80 + f"\nERROR\nExtraction failed for\n{hdf}\n" + "-" * 80)
    return

olds = glob(lvdir + "\\*.exe') + glob(lvdir + "\\*.log') + glob(lvdir + "\\*.dat') + glob(lvdir + "\\*.txt')
olds += glob(lvdir + "\\*dll")
for old in olds:
    os.unlink(old)

print('-' * 80 + '\n' + "FORMAT RESULTS MODULE COMPLETED" + '\n' + '-' * 80)

def extractdata(hdf, flavor):
    if flavor == "HD":
        hdfpre = "Hydrodynamic_"
    else:
        hdfpre = "WaterProperties_"

    with h5py.File(hdf, 'r') as hdfin:
        dateini = np.array(hdfin['Time/Time_00001'], dtype=np.int)
        datefin = np.array(hdfin['Time/Time_00002'], dtype=np.int)

    dateini = datetime(dateini[0], dateini[1], dateini[2], dateini[3], dateini[4])
    datefin = datetime(datefin[0], datefin[1], datefin[2], datefin[3], datefin[4])
    stptime = (datefin-dateini).total_seconds()
    datalen = int(24*3600/stptime)

    for instant in range(datalen):

```

```

hdfdate = dateini+timedelta(seconds=instant*stptime)
hdfout = hdfpre + hdfdate.strftime('%Y%m%d_%H%M.hdf5')
print(' Writing...', hdfout, f' {instant+1}/{datalen}')

dat = open('Extractor.dat', 'w')
dat.write(f'""FILENAME      : {hdf}\nOUTPUTFILENAME    : {hdfout}

```

```

START_TIME : {hdfdate:%Y %m %d %H %M} 0
END_TIME   : {hdfdate:%Y %m %d %H %M} 0

```

```

INTERVAL      : 0\n\n""")

```

```

    if flavor == 'HD':

```

```

        dat.write('""<BeginParameter>\n

```

```

PROPERTY      : velocity U
HDF_GROUP     : /Results/velocity U
<EndParameter>

```

```

<BeginParameter>

```

```

PROPERTY      : velocity V
HDF_GROUP     : /Results/velocity V
<EndParameter>

```

```

<BeginParameter>

```

```

PROPERTY      : velocity W
HDF_GROUP     : /Results/velocity W
<EndParameter>

```

```

<BeginParameter>

```

```

PROPERTY      : velocity modulus
HDF_GROUP     : /Results/velocity modulus
<EndParameter>

```

```

<BeginParameter>

```

```

PROPERTY      : water level
HDF_GROUP     : /Results/water level
<EndParameter>\n""")

```

```

    else:

```

```

        dat.write('""<BeginParameter>

```

```

PROPERTY      : temperature
HDF_GROUP     : /Results/temperature
<EndParameter>

```

```

<BeginParameter>

```

```

PROPERTY      : salinity
HDF_GROUP     : /Results/salinity
<EndParameter>

```

```

<BeginParameter>

```

```

PROPERTY      : density
HDF_GROUP     : /Results/density
<EndParameter>\n""")

```

```

dat.close()

```

```

run('HDF5Extractor.exe > log_run.txt 2> log_err.txt', shell=True)

```

```

if not os.path.isfile('log_run.txt') or not os.path.isfile('log_err.txt'):
    return 0

```

```

log_run = [line.strip() for line in open('log_run.txt', 'r')]
log_err = [line.strip() for line in open('log_err.txt', 'r')]

if 'Program HDF5Extractor successfully terminated' not in log_run or log_err:
    return 0

return 1

```

Appendix J – SOMA outputs OCASO formatting

Module to conduct the formatting of SOMA outputs to upload to OCASO SFTP.

```

import os
from shutil import rmtree
from glob import glob
from time import sleep
from datetime import datetime, timedelta

import pysftp

from post_pde_buildnc import BuildNetcdf

def convertpde(resdir, oupdate, outdir, modset, levels):
    return_dir = os.getcwd()

    def makedir(path):
        if not os.path.isdir(path):
            os.mkdir(path)

    print('-'*80+'\n'+ 'PUERTOS DEL ESTADO NETCDF MODULE'.center(80)+'\n'+ '-'*80)
    if not os.path.isdir(resdir) or not os.path.isdir(outdir):
        print("ERROR\nIn/out directory not found\n" + "-" * 80)
        return
    print("INDIR :", resdir)

    outdir += "\\PDE"
    makedir(outdir)

    incmnt = 1
    if modset == "2":
        levels -= 1
        incmnt += 1

    outdir += oupdate.strftime("\\PDE_%y%m%d")
    if os.path.isdir(outdir):
        rmtree(outdir)
        os.mkdir(outdir)

    for level in range(levels):
        hdfs = glob(resdir + f"\\LV{level + incmnt:02d}_*.hdf5")
        if len(hdfs) != 4:
            print("-" * 80 + "\nERROR\nSimulation outputs missing\n" + "-" * 80)
            return

```

```

for level in range(levels):
    hdfin = resdir + f"\\LV{level+incrm:02d}_Hydrodynamic_1.hdf5"
    print("HDFIN :", hdfin.replace(resdir, "."))
    tool = BuildNetcdf(outdir, level + 1)

    tool.time_and_grid(hdfin)
    dsetnum = tool.dsetsamnt(hdfin)

    print(" Velocity U")
    velu = tool.trid_dsets(hdfin, "/Results/velocity U", 1, dsetnum, "uo")
    print(" Velocity V")
    velv = tool.trid_dsets(hdfin, "/Results/velocity V", 1, dsetnum, "vo")
    print(" Velocity W")
    tool.trid_dsets(hdfin, "/Results/velocity W", 1, dsetnum, "wo")
    print(" Water Level")
    tide = tool.trid_dsets(hdfin, "/Results/water level", 0, dsetnum, "zos")

    hdfin = resdir + f"\\LV{level+incrm:02d}_WaterProperties_1.hdf5"
    print("HDFIN :", hdfin.replace(resdir, "."))
    dsetnum = tool.dsetsamnt(hdfin)

    print(" Temperature")
    temp = tool.trid_dsets(hdfin, "/Results/temperature", 0, dsetnum, 'thetao')
    print(" Salinity")
    sali = tool.trid_dsets(hdfin, "/Results/salinity", 0, dsetnum, "so")
    print(" Density")
    tool.trid_dsets(hdfin, "/Results/density", 0, dsetnum, "rho")

    hdfin = resdir + f"\\LV{level + incrm:02d}_Hydrodynamic_1_Surface.hdf5"
    print("HDFIN :", hdfin.replace(resdir, "."))
    dsetnum = tool.dsetsamnt(hdfin, surf=1)

    print(" Velocity U")
    tool.bid_dsets(hdfin, "/Results/velocity U", 1, dsetnum, "uo", velu)
    print(" Velocity V")
    tool.bid_dsets(hdfin, "/Results/velocity V", 1, dsetnum, "vo", velv)
    print(" Tide")
    tool.bid_dsets(hdfin, "/Results/water level", 0, dsetnum, "zos", tide)

    hdfin = resdir + f"\\LV{level + incrm:02d}_WaterProperties_1_Surface.hdf5"
    print("HDFIN :", hdfin.replace(resdir, "."))
    dsetnum = tool.dsetsamnt(hdfin, surf=1)

    print(" Temperature")
    tool.bid_dsets(hdfin, "/Results/temperature", 0, dsetnum, "thetao", temp)
    print(" Salinity")
    tool.bid_dsets(hdfin, "/Results/salinity", 0, dsetnum, "so", sali)

os.chdir(return_dir)

try:
    sleep_time = datetime.fromordinal((opdate + timedelta(1)).toordinal()) - datetime.today()
    sleep(sleep_time.total_seconds())
except ValueError:
    pass

print("-"*80+"\n\nUpload to SFTP")

```

```

cnopts = pysftp.CnOpts()
cnopts.hostkeys = None
sftp = pysftp.Connection('ualg-ocaso.ualg.pt', username='userftpocaso',
password='iK7re8baYXpsEGLxjkWx',
                    cnopts=cnopts)

sftp.mkdir('/PDE/' + os.path.basename(outdir))
sftp.put_r(outdir, '/PDE/' + os.path.basename(outdir), preserve_mtime=True)
sftp.close()

print("\n"+"-"*80)
print('MODULE COMPLETED')
print("-"*80)

```

Appendix K – OCASO formatting supporting class

Class with the methods to manipulate the datasets and write them in the NetCDF files.

```

from datetime import datetime, timedelta

import numpy as np
from h5py import File
from netCDF4 import Dataset

class BuildNetcdf:
    def __init__(self, outdir, level):
        self.outdir = outdir
        self.level = level
        self.project = "SOMA"
        self.valfill = -32767
        self.inifct = None
        self.frange = None
        self.surf = None
        self.vgrid = None
        self.scfofs = {}

    def time_and_grid(self, hdfin):
        with File(hdfin, "r") as hdf:
            self.inifct = datetime(*tuple([int(val) for val in hdf["/Time/Time_00001"]]))
            dsets_num = len(hdf["/Time"])
            endfct = datetime(*tuple([int(val) for val in hdf[f"/Time/Time_{dsets_num:05d}"]]))

            lat = np.array(hdf['Grid/Latitude'])[0]
            lon = np.array(hdf['Grid/Longitude']).transpose()[0]
            layers = np.array(hdf['Grid/VerticalZ/Vertical_00001']).transpose()

            lat = lat[:-1]
            lon = lon + (lon[1] - lon[0]) / 2
            lon = lon[:-1]

            maxdep = np.where(layers == np.amax(layers))
            layers = layers[maxdep[0][0], maxdep[1][0], ...]

```



```

layers_amnt = len(layers) - 1

layers = layers[layers > 0]
layers.sort()

self.surf = lat, lon
self.vgrid = layers, layers_amnt

frange = int((endfct - self.inifct).total_seconds()/3600)
if frange % 24 == 0:
    self.frange = int(frange/24)
elif (frange + 1) % 24 == 0:
    self.frange = int((frange + 1)/24)
else:
    self.frange = 0

def writetime(valfill, ncout, time_inst):
    ncout.createDimension('time', 1)
    varid = ncout.createVariable('time', 'f4', ('time',), fill_value=valfill)
    varid.axis = 'T'
    varid.calendar = 'gregorian'
    varid.standard_name = 'time'
    varid.long_name = 'time'
    varid.units = 'hours since 1950-01-01 00:00:00'
    greg = (time_inst - datetime(1950, 1, 1, 0)).total_seconds() / 3600
    varid.valid_min = greg
    varid.valid_max = greg
    varid.CoordinateAxisType = 'Time'
    varid[:] = np.array([greg])

def writehgrid(ncout, latarr, lonarr):
    ncout.createDimension('latitude', len(latarr))
    ncout.createDimension('longitude', len(lonarr))

    varid = ncout.createVariable('latitude', 'f4', ('latitude',), fill_value=False)
    varid.standard_name = 'latitude'
    varid.long_name = 'Latitude'
    varid.units = 'degrees_north'
    varid.unit_long = 'Degrees North'
    varid.axis = 'Y'
    varid.Valid_min = latarr.min()
    varid.Valid_max = latarr.max()
    varid.step = latarr[1] - latarr[0]
    varid.CoordinateAxisType = 'Lat'
    varid[:] = latarr

    varid = ncout.createVariable('longitude', 'f4', ('longitude',), fill_value=False)
    varid.standard_name = 'longitude'
    varid.long_name = 'Longitude'
    varid.units = 'degrees_east'
    varid.unit_long = 'Degrees East'
    varid.axis = 'X'
    varid.Valid_min = lonarr.min()
    varid.Valid_max = lonarr.max()
    varid.step = lonarr[1] - lonarr[0]
    varid.CoordinateAxisType = 'Lon'
    varid[:] = lonarr

```

```

def writevgrid(ncout, vgrid):
    ncout.createDimension('depth', len(vgrid))
    varid = ncf.createVariable('depth', 'f4', ('depth',), fill_value=False)
    varid.standard_name = 'depth'
    varid.long_name = 'Vertical distance below the surface'
    varid.axis = 'Z'
    varid.units = 'm'
    varid.unit_long = 'Meters'
    varid.Valid_min = vgrid.min()
    varid.Valid_max = vgrid.max()
    varid.positive = 'down'
    varid.CoordinateAxisType = 'Height'
    varid.CoordinateZisPositive = 'down'
    varid[:] = layers

for inst in range(self.frange):
    instant = self.inifct + timedelta(inst)
    ncname = '\\' + self.project + '-L' + str(self.level) + instant.strftime('_dm-%Y%m%d%H-B') +
self.inifct.strftime('%Y%m%d%H-FC.nc')

    with Dataset(self.outdir + ncname, 'w') as ncf:
        writetime(self.valfill, ncf, instant)
        writehgrid(ncf, lat, lon)
        writevgrid(ncf, self.vgrid[0])

for inst in range(self.frange*24):
    instant = self.inifct + timedelta(hours=inst)
    ncname = '\\' + self.project + '-L' + str(self.level) + instant.strftime('_hv-%Y%m%d%H-B') +
self.inifct.strftime('%Y%m%d%H-FC.nc')

    with Dataset(self.outdir + ncname, 'w') as ncf:
        writetime(self.valfill, ncf, instant)
        writehgrid(ncf, lat, lon)

def dssetsamnt(self, hdfin, surf=0):
    with File(hdfin, "r") as hdf:
        secfct = datetime(*tuple([int(val) for val in hdf["/Time/Time_00002"]]))

    step = (secfct - self.inifct).total_seconds()

    if surf == 0:
        amnt = int(86400/step)
    else:
        amnt = int(3600/step)

    if amnt < 1:
        return 1

    return amnt

def trid_dssets(self, hdfin, grp, land, dset_amnt, ncv):
    hdf = File(hdfin, "r")
    dset_shape = len(hdf["/Time"].keys()), self.vgrid[1], len(self.surf[1]), len(self.surf[0])
    dset = importdset(hdf[grp], dset_shape)

    if land == 1:
        opnpts = importdset(hdf['/Grid/OpenPoints'], dset_shape)

```

```

    dset = dset + (((opnpts - 1) * (-1)) * (-9.9e15))

hdf.close()

scf, off = scaleoffset(dset)
mean = np.zeros((self.frange, dset_shape[1], dset_shape[2], dset_shape[3]))

for day in range(self.frange):
    mean[day] = np.mean(dset[day * dset_amnt:(day + 1) * dset_amnt], axis=0)
del dset

for day in range(self.frange):
    instant = self.inifct + timedelta(day)
    ncname = '\\' + self.project + '-L' + str(self.level) + instant.strftime('_dm-%Y%m%d%H-B') + \
        self.inifct.strftime('%Y%m%d%H-FC.nc')

    with Dataset(self.outdir + ncname, "a") as ncfile:
        varid = ncfile.createVariable(ncvar, 'f8', ('time', 'depth', 'longitude', 'latitude'),
            fill_value=self.valfill)
        dsetatt(varid, grpuid, scf, off)
        prop = mean[day]
        prop = prop[::-1]
        prop = prop[:len(self.vgrid[0])]
        prop = np.ma.masked_less(prop, -98.99)
        varid[:] = prop # Write array in variable

return scf, off

def bid_dsets(self, hdfin, grpuid, land, dset_amnt, ncvar, ncsc1):
    hdf = File(hdfin, "r")
    dset_shape = len(hdf["/Time"].keys()), len(self.surf[1]), len(self.surf[0])
    dset = importdset(hdf[grpuid], dset_shape)

    if land == 1:
        opnpts = importdset(hdf['/Grid/OpenPoints'], dset_shape)
        dset = dset + (((opnpts - 1) * (-1)) * (-9.9e15))

    hdf.close()

    frange = int(self.frange*24)
    mean = np.zeros((frange, dset_shape[1], dset_shape[2]))

    for hour in range(frange):
        mean[hour] = np.mean(dset[hour * dset_amnt:(hour + 1) * dset_amnt], axis=0)
    del dset

    for hour in range(frange):
        instant = self.inifct + timedelta(hours=hour)
        ncname = '\\' + self.project + '-L' + str(self.level) + instant.strftime('_hv-%Y%m%d%H-B') + \
            self.inifct.strftime('%Y%m%d%H-FC.nc')

        with Dataset(self.outdir + ncname, "a") as ncfile:
            varid = ncfile.createVariable(ncvar, 'f8', ('time', 'longitude', 'latitude'), fill_value=self.valfill)
            dsetatt(varid, grpuid, ncsc1[0], ncsc1[1])
            prop = mean[hour]
            prop = np.ma.masked_less(prop, -98.99)
            varid[:] = prop

```

```

def importdset(group, shape):
    dataset = np.zeros(shape)
    indice = 0

    for key in group.keys():
        dataset[indice] = np.array(group.get(key))
        indice += 1

    return dataset

def scaleoffset(hdfin, groupid):
    """Stretch/compress data to the available packed range"""
    vmin = 9.9e15
    vmax = -9.9e15

    hdf = File(hdfin, "r")
    for key in hdf.get(groupid).keys():
        dset = np.array(hdf.get(groupid + '/' + key))
        dset = dset[dset > -9.889e15]
        vmin = min(vmin, dset.min())
        vmax = max(vmax, dset.max())
    hdf.close()

    n = 8
    scf = (vmax - vmin) / (2 ** (n - 1))
    if -1 < scf < 1:
        scf = round(scf, 4)
    else:
        scf = round(scf)

    # translate the range to be symmetric about zero
    ofs = vmin + 2 ** (n - 1) * scf
    if -1 < ofs < 1:
        ofs = round(ofs, 4)
    else:
        ofs = round(ofs)

    print("SFC", scf, "OFS", ofs)
    return scf, ofs

def dsetatt(varid, group, scale_factor, offset):
    """Writes attributes in netcdf file for a dataset"""
    dset_dict = {
        "/Results/velocity U": ('eastward_sea_water_velocity', 'Eastward Velocity', 'm s-1', 'Meters per
Second'),
        "/Results/velocity V": ('northward_sea_water_velocity', 'Northward Velocity', 'm s-1', 'Meters per
Second'),
        "/Results/velocity W": ('vertical_sea_water_velocity', 'Vertical Velocity', 'm s-1', 'Meters per Second'),
        "/Results/temperature": ('sea_water_potential_temperature', 'Temperature', 'degrees_C', 'Degrees
Celsius'),
        "/Results/salinity": ('sea_water_salinity', 'Salinity', 'psu', 'Practical Salinity Unit'),
        "/Results/density": ('sea_water_density', 'Density', 'kg m3-1', 'kilograms per cubic meter'),
        "/Results/water level": ('sea_surface_height_above_geoid', 'Sea surface height', 'm', 'Meters')}

    varid.scale_factor = scale_factor

```

```

varid.add_offset = offset
varid.long_name = dset_dict.get(group)[0]
varid.standard_name = dset_dict.get(group)[1]
varid.units = dset_dict.get(group)[2]
varid.unit_long = dset_dict.get(group)[3]

varid.set_auto_scale(True)

```

Appendix L – Mercator module

Module to process CMEMS Mercator data for MOHID use.

```

import os
from subprocess import run
from datetime import datetime, timedelta
from glob import glob
from shutil import copyfile

import numpy as np
from netCDF4 import Dataset

from inputsread import initread
from opcommon import mailreport, logentry
from forcstructure import ForcStructure

def download(outdir, grid, cred, daterange):
    if not os.path.isdir(outdir):
        os.mkdir(outdir)

    logs = outdir + '\\download_log.txt', outdir + '\\error.txt'
    command = 'python -m motuclient --motu http://nrt.cmems-du.eu/motu-web/Motu --service-id
GLOBAL_ANALYSIS_FORECAST_PHY_001_024-TDS --product-id global-analysis-forecast-phy-001-
024'
    command += ' --longitude-min ' + str(grid[0]) + ' --longitude-max ' + str(grid[1]) + ' --latitude-min '
    command += str(grid[2]) + ' --latitude-max ' + str(grid[3]) + ' --date-min "' + str(daterange[0])
    command += '" --date-max "' + str(daterange[1]) + '" --depth-min 0.493 --depth-max
5727.918000000001 '

    command += '--variable uo --variable vo --variable thetao --variable so '

    command += f'--out-dir {outdir} --out-name Mercator.nc --user ' + cred[0] + ' --pwd ' + cred[1]
    command += ' > ' + logs[0] + ' 2> ' + logs[1]

    print("Downloading Mercator NETCDF file", end="\n\n")
    run(command, shell=True)

    if not os.path.isfile(outdir + '\\Mercator.nc'):
        print("-" * 80 + "\nERROR - NETCDF file not found\n" + "-" * 80)
        return "ERR02", logs

    with open(logs[1], "r") as log:
        lines = log.readlines()
    if lines:

```

```

print("-" * 80 + "\nERROR - Download failed\n" + "-" * 80)
return "ERR03", logs

with Dataset(outdir + '\\Mercator.nc', 'r') as dset:
    dset_size = dset.variables['time'].size

if dset_size != (daterange[1] - daterange[0]).days + 1:
    print("-" * 80 + "\nERROR - Downloaded NETCDF file with missing datasets\n" + "-" * 80)
    return "ERR04", logs
return "GOOD JOB", ()

def conversion(outdir, dwnfile):
    logs = outdir + '\\conversion_log.txt', outdir + '\\error.txt'

    # vars = (name, units, description)
    velu = ('velocity U', 'm/s', 'Mercator velocity U')
    velv = ('velocity V', 'm/s', 'Mercator velocity V')
    temp = ('temperature', '°C', 'Mercator temperature')
    sali = ('salinity', 'psu', 'Mercator salinity')
    varsid = {'uo': velu, 'vo': velv, 'thetao': temp, 'so': sali}

    dset = Dataset(dwnfile, 'r')
    with open(outdir + '\\ConvertToHDF5Action.dat', 'w') as dat:
        dat.write(f"*****<begin_file>
ACTION      : CONVERT NETCDF CF TO HDF5 MOHID
HDF5_OUT    : 1
NETCDF_OUT  : 0
OUTPUTFILENAME : Mercator.hdf5

<<begin_time>>
NETCDF_NAME : time
<<end_time>>

<<begin_grid>>
NETCDF_NAME_LAT   : latitude
NETCDF_NAME_LONG  : longitude
NETCDF_NAME_MAPPING : uo
MAPPING_LIMIT     : -32000
NETCDF_NAME_DEPTH : depth
INVERT_LAYER_ORDER : 1
BATHYM_FROM_MAP   : 1
BATHYM_FILENAME   : Mercator.dat
<<end_grid>>

PROPERTIES_NUMBER : {len(varsid)}\n\n*****")

    for var in varsid:
        dat.write(f"*****<begin_field>>
NETCDF_NAME      : {var}
NAME             : {varsid[var][0]}
UNITS            : {varsid[var][1]}
DESCRIPTION      : {varsid[var][2]}
DIM              : {dset.variables[var].ndim - 1}
ADD_FACTOR       : {dset.variables[var].add_offset}
MULTIPLY_FACTOR  : {dset.variables[var].scale_factor}
<<end_field>>\n\n*****")

```

```

    dat.write(f"<<begin_input_files>>\n{dwnfile}\n<<end_input_files>>\n<end_file>\n")
dset.close()

os.chdir(outdir)
print("Converting NETCDF file into HDF5", end="\n\n")
run("ConvertToHdf5_release_double.exe > conversion_log.txt 2> error.txt", shell=True)

if not os.path.isfile(outdir + "\\Mercator.hdf5"):
    print('-' * 80 + '\nERROR - Converted hdf5 not found\n' + '-' * 80)
    return 'ERR06', logs

error = [line.strip() for line in open(outdir + "\\error.txt", 'r')]
if error:
    print('-' * 80 + '\nERROR - Conversion failed\n' + '-' * 80)
    return 'ERR07', logs
return "GOOD JOB", ()

def interpolation(outdir, modset, projdir, dwndnc, hdfint, hdfgrd):
    # ----- MODEL BATIM AND GEOMETRY
    batim = projdir + f"\\Sim_Daily\\General Data\\Digital Terrain\\BATIM_LV{modset}.dat"
    if int(modset) < 2:
        geomt = projdir + "\\Sim_Daily\\Level 1\\data\\Geometry_1.dat"
    else:
        geomt = projdir + "\\Sim_Daily\\Level 1\\Level 2\\data\\Geometry_1.dat"
    try:
        if not os.path.isfile(outdir + "\\" + os.path.basename(batim)):
            copyfile(batim, outdir + "\\" + os.path.basename(batim))
        if not os.path.isfile(outdir + "\\Geometry_out.dat"):
            copyfile(geomt, outdir + "\\Geometry_out.dat")
    except FileNotFoundError:
        print("-" * 80 + "\nERROR - Model files not found\n" + "-" * 80)
        return "ERR08", ()

    # ----- WRITE GEOMETRY FROM DOWNLOADED NETCFD FILE
    with Dataset(dwndnc, 'r') as dset:
        depth = np.array(dset.variables['depth'] [...])

    dat = open(outdir + "\\Geometry_in.dat", 'w')
    dat.write("<begindomain>\nID      : 1\nTYPE      : CARTESIAN\nDOMAINDEPTH : 0\n")
    dat.write(f"LAYERS      : {len(depth)}\n<<beginlayers>>\n")
    for layer in range(len(depth)):
        if layer < 1:
            dat.write(f'{depth[layer]}\n')
        else:
            dat.write(f'{depth[layer] - depth[layer - 1]}\n')
    dat.write(f"<<endlayers>>\nMININITIALLAYERTHICKNESS : 1\n<enddomain>\n")
    dat.close()

    # ----- RUN INTERPOLATION
    def convertactiondat(inthdf, grdhdf, hdfout, btmout):
        with open('ConvertToHDF5Action.dat', 'w') as inter_dat:
            inter_dat.write(f"\"<begin_file>
ACTION      : INTERPOLATE GRIDS
TYPE_OF_INTERPOLATION : 3
INTERPOLATION3D : 1
FATHER_FILENAME : {inthdf}

```

```

FATHER_GRID_FILENAME : {grdhdf}
OUTPUTFILENAME       : {hdfout}
NEW_GRID_FILENAME    : {btmout}
FATHER_GEOMETRY      : Geometry_in.dat
NEW_GEOMETRY         : Geometry_out.dat
EXTRAPOLATE_2D      : 2
BASE_GROUP           : /Results
POLI_DEGREE          : 1\n<end_file>""")

os.chdir(outdir)
convertactiondat(hdfint, hdfgrd, f'HYDFORC_LV{modset}.hdf5', f'BATIM_LV{modset}.dat')
print("Interpolating HDF5 file", end="\n\n")
run('Convert2Hdf5.exe > interpolation_log.txt 2> error.txt', shell=True)

logs = outdir + "\\interpolation_log.txt", outdir + "\\error.txt"
error = tuple([line.strip() for line in open(logs[1])])
if 'VerifyBathymetry - Geometry - ERR165' in error:
    hdfgrd = os.path.dirname(hdfgrd) + "\\Mercator_v01.dat"
    os.chdir(outdir)
    convertactiondat(hdfint, hdfgrd, f'HYDFORC_LV{modset}.hdf5', f'BATIM_LV{modset}.dat')
    run('Convert2Hdf5.exe > interpolation_log.txt 2> error.txt', shell=True)

if not os.path.isfile(outdir + f"\\HYDFORC_LV{modset}.hdf5"):
    print('-' * 80 + '\nERROR - Interpolated HDF5 not found\n' + '-' * 80)
    return "ERR09", logs

error = [line.strip() for line in open(logs[1])]
if error:
    print('-' * 80 + '\nERROR - Interpolation failed\n' + '-' * 80)
    return "ERR10", logs
return "GOOD JOB", ()

def mercator(outdir, opdate, inputs):
    srcdir = outdir + "\\Mercator"
    logrun = srcdir + "\\Mercator_run_status.log"

    print("-"*80 + "\n" + "MERCATOR DATA MODULE".center(80) + "\n" + "-"*80)
    print("WOORKING DIRECTORY :", outdir)
    print("PROCESS DATE      :", opdate.strftime("%Y %m %d"))
    print("HDFOUT             :", inputs.get("HDFOUT"))
    print("TSOUT              :", inputs.get("TSOUT"))
    print("-"*80)

    hdfout = inputs.get("HDFOUT") == "1" and len(glob(opdate.strftime(srcdir + "\\Data HDF\\*%y%m
%d.hdf5"))) > 0
    tsout = inputs.get("TSOUT") == "1" and len(glob(opdate.strftime(srcdir + "\\Data TS\\*%y%m
%d.dat"))) > 0
    if hdfout or tsout:
        print('-' * 80 + '\nDATA FILE ALREADY AVAILABLE FOR SELECTED DATE\n' + '-' * 80)
        return 1

    if not os.path.isdir(srcdir):
        os.mkdir(srcdir)
    if not os.path.isfile(logrun):
        logentry(logrun, "DATE STATUS ENDTIME\n", mode="w")
    logentry(logrun, str(opdate))

```



```

cred = initread(("MERC_CRED", ), inputs.get("PROJDIR")).get("MERC_CRED")
if not cred or len(cred.split()) != 2:
    print("-"*80 + "\nERROR - MERC_CRED not found\n" + "-"*80)
    logentry(logrun, datetime.today().strftime(" ERR01 %Y-%m-%d_%H:%M\n"))
    mailreport(inputs.get("MAILTO"), "Mercator process ERROR", "ERR01 - MERC_CRED value
incorrect or missing", ())
    return 0
cred = tuple(cred.split())

print("Removing old files", end="\n\n")
olds = glob(srcdir + "\\Download\\*.nc") + glob(srcdir + "\\Conversion\\*.hdf5")
olds += glob(srcdir + "\\Interpolation\\*.hdf5")
for old in olds:
    os.unlink(old)

manager = ForcStructure(srcdir, inputs.get("MOHID"), update)
manager.datawait(12, 30)

timerun = update - timedelta(inputs.get("HNDCAST") + 1), update + timedelta(7)

status = download(srcdir + "\\Download", inputs.get("GRID"), cred, timerun)
if "ERR" in status[0]:
    logentry(logrun, datetime.today().strftime(f" {status[0]} %Y-%m-%d_%H:%M\n"))
    body = update.strftime(f"%Y-%m-%d operation failed\n{status} - Mercator download failed")
    mailreport(inputs.get("MAILTO"), "Mercator process ERROR", body, status[1])
    return 0

if inputs.get("HDFOUT") == "1":
    manager.hydhdfs()
    status = manager.mohid_conversion(srcdir + "\\Conversion")
    if status < 1:
        print("-" * 80 + "\nERROR - MOHID conversion files not found\n" + "-" * 80)
        logentry(logrun, datetime.today().strftime(" ERR05 %Y-%m-%d_%H:%M\n"))
        body = update.strftime("%Y-%m-%d operation failed\nERR05 - MOHID files not found")
        mailreport(inputs.get("MAILTO"), "Mercator process ERROR", body, ())
        return 0

status = conversion(srcdir + "\\Conversion", srcdir + "\\Download\\Mercator.nc")
if "ERR" in status[0]:
    logentry(logrun, datetime.today().strftime(f" {status[0]} %Y-%m-%d_%H:%M\n"))
    body = update.strftime(f"%Y-%m-%d operation failed\n{status} - Mercator conversion failed")
    mailreport(inputs.get("MAILTO"), "Mercator process ERROR", body, status[1])
    return 0

status = manager.mohid_interpolation(srcdir + "\\Interpolation")
if status < 1:
    print("-" * 80 + "\nERROR - Conversion files not found\n" + "-" * 80)
    logentry(logrun, datetime.today().strftime(" ERR05 %Y-%m-%d_%H:%M\n"))
    body = update.strftime("%Y-%m-%d operation failed\nERR05 - MOHID files not found")
    mailreport(inputs.get("MAILTO"), "Mercator process ERROR", body, ())
    return 0

status = interpolation(srcdir + "\\Interpolation", inputs.get("MODSET"), inputs.get("PROJDIR"),
    srcdir + "\\Download\\Mercator.nc", srcdir + "\\Conversion\\Mercator.hdf5",
    srcdir + "\\Conversion\\Mercator.dat")
if "ERR" in status[0]:
    logentry(logrun, datetime.today().strftime(f" {status[0]} %Y-%m-%d_%H:%M\n"))

```

```

body = update.strftime(f"Operation for: %Y-%m-%d\n{status} - Mercator interpolation failed")
mailreport(inputs.get("MAILTO"), "Mercator process ERROR", body, status[1])
return 0

os.chdir(outdir)
hdf = srcdir + f"\\Interpolation\\HYDFORC_LV{inputs.get('MODSET')}.hdf5"
copyfile(hdf, srcdir + update.strftime(f"\\Data HDF\\HYDFORC_LV{inputs.get('MODSET')}_%y
%m%d.hdf5"))

print('-' * 80 + "\nMERCATOR MODULE COMPLETED\n" + '-' * 80)
logentry(logrun, datetime.today().strftime(" 1 %Y-%m-%d_%H:%M\n"))
mailreport(inputs.get("MAILTO"), "Mercator process COMPLETED", update.strftime("%Y-%m-%d
Files available"), ())
return 1

```

Appendix M – Skiron module

Module to process Skiron data for MOHID use.

```

import os
from ftplib import FTP, error_perm
from shutil import rmtree, copytree, copyfile
from datetime import datetime, timedelta
from glob import glob
from subprocess import run

import numpy as np
from h5py import File

from opcommon import mailreport, logentry
from forcstructure import ForcStructure
from forcgribs import Gribball, writehdf, relhum

def download(outdir, actdate):
    if not os.path.isdir(outdir):
        os.mkdir(outdir)

    ftp = FTP('ftp.mg.uoa.gr')
    ftp.login('mfstep', '!lam')
    ftpdir = actdate.strftime('/forecasts/Skiron/daily/005X005/%d%m%y')
    try:
        ftp.cwd(ftpdir)
    except error_perm:
        rmtree(outdir)
        return "ERR01"

    grbs = ftp.nlst()
    grbs.sort()
    if actdate < datetime.today().date():
        grbs_size = 24
    else:
        grbs_size = 121

```

```

print("Downloading Skiron GRIB files")
for inst in range(grbs_size):
    try:
        print("", grbs[inst])
        ftp.retrbinary("RETR " + grbs[inst], open(outdir + '\\' + grbs[inst], 'wb').write)
    except (EOFError, TimeoutError, ConnectionResetError):
        rmtree(outdir)
        return "ERR02"

ftp.close()

grbs = glob(outdir + '\\*.grb')
if len(grbs) < grbs_size:
    rmtree(outdir)
    return "ERR03"

grbs.sort()
inst = 0
for grb in grbs:
    os.rename(grb, outdir + actdate.strftime("\\MFSTEP005_00%y%m%d_") + f'{inst:03d}.grb')
    inst += 1
return "GOOD JOB"

def conversion(outdir, dwn dates, grid_lim):
    print("Converting GRIB files in one HDF5")
    hdf = outdir + "\\Skiron.hdf5"

    grbs = []
    for folder in glob(dwn dates + "\\*"):
        grbs += glob(folder + "\\*.grb")
    grbs.sort()
    grbs_size = len(grbs)

    grb = Griball(grbs[0])
    status = grb.opengrb()
    if status < 1:
        print('-' * 80 + f'\nERROR - Failed to open grib file, {grbs[0]}\n' + '-' * 80)
        return "ERR04"

    print(" GRID")
    londset, latdset, cutdset = grb.hdfgrid(grid_lim)
    latdset, londset = np.meshgrid(latdset, londset)
    dset = grb.grbdset(15, cutdset=cutdset, hdfout=1).astype('i4') # land 1 sea 0
    batim = dset.copy()
    batim = batim * (-99) + abs(batim - 1) * 10000
    writehdf(hdf, 'Grid/Latitude', latdset, "deg", "w")
    writehdf(hdf, 'Grid/Longitude', londset, "deg", "a")
    writehdf(hdf, 'Grid/WaterPoints', abs(dset - 1), "-", "a")
    writehdf(hdf, 'Grid/Bathymetry', batim, 'm', "a")
    del latdset, londset, batim, grb

    inst = 1
    for src_grb in grbs:
        print(f'\r DATASETS {inst / grbs_size * 100:.1f}%', end="")
        grb = Griball(src_grb)

```

```

status = grb.opengrb()
if status < 1:
    print('-' * 80 + f'\nERROR - Failed to open grib file, {grbs[0]}\n' + '-' * 80)
    return "ERR04"

dset = grb.grbtime()
dset = np.array([dset.year, dset.month, dset.day, dset.hour, dset.minute, dset.second])
writehdf(hdf, f'Time/Time_{inst:05d}', dset, 'YYYY/MM/DD HH:MM:SS', "a")

sp_hum = grb.grbdset(4, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/specific humidity/specific humidity_{inst:05d}', sp_hum, "kg/kg", "a")
apress = grb.grbdset(6, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/atmospheric pressure/atmospheric pressure_{inst:05d}', apress, "Pa", "a")
airtmp = grb.grbdset(3, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/air temperature/air temperature_{inst:05d}', airtmp, "C", "a")
dset = relhum(sp_hum, airtmp, apress)
writehdf(hdf, f'Results/relative humidity/relative humidity_{inst:05d}', dset, '-', "a")
del sp_hum, apress, airtmp

velu = grb.grbdset(1, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/wind velocity X/wind velocity X_{inst:05d}', velu, 'm/s', "a")
velv = grb.grbdset(2, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/wind velocity Y/wind velocity Y_{inst:05d}', velv, 'm/s', "a")
dset = np.sqrt(velu ** 2 + velv ** 2)
writehdf(hdf, f'Results/wind modulus/wind modulus_{inst:05d}', dset, "m/s", "a")
del velu, velv

dset = grb.grbdset(5, cutdset=cutdset, fraction=1, hdfout=1)
writehdf(hdf, f'Results/cloud cover/cloud cover_{inst:05d}', dset, "fraction", "a")
dset = grb.grbdset(7, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/precipitation/precipitation_{inst:05d}', dset, "kg/m2", "a")
dset = grb.grbdset(8, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/downward short wave radiation/downward short wave radiation_{inst:05d}',
dset,
"W/m2", "a")
dset = grb.grbdset(9, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/upward short wave radiation/upward short wave radiation_{inst:05d}', dset,
"W/m2", "a")
dset = grb.grbdset(10, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/downward long wave radiation/downward long wave radiation_{inst:05d}',
dset, "W/m2",
"a")
dset = grb.grbdset(11, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/upward long wave radiation/upward long wave radiation_{inst:05d}', dset,
"W/m2", "a")
dset = grb.grbdset(12, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/evaporation/evaporation_{inst:05d}', dset, "kg/m2", "a")
dset = grb.grbdset(13, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/latent heat/latent heat_{inst:05d}', dset, "W/m2", "a")
dset = grb.grbdset(14, cutdset=cutdset, hdfout=1)
writehdf(hdf, f'Results/sensible heat/sensible heat_{inst:05d}', dset, "W/m2", "a")

inst += 1
del dset, grb
print("\n")
return "GOOD JOB"

```

```

def interpolation(outdir, batdir, modset, levels, hdfski):
    model_batims = glob(batdir + "\\BATIM_LV*.dat")
    if modset == "1":
        levels_range = levels
        level_incrmt = 1
    else:
        model_batims.remove(batdir + "\\BATIM_LV1.dat")
        levels_range = levels - 1
        level_incrmt = 2

    for dat in model_batims:
        copyfile(dat, outdir + "\\" + os.path.basename(dat))

    model_batims = glob(outdir + "\\BATIM_LV*.dat")
    model_batims.sort()
    lat = lon = None
    lines = [line for line in open(model_batims[0])]

    for line in lines:
        if 'LATITUDE' in line:
            lat = line[line.find(":") + 1:].strip()
        if 'LONGITUDE' in line:
            lon = line[line.find(":") + 1:].strip()

    if None in (lat, lon):
        print('-' * 80 + '\nERROR - Unable to read model batim\n' + '-' * 80)
        return "ERR06", ()

    with File(hdfski, "r") as hdf:
        latdset = np.array(hdf['/Grid/Latitude'])[0]
        londset = np.array(hdf['/Grid/Longitude']).transpose()[0]
        batdset = np.array(hdf['/Grid/Bathymetry'])

    with open(outdir + '\\BATIM_SKIRON.dat', 'w') as dat:
        dat.write(f'""ILB_IUB          : 1          {len(latdset) - 1}
JLB_JUB          : 1          {len(londset) - 1}
COORD_TIP        : 4
ORIGIN           : 0          0
GRID_ANGLE       : 0.000
LATITUDE         : {lat}
LONGITUDE        : {lon}
FILL_VALUE       : -99.0
<BeginGridData2D>\n''')

        for val in batdset.transpose().flatten():
            dat.write(f' {val:^14}\n')
        dat.write("<EndGridData2D>\n<BeginXX>\n")

        for val in londset:
            dat.write(f' {val:^14}\n')
        dat.write("<EndXX>\n<BeginYY>\n")

        for val in latdset:
            dat.write(f' {val:^14}\n')
        dat.write("<EndYY>\n")

# ----- INTERPOLATION

```

```

print("Interpolating HDF5 file", end="\n\n")
os.chdir(outdir)
for level in range(levels_range):
    if level == 0:
        hdfin = hdfski
        gridin = outdir + '\\BATIM_SKIRON.dat'
    else:
        hdfin = outdir + f'\\ATMFORC_LV{level + level_incrmt - 1}.hdf5'
        gridin = outdir + f'\\BATIM_LV{level + level_incrmt - 1}.dat'

    hdfout = outdir + f'\\ATMFORC_LV{level + level_incrmt}.hdf5'

    with open(outdir + '\\ConvertToHDF5Action.dat', 'w') as dat:
        dat.write(f'<begin_file>
ACTION          : INTERPOLATE GRIDS
TYPE_OF_INTERPOLATION : 1
FATHER_FILENAME   : {hdfin}
FATHER_GRID_FILENAME : {gridin}
OUTPUTFILENAME   : {hdfout}
NEW_GRID_FILENAME : BATIM_LV{level + level_incrmt}.dat
EXTRAPOLATE_2D   : 1
BASE_GROUP        : /Results
POLI_DEGREE       : 4\n<end_file>')

    logs = outdir + '\\interpolation_log.txt', outdir + '\\error.txt'
    command = "Convert2Hdf5.exe > " + logs[0] + " 2> " + logs[1]
    run(command, shell=True)

    if not os.path.isfile(hdfout):
        print('-' * 80 + '\nERROR - Interpolated HDF5 not found\n' + '-' * 80)
        return "ERR07", logs

    error = [line.strip() for line in open(logs[1])]
    if error:
        print('-' * 80 + '\nERROR - Interpolation failed\n' + '-' * 80)
        return "ERR08", logs
os.chdir("../..")
return "GOOD JOB", ()

def skiron(outdir, odate, inputs):
    srcdir = outdir + "\\Skiron"
    logrun = srcdir + "\\Skiron_run_status.log"

    print("-"*80 + "\n" + "SKIRON DATA MODULE".center(80) + "\n" + "-"*80)
    print("WOORKING DIRECTORY :", outdir)
    print("PROCESS DATE      :", odate.strftime("%Y %m %d"))
    print("HDFOUT            :", inputs.get("HDFOUT"))
    print("TSOUT             :", inputs.get("TSOUT"))
    print("-" * 80)

    hdfout = inputs.get("HDFOUT") == "1" and len(glob(opdate.strftime(srcdir + "\\Data HDF\\*%y%m%d.hdf5"))) > 0
    tsout = inputs.get("TSOUT") == "1" and len(glob(opdate.strftime(srcdir + "\\Data TS\\*%y%m%d.dat"))) > 0
    if hdfout or tsout:
        print('-' * 80 + '\nDATA FILE ALREADY AVAILABLE FOR SELECTED DATE\n' + '-' * 80)

```

```

return 1

if not os.path.isdir(srkdir):
    os.mkdir(srkdir)
if not os.path.isfile(logrun):
    logentry(logrun, "DATE STATUS ENDTIME\n", mode="w")
logentry(logrun, str(opdate))

print("Removing old files", end="\n\n")
olds = glob(srkdir + "\\Conversion\\*.hdf5") + glob(srkdir + "\\Interpolation\\*.hdf5")
for old in olds:
    os.unlink(old)

if not os.path.isdir(srkdir + "\\Download"):
    os.mkdir(srkdir + "\\Download")
bkupdir = srkdir + "\\BKUP"
if inputs.get("BKUP") == "1" and not os.path.isdir(bkupdir):
    os.mkdir(bkupdir)

manager = ForcStructure(srkdir, inputs.get("MOHID"), opdate)
manager.datawait(7, 0)

timerun = opdate - timedelta(inputs.get("HND CST") + 1), opdate

# ----- DOWNLOAD GRIB FILES
for inst in range((timerun[1] - timerun[0]).days + 1):
    dir_date = timerun[0] + timedelta(inst)
    dir_dwld = srkdir + dir_date.strftime("\\Download\\%y%m%d")

    status = "GOOD JOB"
    if not os.path.isdir(dir_dwld):
        status = download(dir_dwld, dir_date)

    if "ERR" in status and os.path.isdir(bkupdir + dir_date.strftime("\\%y%m%d")):
        copytree(bkupdir + dir_date.strftime("\\%y%m%d"), dir_dwld)
        status = "GOOD JOB"

    if "ERR" in status:
        print('-' * 80 + dir_date.strftime(f"\nERROR - {status} Download failed for %d/%b/%Y data \n") +
              '-' * 80)
        logentry(logrun, datetime.today().strftime(f" {status} %Y-%m-%d_%H:%M\n"))
        body = opdate.strftime(f"%Y-%m-%d operation failed\n{status} - Skiron download failed")
        mailreport(inputs.get("MAILTO"), "Skiron process ERROR", body, ())
        return 0

dir_dates = []
for folder in glob(srkdir + "\\Download\\*"):
    if os.path.isfile(folder):
        os.unlink(folder)
        continue
    try:
        dir_dates.append(datetime.strptime(os.path.basename(folder), "%y%m%d").date())
    except ValueError:
        rmtree(folder)
        continue

grbs = []
for date in dir_dates:

```

```

if date < datetime.today().date():
    grbs += glob(srcdir + date.strftime("\\Download\\%y%m%d\\*.grb"))
for grb in grbs:
    grb_time = int(os.path.basename(grb).split("_")[-1][:3])
    if grb_time > 23:
        os.unlink(grb)

for date in dir_dates:
    if date >= timerun[0]:
        continue
    folder = srcdir + date.strftime("\\Download\\%y%m%d")
    if inputs.get("BKUP") == "1":
        try:
            copytree(folder, bkupdir + date.strftime("\\%y%m%d"))
        except FileExistsError:
            pass
    rmtree(folder)

if inputs.get("HDFOUT") == "1":
    manager.atmhdfs(inputs.get("MODSET"), inputs.get("LEVELS"))
    manager.std_conversion()

status = conversion(srcdir + "\\Conversion", srcdir + "\\Download", inputs.get("GRID"))
if "ERR" in status[0]:
    logentry(logrun, datetime.today().strftime(f" {status[0]} %Y-%m-%d_%H:%M\n"))
    body = opdate.strftime(f"%Y-%m-%d operation failed\n{status} - Skiron conversion failed")
    mailreport(inputs.get("MAILTO"), "Skiron process ERROR", body, status[1])
    return 0

batdir = inputs.get("PROJDIR") + "\\Sim_Daily\\General Data\\Digital Terrain"
hdfski = srcdir + "\\Conversion\\Skiron.hdf5"

status = manager.mohid_interpolation(srcdir + "\\Interpolation")
if status < 1:
    print("-" * 80 + "\nERROR - Conversion files not found\n" + "-" * 80)
    logentry(logrun, datetime.today().strftime(" ERR05 %Y-%m-%d_%H:%M\n"))
    body = opdate.strftime("%Y-%m-%d operation failed\nERR05 - MOHID files not found")
    mailreport(inputs.get("MAILTO"), "Skiron process ERROR", body, ())
    return 0

status = interpolation(srcdir + "\\Interpolation", batdir, inputs.get("MODSET"),
inputs.get("LEVELS"), hdfski)
if "ERR" in status[0]:
    logentry(logrun, datetime.today().strftime(f" {status[0]} %Y-%m-%d_%H:%M\n"))
    body = opdate.strftime(f"%Y-%m-%d operation failed\n{status} - Skiron interpolation failed")
    mailreport(inputs.get("MAILTO"), "Skiron process ERROR", body, status[1])
    return 0

hdfs = glob(srcdir + f"\\Interpolation\\ATMFORC_LV*.hdf5")
for hdf in hdfs:
    hdfnew = os.path.basename(hdf).split(".")[0] + opdate.strftime("_%y%m%d.hdf5")
    copyfile(hdf, srcdir + "\\Data HDF\\" + hdfnew)

os.chdir(outdir)

print('-' * 80 + "\nSKIRON PROCESS COMPLETED\n" + '-' * 80)
logentry(logrun, datetime.today().strftime(" 1 %Y-%m-%d_%H:%M\n"))

```



```
mailreport(inputs.get("MAILTO"), "Skiron process COMPLETED", opdate.strftime("%Y-%m-%d Files  
available"), 0)  
return 1
```