A Generic Software Framework for Data Assimilation and Model Calibration

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K. C. A. M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 07 juni 2010 om 15.00 uur
door

Cornelis VAN VELZEN

Doctorandus in de wiskunde
geboren te Rijswijk.

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. A.W. Heemink

Copromotor:
Dr. ir. H. X. Lin

Samenstelling Promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. A.W. Heemink,	Technische Universiteit Delft, promotor
Dr. ir. H. X. Lin,	Technische Universiteit Delft, copromotor
Prof. dr. ir. A. E. Mynett,	UNESCO-IHE/Technische Universiteit Delft
Prof. dr. ir. G. S. Stelling,	Technische Universiteit Delft
Prof. dr. ir. J. D. Jansen,	Technische Universiteit Delft
Prof. dr. K. Ponnambalam,	University of Waterloo, Canada
Dr. ir. M. Verlaan,	Technische Universiteit Delft

Dit onderzoek is financieel ondersteund door VORtech bv en het Rijksinstituut voor Kust en Zee, Rijkswaterstaat.

ISBN 978-90-8570-517-8

Typesetting system: LaTeX
Printed in The Netherlands by: Wöhrmann Print Service

A Generic Software Framework for Data Assimilation and Model Calibration

Nils van Velzen

April 24, 2010

Contents

Chapter 1

Introduction

1.1 Background

Simulation models are widely used in various application areas like meteorology, oceanography and chemistry. The use and results of these models play an important role in nowadays live. Best known are probably models for weather forecasting (Figure 1.1) but other models are important as well. Some models are used for providing warning to the population for hazardous situations like storms, floods and environmental accidents. Governments use models in order to predict and investigate the impact of policies, and in engineering, models are used to study new designs.

The quality of models can be improved by combining them with observed data. The observations can be used to calibrate the model. In this way the general performance of the model can be improved. Another way to improve the quality of the predictions that are computed by the model is to combine the model predictions with observations into a combined, updated prediction. This approach is called data assimilation.

Simulation models are often implemented in a computer program. These programs contain thousands to millions of lines of source code and are in general very complex. The size and complexity of these programs is even larger when data assimilation and calibration techniques are added.

The complexity of programs often increases much faster than the complexity of the underlying mathematical models and the used mathematical techniques [Langtangen, 2006]. The development and maintenance of these large and complex programs is extremely expensive. The amount of work is at least linear dependent from the program size and more skillful and therefore more expensive programmers are needed when the program complexity

Figure 1.1: Simulation models play in important role in daily life. For example the weather forecast that is computed using complex simulation models.

increases. This relationship between the complexity of the model, software and development costs is illustrated in Figure 1.2.

A computer program needs to be able to perform various tasks. In case of a simulation model for example the program is able to read and write input and output files, process user input trough a (graphical) user interface, build a mathematical model of the problem, and solve this model.

From Figure 1.2 it can be derived that the sum of the development and maintenance costs of a number of small independent sub-programs is less than the development and maintenance costs of a single large program that contains the functionality of these sub-programs.

Small sub programs with a limited functionality are more likely to be (re)used in multiple contexts. This is an additional advantage and cost reduction of developing small independent sub-programs in stead of large complex programs. Examples of these kind of sub-programs is a generic software library for reading and writing XML files or a numerical library for solving linear algebra problems. Only a limited number of these generic libraries have been developed like the Gnome C XML parser (libxml) [LIBXML, 2009] for handling XML and LAPACK [Anderson et al., 1999] for solving linear algebra problems. These libraries are used in thousands of programs saving the developers of these programs a significant effort in designing, programming and debugging code.

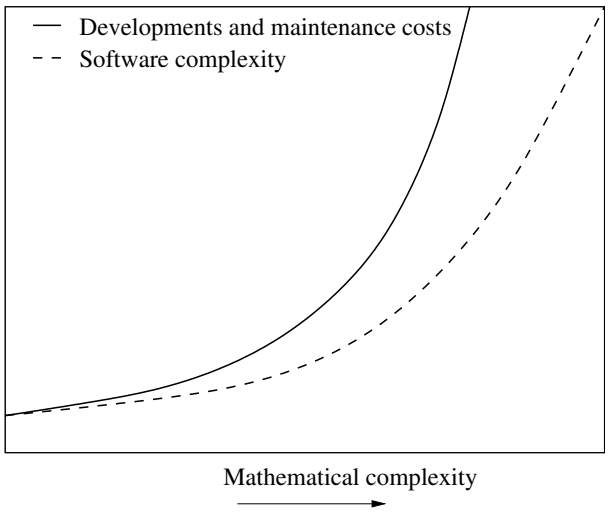

Figure 1.2: Relation between the mathematical complexity of the mathematical model, the software complexity and development costs. The software complexity grows often faster than the complexity of the mathematical model. The development and maintenance costs grow even faster.

1.2 Research objectives

Computer programs that implement simulation models are already very complex. Significantly more complex are programs that implement data assimilation methods and/or calibration methods as well besides a simulation model. The development and maintenance of these kind of programs is even more complex and therefore very expensive.

The main research objective of the work that is presented in this thesis is the design and development of a software framework for data assimilation and model calibration. The purpose of this framework is to make it possible to rapidly compose a data assimilation or model calibration program from various independent sub programs. This approach should not only reduce the program development and maintenance costs and complexity but should as well enable the exchange of implementations of data assimilation methods such that these implementations can be used in combination with various simulation models. The possibility to make arbitrary combinations between models and methods is a significant increment of the functionality compared to a program that contains only a single or a few methods in combination with a single model. Besides this it also reduces the implementation costs

and increases the software reliability by using tested and evaluated methods and sub-programs.

The resulting framework must be a serious alternative for programs containing both the model and the data assimilation or model calibration method. In order to achieve this objective, a number of requirements must be met by the framework. The most important and most challenging requirements are presented in the following paragraphs.

The amount of computational capacity that is necessary to perform simulations, especially when data assimilation or model calibration techniques are used can be enormous. The performance of the framework is very important because the framework has no practical use when large operational models cannot be run due to poor performance of the framework. The design and implementation of the framework must result in a computational performance that is of the same order as existing programs.

The need for support for parallel computing is related to the performance requirement. The amount of computations in a data assimilation or model calibration run can be several orders higher than the amount of computations in a normal simulation run. Parallel computing is therefore a vital necessity for the framework in order to make these runs feasible. Support for parallel computing should not be limited only to the data assimilation and model calibration algorithms. Large simulation models use parallel computing themselves as well. It must therefore be possible to use these parallel models in the framework.

Not only the performance must be comparable to existing programs, but the functionality as well. Using the framework, it must be possible to exactly implement equivalent versions of existing algorithms such that the results of existing software can be reproduced.

The framework should not only be useful for new applications but it must be attractive as well to introduce the framework in existing programs that already implement data assimilation or model calibration methods. The amount of work to migrate existing code into the framework should be relatively limited.

By reusing significant parts of code it must be possible to reduce the programming costs and increase the reliability of the software since code that is reused multiple times contains less errors than new code.

1.3 Research path and outline of this thesis

The framework that is developed as the results of the work is called COSTA (Common Tool Set of Tools for the Assimilation of Data). It must be possible to implement a wide range of data assimilation and model calibration methods in COSTA in combination with dynamic models from various application areas. The introductionary Chapter 2 describes the dynamic model that is considered in this work. Additionally it will give an overview of various popular data assimilation and model calibration methods. Chapter 2 has two purposes; first it provides a reader that is not familiar with data assimilation and model calibration the necessary background, second it gives a sketch of the interaction between the various methods and the dynamic model.

The main research objectives that are formulated in Section 1.2 do not come out of the blue. The objectives were initially not yet specified. The initial step of the research was to initiate a users group. This group of potential users of the software framework was formed from data assimilation experts from various research institutes in the Netherlands. The experience and knowledge of the members of the users group have significantly contributed in formulation of the objectives, initial set up of the main design and making the various choices that have been made during the development. The members of the users group represent a subset of the diverse field of data assimilation and model calibration and they cover various application areas including air quality, hydrodynamic and water quality, ground water modelling and oil reservoir modelling.

The COSTA framework has not followed the path of starting with a complete design and then start the implementation (the waterfall approach [Royce, 1970]). Instead an evolutionary iterative approach is used [Larman, 2004]. In the iterative approach, the system evolves in a large number of iterations. The result of each iteration is a new working version of the software. Each iteration is an individual mini project including requirements analysis, design and testing. The product of an iteration is a so called iteration release, which is a stable, integrated and tested partially complete system. The iteration release often offers new functionality but sometimes, due to new insights parts of the system are changed or even discarded and redeveloped. The development is evolutionary because the overall design is not fully defined at front. The design and requirements evolve during the whole development process. The advantage of this evolutionary approach is that it is possible to use and experiment with the system in an early stage. In this way the iteration releases offer important insight in strong

and weak points of the design of the system that can be further exploit or changed in the following iterations. Chapter 3 will give an overview of the design of the COSTA framework. Chapter 3 also presents the complete set of requirements as they are formulated by the users group and the assessment of a number of alternative frameworks.

The core of this thesis is formed by the three chapters 4, 5 and 6. These chapters contain the following three papers:

- COSTA a problem solving environment for data assimilation applied for hydrodynamical modelling, by Nils van Velzen and Martin Verlaan, published in Meteorologische Zeitschrift, 16:777-793, 2007,

- A problem-solving environment for data assimilation in air quality modelling, by Nils van Velzen and Arjo Segers, published in Environmental Modelling & Software, 2009, DOI: 10.1016/j.envsoft.2009.08.008

- A generic object oriented approach towards parallel computing and combining models in the COSTA framework for data assimilation, by Nils van Velzen, Hai Xiang Lin, Edwin Vollebregt and Erwin Loots, submitted to Scientific Programming.

In these three papers work is presented that provides the answers to the main research objectives.

One of the main objectives is that the COSTA framework should provide more flexibility and functionality than existing operational data assimilation systems. Therefore it must be possible to include the functionality of existing operational systems in COSTA. This property is investigated in an early stage. A first experiment with a large operational model is performed directly after the initial design and implementation of the framework. In the first experiment it was investigated whether it was possible to exactly reproduce the results and functionality of an operational model with data assimilation capabilities. The WAQUA/TRIWAQ shallow water model [Stelling, 1983, Vollebregt et al., 2003] with the RRSQRT-filter [Verlaan and Heemink, 1997b, Verlaan, 1998] was selected as candidate for this first experiment. It was known in advance that the implemented RRSQRT filter code contained numerous model specific issues. For example the way that drying and flooding is handled in the RRSQRT-filter implementation and not in the model code. The experiment consists of two parts. First the model is wrapped such that it can be used in COSTA. The second part concerns the transformation of the existing RRSQRT algorithm into a COSTA RRSQRT algorithm that can be used with other models as well.

The first experiment and the results are described in Chapter 4. Chapter 4 serves as well as a general guide for transforming an existing model such that it can be used in the COSTA framework. Other models like the air quality models LOTOS-EUROS[Schaap et al., 2008] and Chimere [Bessagnet et al., 2008] are transformed for usage in COSTA using the same approach.

The successful use of the WAQUA/TRIWAQ model in COSTA only illustrated that it was possible to substitute an existing data assimilation application with COSTA. In order to show that the framework offers additional functionality a special second case study has been set up. The goal of this second case study is to show that it is easy to combine a model in COSTA with various data assimilation methods. In this way it becomes possible to compare the performance of various methods and select the best suited data assimilation method. This second experiment is used as well to show that off-the-shelf data assimilation methods can be used for an arbitrary model and that it is possible to exchange these methods. A pitfall in the development of a generic framework is the scope of genericity. The WAQUA/TRIWAQ model was therefore not a suitable candidate since it has already been used earlier in the first study that is described in Section 4. Therefore an alternative model, the LOTOS-EUROS air chemistry model is used in this second study. The experiment and the results are discussed in detail in Chapter 5.

Concepts from object oriented programming make it possible to handle parallel computing in the COSTA framework in an elegant way. In Chapter 6 it is explained how concepts from object oriented programming are used to automatically parallelize the computations in the data assimilation and model calibration methods. It is shown that the chosen approach can be used as well for creating composite models from a number of smaller models and to use parallel models in COSTA. The parallel capabilities are illustrated by various experiments with the LOTOS-EUROS model, a large domain decomposition WAQUA/TRIWAQ model and the Chimere model.

This thesis concludes with Chapter 7. In this section the main results and work are summarized and a roadmap is set out for future development and research.

Chapter 2

Data assimilation and model calibration

2.1 Introduction

Data assimilation techniques are used to improve the predictions of a dynamic model using observations. Depending on the application area different sources of observations are available. Observations can be gathered by earth observation satellites. These satellites can be used to measure various aspects of the earth and the atmosphere among others cloud systems, land and water temperature, snow cover, ocean currents, vegetation, ice fields and air pollution. Besides satellites other observation devices can be used as well like buoys for measuring water level and current speed and direction and air quality measurement stations. The source, kind, amount and quality of the observations differs as well. The data assimilation techniques that are used to combine these observations with the model predictions can be used in a wide range of application areas and kinds of observations.

Another group of techniques that uses observations to improve the quality of the predictions of dynamic models are calibration techniques. In model calibration, not the prediction of the model is adjusted but the model itself is changed using the observations. In general, data assimilation techniques are used in an on-line operational setting and calibration techniques are used to calibrate a model before it will be used operational.

The basic concepts from dynamic models, data assimilation and model calibration methods are presented in this chapter. The goal of this chapter is to provide the necessary background into the formulation of the model and the data assimilation and model calibration methods.

The basic concepts of dynamic simulation models that are considered in this thesis are described in Section 2.2. Then in Section 2.3 a number of popular data assimilation methods are presented. This section will discuss some empirical methods, the Kalman filter type methods and variational methods. The reader is referred to [Bouttier and Courtier, 1999, Cooper et al., 1999] and [Evensen, 2006] for a more detailed introduction to data assimilation methods.

Section 2.4 describes the model calibration techniques that are currently available in COSTA. An more detailed description of methods can be found in [Edgar and Himmelblau, 1989].

It is not the intention of this chapter to give a thorough overview of the field of data assimilation. It provides an overview of the range of data assimilation methods that are already or should be supported by COSTA. Additional, this chapter will provide the reader that is not very familiar to data assimilation methods and model calibration methods the necessary background for understanding the following chapters of this thesis. However it does provide an overview and insight into the various (mathematical) operations that are used in the formulations of a wide range of data assimilation and model calibration methods as discussed in Section 2.5.

2.2 Dynamical simulation models and observations

A mathematical model is a representation of a system in a mathematical language. Here a system is a set of interacting or interdependent entities, real or abstract, forming an integrated whole. A system that is described in a mathematical language can be investigated in a systematic way using generic mathematical tools. In order to create a mathematical model it is necessary to identify and understand the mechanisms behind the system. A mathematical model has the form of a set of variables and relationships between them. Mathematical models can be used in many application areas including physics, biology, earth science, meteorology, electrical engineering and economics. Mathematical models can be classified in various ways for example linear/non-linear, deterministic/stochastic and static/dynamic models.

Dynamic models describe systems in which time plays a role. These models can be used to produce forecasts, like the weather prediction, air quality prediction and waterlevel predictions of rivers and seas.

The variables in a dynamic model can be classified in various groups. Often all variables of a single group are represented by a vector. The general

form of the dynamic models that are considered in this thesis is the following:

$$\frac{d\mathbf{x}(t)}{dt} = \mathcal{M}\left(\mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \mathbf{w}(t), t\right) \tag{2.1}$$

where the variables are classified in four groups: the state variables $\mathbf{x}$, parameters $\mathbf{p}$, forcings or input variables $\mathbf{u}$ and the random variables $\mathbf{w}$, which is a Brownian motion in the continuous formulation. The dynamic model is denoted by the operator $\mathcal{M}$ and time by t. All variables are time dependent except for the model parameters.

Most models that are implemented make use of a time discretization with $t_i = t_0 + i\Delta t$ with $i \in \mathbb{N}^+$. The general non-linear model timestep can then be written as

$$\mathbf{x}_{k+1} = M_k\left(\mathbf{x}_k, \mathbf{p}, \mathbf{u}_k, \mathbf{w}_k\right), \tag{2.2}$$

where the sub-script denotes the time index of the variable e.g. $\mathbf{x}_k = \mathbf{x}\left(t_k\right)$. The random variable $\mathbf{w}_k$ for the time descrete formulation is assummed to have zero mean and covariance $\mathbf{Q}_k$. Linear models are an important sub-class of dynamic models. Linear models are the least complex class of models. Most data assimilation methods and model calibration methods for non-linear models are extensions of the methods for linear models. A discrete linear model can be written in matrix form as:

$$\mathbf{x}_{k+1} = \mathbf{M}_k\mathbf{x}_k + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k \tag{2.3}$$

Complex systems can in general not be described exactly by a model. When a model is developed a trade off needs to be made between complexity and accuracy of the model. Therefore, models contain in general simplifications and are not able to exactly describe the system that is modeled. The model operator M_k will therefore not be exact and contains errors, differences between the real system and the model. Besides the errors in the model operator there are other sources of errors. For instance the numerical errors that are the result of performing the computations on a computer with a finite precision representation of the variables and errors in the initial model state, parameters and forcings of the model.

Besides a model of the real system, observations are often available as well. Observations of the system that are available at time t are denoted by $\mathbf{y}(t)$. The corresponding observation operator is:

$$\mathbf{y}(t) = H\left(t, \mathbf{x}(t)\right) + \mathbf{v}(t) \tag{2.4}$$

the operation H interpolates and transforms the state variables to the observations. The observation error $\mathbf{v}(t)$ includes both the instrumental and

representation error and is supposed to have zero mean and covariance $\mathbf{R}(t)$. In the case timediscretization is used, the operation is written as:

$$\mathbf{y}_k = H_k\left(\mathbf{x}_k\right) + \mathbf{v}_k \tag{2.5}$$

and for linear operators as:

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k \tag{2.6}$$

with error covariance matrix $\mathbf{R}_k$.

The differences between the system and the results of the dynamic model can be reduced by using these observations of the system. Two classes of methods that combine observations with a dynamic system are considered in this thesis. The first category are the data assimilation methods and the second category are the calibration methods.

An short introduction on these methods will be given in the Sections 2.3 and 2.4.

2.3 Data assimilation techniques

2.3.1 Introduction

Data assimilation methods combine a model prediction with observations in order to create a more accurate prediction. The algorithm improves the state of the model $\mathbf{x}_k$ by use of the observations in the so called analysis step.

Data assimilation techniques can be divided into two groups: sequential data assimilation and non-sequential data assimilation. Sequential data assimilation techniques start with a guess of the initial state $\mathbf{x}_0$. This state is propagated forward in time by the dynamic model to the moment that observations are available. This is the forecasted state $\mathbf{x}_k^f$. The differences between the observations and the forecasted state are used in the analysis step to compute the analyzed state $\mathbf{x}_k^a$. The analyzed state is then propagated to the next time where observations are available and the procedure is repeated. This process is illustrated in figure 2.1.

The other group of data assimilation methods, the non-sequential data assimilation methods do not only use observations at a single time instance in the analysis but all observations in a time window. The observations are used to improve the background state $\mathbf{x}^b$ in order to improve the model results in the time window. This procedure is illustrated in Figure 2.2. There is an important distinction between the background state $\mathbf{x}^b$ and the

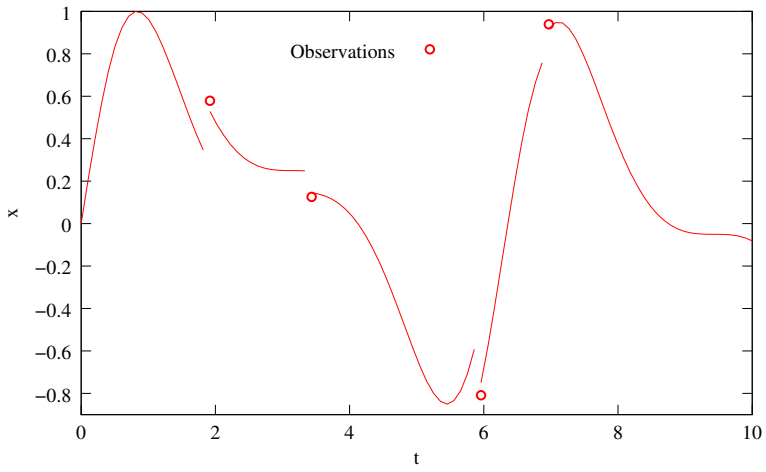

Figure 2.1: Example of a sequential data assimilation method. The correction is made at the moments that observations are available.

forecasted state $\mathbf{x}^f$. The forecasted state already contains information from all observations in the past, where the background state does not.

A large number of data assimilation methods has been developed and applied. These methods differ in their accuracy, suitability and amount of computations that have to be performed for computing an analysis. It is not in the scope of this chapter to give a thorough overview of the different data assimilation techniques. Only a few examples are described to give a basic understanding of the differences between the various methods and their application. These methods vary from computational inexpensive and generally inaccurate to computationally expensive and accurate.

2.3.2 BLUE

This section presents the basic formulation and notation of the data assimilation problem and will conclude with the BLUE estimator. This estimator is important since many data assimilation methods are derived from it. The model state as computed by the model without applying data assimilation is called the background state and denoted by $\mathbf{x}^b$. The state that is computed after the analysis is denoted by $\mathbf{x}^a$ and the true state of the system is denoted by $\mathbf{x}^t$. The state that is predicted by the model is an approximation of the true state. The difference between the true state and the background state is the background error, defined by

$$\epsilon^b = \mathbf{x}^b - \mathbf{x}^t. \tag{2.7}$$

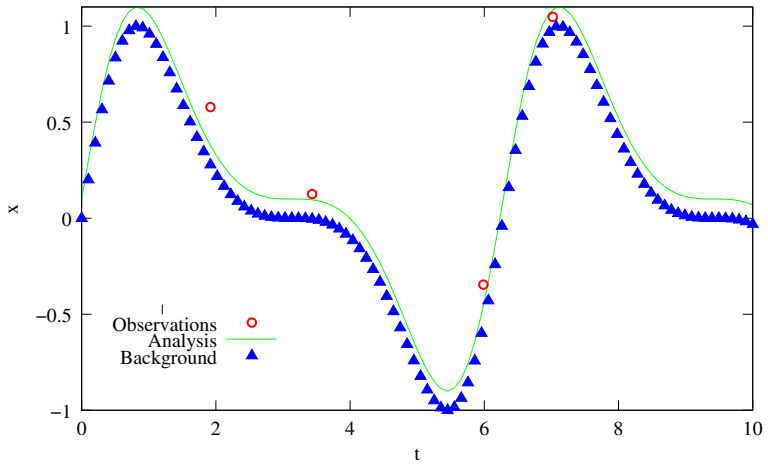

Figure 2.2: Example of a non-sequential data assimilation method. All states are improved in a time window in which observations are available. This method yields a more continuous solution in time.

The errors are caused by various sources like simplifications in the model operator, errors in the forcings and initial state-vector. If we assume that all these sources are stochastic, than ϵ^b is a stochastic variable as well. A popular choice is to assume a Gaussian distribution for the background error. The background error covariance matrix is defined by

$$\mathbf{P}^b = \mathrm{E}\left(\left(\epsilon^b - \mathrm{E}(\epsilon^b)\right)\left(\epsilon^b - \mathrm{E}(\epsilon^b)\right)^T\right). \tag{2.8}$$

The observations of the system that are available are denoted by $\mathbf{y}$. The observations cannot be directly related to elements of the state. The observation operator makes the relation from the state-space to the observation space. When this relation is linear than the true observation (without any error) is given by

$$\mathbf{y}^t = \mathbf{H}\mathbf{x}^t \tag{2.9}$$

or in the non-linear case by

$$\mathbf{y}^t = H\left(\mathbf{x}^t\right) \tag{2.10}$$

Like the model state, the measurements are likely to contain errors. A Gaussian measurement error is defined by

$$\epsilon^o = \mathbf{y} - \mathbf{H}\mathbf{x}^t, \tag{2.11}$$

or for a non-linear observation operator

$$\epsilon^o = \mathbf{y} - H(\mathbf{x}^t), \tag{2.12}$$

with covariance matrix

$$\mathbf{R} = \mathrm{E}\left(\left(\epsilon^o - \mathrm{E}(\epsilon^o)\right)\left(\epsilon^o - \mathrm{E}(\epsilon^o)\right)^T\right). \tag{2.13}$$

The differences between $\mathbf{H}\mathbf{x}^b$ and the observations $\mathbf{y}$ are called innovations and are denoted by

$$\mathbf{d} = \mathbf{y} - \mathbf{H}\mathbf{x}^b. \tag{2.14}$$

Based on these assumptions we can derive the Best Linear Unbiased Estimator (BLUE) of $\mathbf{x}^t$. This estimator is very important since many algorithms are derived from it. For the case that the errors ϵ^b and ϵ^o are unbiased and not correlated ($\mathrm{E}(\epsilon_b \epsilon^o) = 0$). The BLUE analysis is given by

$$\mathbf{x}^a = \mathbf{x}^b + \mathbf{K}\left(\mathbf{y} - \mathbf{H}\mathbf{x}^b\right) \tag{2.15}$$

with

$$\mathbf{K} = \mathbf{P}^b\mathbf{H}^T\left(\mathbf{H}\mathbf{P}^b\mathbf{H}^T + \mathbf{R}\right)^{-1} \tag{2.16}$$

The analysis error covariance matrix $\mathbf{P}^a$ is minimized by the BLUE analysis and is given by

$$\mathbf{P}^a = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\mathbf{P}^b\,(\mathbf{I} - \mathbf{K}\mathbf{H}) + \mathbf{K}\mathbf{R}\mathbf{K}^T. \tag{2.17}$$

Equation 2.17 is valid for all choices of $\mathbf{K}$. If $\mathbf{K}$ is optimal and satisfies Equation 2.16 then Equation 2.17 can be simplified to

$$\mathbf{P}^a = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\mathbf{P}^b. \tag{2.18}$$

Until now we have derived the BLUE estimator given the background state and know error statistics. This method does not include time and the model operator. This concept can be extended for a sequential data assimilation context as well where time and the model play a role. These methods will update the forecasted state $\mathbf{x}^f$ and optionally the error statistics at each time that observations are available. Various methods that include time and model are presented in the following sections.

2.3.3 Emperical linear interpolation and Optimal interpolation

The emperical methods are a class of methods that have been polular from the late 1950's to about 1980 [Cooper et al., 1999]. These methods are computational inexpensive and easy to understand. Three popular emperical linear interpolation methods are the Cressman Analysis [Cressman, 1959], nudging and optimal interpolation [Kalnay, 2002].

The Cressman analysis updates the points of the forecasted state by a linear combination of the inovations, the difference between the forcasted values and the observations. Each element of the state is updated according to

$$\mathbf{x}_k^a = \mathbf{x}_k^f + \mathbf{K}\mathbf{d}_k \tag{2.19}$$

where $\mathbf{d}_k$ denotes the innovation $\mathbf{d}_k = \mathbf{y}_k - \mathbf{H}\mathbf{x}_k^f$.

The matrix $\mathbf{K}$ is weighting matrix of weighting factors $k(i,j)$ The weighting factors are chosen based on the distance between the element of the state $x(i)$ and the observation $y(j)$. The state will be updated to the observed value near the observation and will not be updated for elements that lie far from the observation location. There are many ways to choose the weighting values.

A popular variation is nudging or Newtonian Relaxation Scheme where the state is slightly changed towards the observations. The values of $w(i,j)$ can be smaller that 1 at the observation locations. This approach avoids large changes to the model that can cause problems in the model. The coefficients are determined a priori but can vary in time.

Both Cressman analysis as nudging assume that the observations represent the true value and do not contain any measurement errors.

Optimal interpolation (OI) also called statistical interpolation is a method that uses the background error covariance of the model and the error covariance of the observations in order to compute the weights $w(i,j)$. This method is an approximation of the best linear unbiased estimator and is based on the idea that only observations in the neighborhood of a state element determine the analysis state increment as illustrated in Figure 2.3. The global analysis problem is therefore split up into blocks. This reduces the size of the problem that needs to be solved in the analysis as well as the computational time. When the analysis of various blocks are combined that overlap it might be possible that jumps occurs in the model state. An other disadvantage of OI is that it is difficult to use observations with complex observation operators because the background error of the model must be computed.

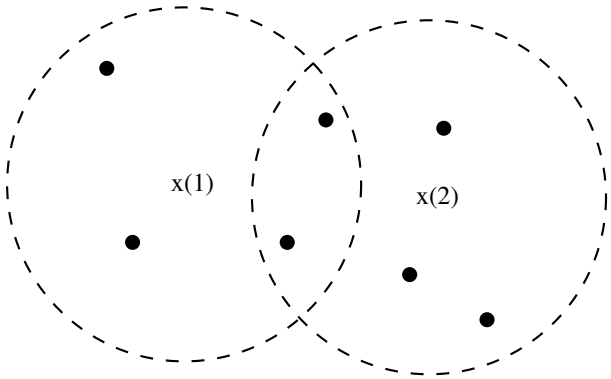

Figure 2.3: Example of selecting the observations that are used for computing the analysis of two elements of the state-vector. Two of the observations are selected for both state elements.

2.3.4 Kalman filter methods

The classical Kalman filter (KF) algorithm is a sequential data assimilation method. The analysis step of this method only uses the state of the model and the available measurements at a single time instance.

A single timestep with the Kalman filter for linear systems is computed according to

$$\begin{aligned}
\mathbf{x}^f_{k+1} &= \mathbf{M}_k \mathbf{x}^a_k && (2.20)\\
\mathbf{P}^f_{k+1} &= \mathbf{M}_k \mathbf{P}^a_k \mathbf{M}^T_k + \mathbf{Q}_k && (2.21)\\
\mathbf{x}^a_{k+1} &= \mathbf{x}^f_{k+1} + \mathbf{K}_{k+1}\left(\mathbf{y}_{k+1} - \mathbf{H}_{k+1}\mathbf{x}^f_{k+1}\right) && (2.22)\\
\mathbf{K}_{k+1} &= \mathbf{P}^a_{k+1}\mathbf{H}^T_{k+1}\left(\mathbf{H}_{k+1}\mathbf{P}^a_{k+1}\mathbf{H}^T_{k+1} + \mathbf{R}\right)^{-1} && (2.23)\\
\mathbf{P}^a_{k+1} &= \left(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\right)\mathbf{P}^a_k && (2.24)
\end{aligned}$$

where $\mathbf{M}_k$ denotes the model operator and $\mathbf{Q}_k$ the error covariance matrix of added noise to the model.

The classical Kalman filter can only be used for linear models. The extension of the classical Kalman filter is the Extended Kalman Filter (EKF). The model operator $\mathbf{M}_k$ is replaced by the tangent linear approximation (TLM) of the non-linear model $M(\mathbf{x}_k)$. The TLM is a matrix of partial derivatives. The j-th column of TLM is defined by:

$$\mathbf{M}_{:,j} = \frac{\partial M_k\left(\mathbf{x}_k\right)}{\partial (x_k)_j\left(t_k\right)}. \qquad (2.25)$$

Where $(x_k)_j$ denotes the j-th element of the vector $\mathbf{x_k}$. Both KF as EKF can only be used for dynamic models with a small state size because the matrix $\mathbf{P}$ is too large to handle for large state spaces. Another complication of EKF is the need for a tangent linear model.

Various algorithms are developed that avoid these problems and can be used for dynamic models with a large state space. These methods approximate the error covariance $\mathbf{P}$ and do not use the TLM. Examples of these algorithms are EnKF [Evensen, 2003], RRSQRT [Verlaan and Heemink, 1997b, Verlaan, 1998], EnSRF [Whitaker and Hamill, 2002] and COFFEE [Heemink et al., 2001]. These algorithms are discussed in more detail in Section 5.4.

A Kalman filter can be used as well for the calibration of models. In order to be able to calibrate the model using a Kalman filter it is neccesary to extend the model state $\mathbf{x}^m$ by adding the parameters $\mathbf{x}^p$ that have to be calibrated according to

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}^m \\ \mathbf{x}^p \end{bmatrix} \tag{2.26}$$

and the stochastic model then becomes

$$\mathbf{x}_{k+1} = \begin{pmatrix} M\left(\mathbf{x}_k^m, \mathbf{x}_k^p, \mathbf{u}_k, \mathbf{w}_k, t\right) \\ \mathbf{x}_k^p + \mathbf{v}_k \end{pmatrix}, \tag{2.27}$$

where the noise $\mathbf{v}$ is added to the parameters. The model parameters ($\mathbf{x}^p$) are then calibrated by the Kalman filter that is appied on this extended model. It must be noted that the parameters are changed during the simulation as a result of the added noise and filter updates. Updating parameters during the run is not always trivial and can therefore not be implemented easily for some models. In that case it is probably better to use one the optimization methods that are discussed in Section 2.4 that do not require the parameters to be changed during the model run.

2.3.5 Kalman smoother

A Kalman smoother [Anderson and Moore, 1979, Stephen et al., 1994, Evensen and van Leeuwen, 2000] is an extension of a Kalman filter that makes it possible to improve the model state using observations from the future as well.

A Kalman filter or one of the derived methods like EnKF, computes the estimate of the current state given all observations from the past. The general idea of a Kalman smoother is to use the observations from future as

well. This is accomplished by extending the state of the model with values of the state from the past.

There are three main smoothing strategies. Fixed time smoothing is a method where observations from future and past are used to estimate the system at one particular time. Fixed interval smoothing is similar to 4D-var, a variational data assimilation method that is presented in Section 2.3.6 where the observations of given timewindows are used to improve the state vector. The third strategy is fixed lag smoothing where a moving time window is used to improve the system.

The overall state-vector $\mathbf{x}^s$ for a fixed lag smoother of length n is given by

$$\mathbf{x}_k^s = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_{k-1} \\ \mathbf{x}_{k-2} \\ \vdots \\ \mathbf{x}_{k-n} \end{bmatrix}, \tag{2.28}$$

and the corresponding model by

$$\mathbf{x}_{k+1}^s = \begin{pmatrix} M\left(\mathbf{x}_k, \mathbf{x}_k^p, \mathbf{u}_k, \mathbf{w}_k, t\right) \\ \mathbf{x}_k \\ \vdots \\ \mathbf{x}_{k-n+1} \end{pmatrix} \tag{2.29}$$

By extending the model with state variables from the past it is possible to use the observations up to $n+1$ timesteps for updating the model state. If n is chosen big enough than all observations can be used to improve the value of the initial state $\mathbf{x}_0$.

2.3.6 Variational methods

An other class of methods are the variational methods. In this section we will discuss variational methods for finding the optimal value $\mathbf{x}^a$. The variational methods are based on a different but equivalent formulation of the BLUE and are stated in the form of a minimization problem. The goal is to find the (initial) state $\mathbf{x}^a$ that minimizes the object function:

$$J(\mathbf{x}) = \left(\mathbf{x} - \mathbf{x}^b\right)^T \left(\mathbf{P}^b\right)^{-1} \left(\mathbf{x} - \mathbf{x}^b\right) + (\mathbf{y} - \mathbf{H}\mathbf{x})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{H}\mathbf{x}) \tag{2.30}$$

3D-var is a data assimilation method that computes the analysis by minimizing Equation 2.30. The gradient of the criterion 2.30 is given by

$$\nabla J\left(\mathbf{x}\right) = 2\left(\mathbf{P}^{b}\right)^{-1}\left(\mathbf{x}-\mathbf{x}^{b}\right) - 2\mathbf{H}^{T}\mathbf{R}^{-1}\left(\mathbf{y}-\mathbf{H}\mathbf{x}\right) \tag{2.31}$$

and is zero at the minimum. The minimization problem is in general not completely solved but an approximation is computed by performing a number of iterations of a minimization algorithm such that

$$|\nabla J(\mathbf{x})| < \epsilon \tag{2.32}$$

for some predefined tolerance ϵ. The iterative process is illustrated in Figure 2.3.6.

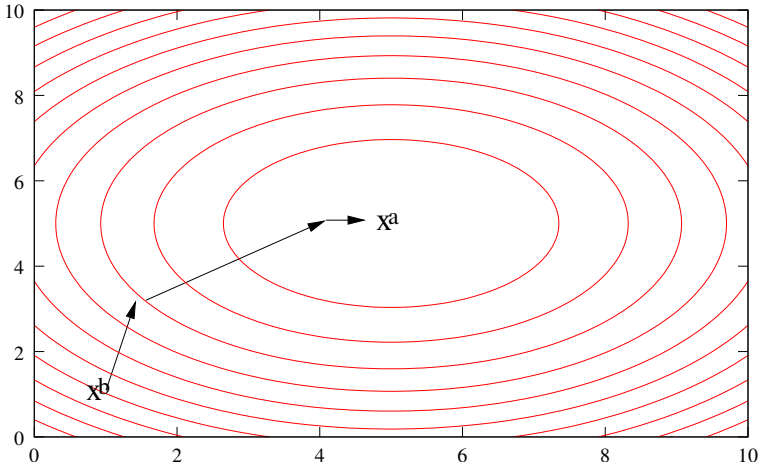

Figure 2.4: Example of the minimization process in a 3D-var algorithm. The contour lines represent the values of $J(\mathbf{x})$.

The advantage of 3D-var is that it is quite straight forward, it can handle complex observation operators, as long as the TLM of the observation operator is available. However, it is necessary to construct a positive definite background error covariance matrix $\mathbf{P}^{b}$ and specify error covariances in the observation sub-space in order to be able to use the 3D-var method.

4D-var is an non-sequential data assimilation method that can be seen as an extension of 3D-var. In 4D-var all observations are taken into account in a given time interval. The non-linear object function that is minimized

in the 4D-var algorithm is given by

$$J(\mathbf{x}_0) = (\mathbf{x}_0 - \mathbf{x}^b)^T \left(\mathbf{P}^b\right)^{-1} (\mathbf{x}_0 - \mathbf{x}^b) + \sum_{k=0}^{n} (\mathbf{y}_k - \mathbf{H}_k \mathbf{x}_i)^T \mathbf{R}_k^{-1} (\mathbf{y}_k - \mathbf{H}_k \mathbf{x}_k). \quad (2.33)$$

Note that all values of $\mathbf{x}_k$ can be expressed in $\mathbf{x}_0$ by

$$\mathbf{x}_k = M_{0\to k-1}(\mathbf{x}_0) = M_{k-1}(M_{k-2}(...M_0(\mathbf{x}_0.))) \quad (2.34)$$

Where M_k denotes the nonlinear timestep operator according to Equation 2.2 with known parameters and forcings. Using the TLM of the model operator and observation operator it is possible to derive the following equation for the foreward sensitivities:

$$\mathbf{y}_k - H_k M_{0\to k-1}\mathbf{x}_0 \approx \mathbf{y}_k - \mathbf{H}_k \mathbf{M}_{0\to k-1}\mathbf{x}_0. \quad (2.35)$$

The object function 2.33 and the corresponding gradient can be computed in the following way: In a single forward run compute all values of $\mathbf{x}_i$ for $i = 0..n$ and the corresponding departures multiplied by the inverse of the observation error covariance $\mathbf{R}_i$ according to

$$\mathbf{d}_k = \mathbf{R}_k^{-1}(\mathbf{y}_k - \mathbf{H}_k(\mathbf{x}_k)). \quad (2.36)$$

The object function can then be computed by:

$$J(\mathbf{x}_0) = (\mathbf{x}_0 - \mathbf{x}^b)^T \left(\mathbf{P}^b\right)^{-1} (\mathbf{x}_0 - \mathbf{x}^b) + \sum_{k=0}^{n} (\mathbf{y}_k - \mathbf{H}_k(\mathbf{x}_k))^T \mathbf{d}_k. \quad (2.37)$$

The gradient is in this case given by

$$\nabla J(\mathbf{x}_0) = 2\left(\mathbf{P}^b\right)^{-1}\left(\mathbf{x}_0 - \mathbf{x}^\mathbf{b}\right) - 2\sum_{k=0}^{n} \mathbf{M}_0^T ... \mathbf{M}_{k-1}^T \mathbf{H}_k^T \mathbf{d}_k \quad (2.38)$$

There is an additional requirement for using 4D-var compared to 3D-var. In order to efficiently compute the gradient (Equation 2.38) it is necessary to have the adjoint model operator $\mathbf{M}_i^T$ that appears in Equation 2.38. The construction of this operator is far from trivial for complex dynamic models and involves writing an adjoint version of all (computational) routines of the model code. Even using software tools like OpenAD and TAF [Naumann et al., 2009, Giering and Kaminski, 2000] this is still an significant amount of work.

A gradient based minimization method is often used the 3D- and 4D-var minimization method. Some of these methods will be discussed in Section 2.4 because they are used for the calibration of models as well.

2.4 Calibration

2.4.1 Introduction

Model calibration techniques are used in order to find the best set of model parameters $\mathbf{p}$ for a (dynamic) model such that the output of the model matches best to the system. Model calibration is generally performed off-line.

The calibration methods that are considered in this section try to find the model parameters $\mathbf{p}$ such that it minimizes an object function that measures the misfit between the dynamic model and the system.

The weighted least square difference between the model predictions and observations is an example of such an object function and given by

$$f(\mathbf{p}) = \sum_{k=0}^{n} \left(\mathbf{y}_k - \mathbf{H}_k \mathbf{x}_k^{\mathbf{p}}\right)^T \mathbf{W}_k \left(\mathbf{y}_k - \mathbf{H}_k \mathbf{x}_k^{\mathbf{p}}\right). \tag{2.39}$$

where $\mathbf{x}^{\mathbf{p}}$ denotes the model state computed by the model with parameters $\mathbf{p}$ and a weighting matrix $\mathbf{W}_k$ e.g. based on the model errors in what case we can substitite it by $\mathbf{R}_k^{-1}$.

The calibration problem can be illustrated using a simple 1-D advection model. This model transports the forcing that is specified at the left side of the domain with a constant speed v from left to right as illustrated in Figure 2.5.

Assume for the calibration model that the frequency of the model forcings $\mathbf{u}(t_k) = 1 + \sin\left(\frac{2\pi}{p} t_k\right)$ is specified by a single parameter p. The exact value of p is not known but observations at different locations and moments are available. The solution of the advection model for an initial choice of $\mathbf{p}$ together with the observations at a single time instance are plotted in Figure 2.6. A calibration algorithm will try to find a suitable value for $\mathbf{p}$ such that the frequency matches to the observed values. This solution is given by the dotted line in Figure 2.6. Note that this simple 1-D advection can be very hard to tackle due to the periodic behavior of the model especially when the observations are sparse in time. If the errors between the observations and the model are not caused by a wrong frequency but e.g. as a result of a phase shift it is possible that the calibration method will end up with a completely wrong value of p. Therefore it is important to carefully select those model parameters for calibration that are suitable for reducing the differences between the model and observations.

Model calibration is a minimization problem like 3D- and 4D-var. There are however some differences. Calibration does in general not involve the

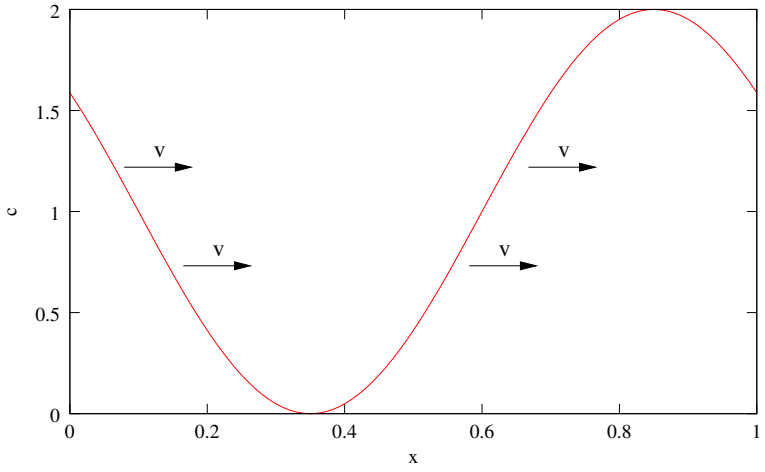

Figure 2.5: Solution of the advection model at $t = 0.4$, with forcing $\mathbf{u}(t_k) = 1 + sin\left(\frac{2\pi}{10}t_k\right)$ at the left of the domain. The advection model moves the initial scalar field with constant speed $v = 1$.

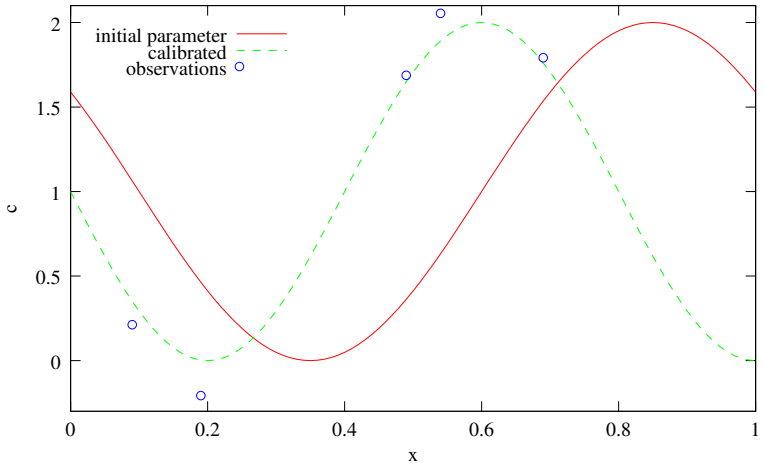

Figure 2.6: Calibration of the frequency of the forcings of the 1-D advection model. A single time instance is plotted for the initial parameter, the calibrated parameter and the observed values.

background error covariance matrix $\mathbf{P}^b$ which needs to be constructed for 3D- and 4D-var. The minimization problem that has to be solved for the calibration can be quite more complex than the 3D-var and 4D-var problems. Since the model parameters $\mathbf{p}$ often have a physical meaning and a valid value should often be in some predefined range. This can be specified by adding inequality constraints to the minimization problem.

Data assimilation algorithms generally concern improving the results of only a single scenario. With a scenario is meant a simulation in a given time window $t_0..t_f$ with a single set of forcings $\mathbf{u}_t$ and initial model-state $\mathbf{x}_0$.

The objective of model calibration can be to improve the parameters such that the model performs better for multiple scenarios. An example of a calibration that involves multiple scenarios is the following; assume a river model is to be calibrated. In the calibration the bottom roughness is adjusted. There are two sets of observations available one set for a period in the summer period and one in the winter period. The parameters must be selected such that the model is improved both for the winter as summer scenario. An evaluation of the object function will then consists of at least running two scenarios.

Model calibration algorithms are often very computational demanding. The computation of the object function involves at least a single model run of the dynamic model.

The calibration methods that are available in COSTA will be described in the following sections.

2.4.2 Gradient methods

A special class of calibration methods are the so called gradient methods. For these methods it is not only needed to be able to compute the value of the object function $f(\mathbf{p})$ but the corresponding gradient

$$\nabla f(\mathbf{p}) = \left(\frac{\partial f}{\partial p_1}, \frac{\partial f}{\partial p_2}, ..., \frac{\partial f}{\partial p_n} \right)^T \tag{2.40}$$

as well. These algorithms are in general more robust and need less overall iterations to converge to a (local) solution than the gradient free methods like the Simplex method that is discussed in Section 2.4.5. A drawback of these methods is the need of a gradient.

When the number of parameters is limited it is possible to approximate the gradient by finite difference,

$$\nabla_i f(\mathbf{p}) \approx \epsilon^{-1} \left(f(\mathbf{p} + \mathbf{e}_i \epsilon) - f(\mathbf{p}) \right), \tag{2.41}$$

where $\mathbf{e}_i$ denotes the i-th unit vector. If some of the parameters are (almost) independent it is even possible to reduce the number of model runs in the finite difference approximation of the gradient by disturbing multiple independent parameters in a single run. For a small number of independent groups of parameters it is still feasible to compute the gradient by finite difference but for a large number of parameters it is not.

For larger numbers of parameters it is necessary to compute the gradient in a more efficient way e.g. by using the adjoint of the dynamic model.

There are a vast number of gradient based algorithms that can be used for calibration. In the next two sections (2.4.3 and 2.4.4) two of these methods, that are implemented in COSTA are described.

2.4.3 The conjugate gradients method

The problem of minimization of a multivariable function is usually solved by determining a search direction vector and solve it as a line minimization problem. If $\mathbf{p}$ is a vector containing the variables to be determined and $\mathbf{h}$ is the vector of search direction, at each iteration step the minimization problem of the function f is formulated as to find the step size λ that minimizes $f(\mathbf{p} + \lambda\mathbf{h})$. At the next iteration, $\mathbf{p}$ is replaced by $\mathbf{p} + \lambda\mathbf{h}$ and a new search direction is determined. Different methods basically propose different ways of finding the search direction.

The conjugate gradient method is an algorithm for finding the nearest local minimum of a function which uses conjugate directions for going downhill. Two vectors $\mathbf{u}$ and $\mathbf{v}$ are said to be conjugate (with respect to a matrix $\mathbf{A}$) if

$$\mathbf{u}^T\mathbf{A}\mathbf{v} = 0, \tag{2.42}$$

where in the minimization problem, $\mathbf{A}$ is typically the Hessian matrix of the cost function. In the conjugate gradient methods, the search direction is somehow constructed to be conjugate to the old gradient.

The two most important conjugate gradient methods are the Fletcher-Reeves [Fletcher and Reeves, 1964] and the Polak-Ribierre methods [Press et al., 1989]. These algorithms calculate the mutually conjugate directions of search with respect to the Hessian matrix of the cost function directly from the function and the gradient evaluations, but without the direct evaluation of the Hessian matrix. The new search direction $\mathbf{h}_{i+1}$ is determined by using

$$\mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i\mathbf{h}_i \tag{2.43}$$

where $\mathbf{h}_i$ is the previous search direction, $\mathbf{g}_{i+1}$ is the negative of local gradient at iteration step $i + 1$, while γ_i is determined by using the following

equations for Fletcher-Reeves and the Polak-Ribierre methods respectively:

$$\gamma_i = \frac{\langle g_{i+1}, g_{i+1} \rangle}{\langle g_i, g_i \rangle} \tag{2.44}$$

$$\gamma_i = \frac{\langle (g_{i+1} - g_i), g_{i+1} \rangle}{\langle g_i, g_i \rangle} \tag{2.45}$$

where $\langle \mathbf{u}, \mathbf{v} \rangle$ is used to denote the innerproduct between $\mathbf{u}$ and $\mathbf{v}$. If the vicinity of the minimum has the shape of a long, narrow valley, the minimum is reached in far fewer steps than would be the case using the steepest descent method, which makes use of minus of the local gradient as the search direction.

A line search method is used in order to find a suitable value for λ such that is approximately minimizes $f(\mathbf{p} + \lambda \mathbf{h})$.

This method needs a gradient of the object function which has to be approximated using finite difference when no adjoint of the model is available.

2.4.4 The LBFGS method

For the problem of minimizing a multivariable function quasi-Newton methods [Han, 1977, Spellucci, 1993] are widely employed. These methods involve the approximation of the Hessian matrix (or its inverse) of the object function. The LBFGS (Limited memory-Broyden-Fletcher-Goldfarb-Shanno) [Nocedal, 1990, Byrd et al., 1994] method is basically a method to approximate the Hessian matrix in the quasi-Newton method of optimization. It is a variation of the standard BFGS method.

$$\mathbf{p}_{i+1} = \mathbf{p}_i - \lambda_i \mathbf{H}_i \mathbf{g}_i, i = 0, 1, 2, \cdots \tag{2.46}$$

where λ_i is a steplength, $\mathbf{g}_i$ is the local gradient of the cost function, and $\mathbf{H}_i$ is the approximate inverse Hessian matrix which is updated at every iteration by means of the formula

$$\mathbf{H}_{i+1} = \mathbf{V}_i^T \mathbf{H}_i \mathbf{V}_i + \rho_i \mathbf{s}_i \mathbf{s}_i^T \tag{2.47}$$

where

$$\rho_i = \frac{1}{\mathbf{y}_i^T \mathbf{s}_i} \tag{2.48}$$

$$\mathbf{V}_i = I - \rho_i \mathbf{y}_i \mathbf{s}_i^T \tag{2.49}$$

and

$$\mathbf{s}_i = \mathbf{p}_{i+1} - \mathbf{p}_i \tag{2.50}$$

$$\mathbf{y}_i = \mathbf{g}_{i+1} - \mathbf{g}_i \tag{2.51}$$

Using this method, instead of storing the matrices $\mathbf{H}_i$, one stores a certain number of pairs, say m, of pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$ that define them implicitly. The product of $\mathbf{H}_i\mathbf{g}_i$ is obtained by performing a sequence of inner products involving $\mathbf{g}_i$ and the m most recent vector pairs $\{\mathbf{s}_i, \mathbf{y}_i\}$ to define the iteration matrix.

Like in conjugate gradient methods, the line minimization is needed for determining λ in equation (2.46).

2.4.5 The simplex method

The simplex method as suggested by Nelder and Mead [1965] is a systematic procedure for generating and testing candidate vertex solutions to a minimization problem. It begins at an arbitrary corner of the solution set. At each iteration, the simplex method selects the variable that will produce the largest change towards the minimum solution. That variable replaces one of its compatriots that is most severely restricting it, thus moving the simplex to a different corner of the solution set and closer to the final solution.

A simplex is the geometrical figure consisting, in n dimensions, of $n + 1$ points and all their interconnecting line segments, polygonal faces, etc. In two dimensions, a simplex is a triangle. In three dimensions it is a tetrahedron. The simplex method must be started with $n + 1$ points of initial guess, defining the initial simplex. The simplex method now takes a series of steps. The first step is to move the vertex where the cost is largest through the opposite face of simplex to a lower point. This step is called the reflect step. When the cost of the new vertex is even smaller than all the remaining, the method expands the simplex even further, called the expand step. If none of these steps produce a better vertex, the method will contract the simplex in the same direction of the previous step and take a new point. This step is called the contract step. When the cost of the new point is still not better than the previous one, the method will take the last step called shrink. In this step all of the simplex points, except the one with the lowest cost, are 'shrinked' toward the best vertex.

As it basically only tries and compares the solution of several different sets of parameters, the method requires only function evaluations and no derivatives.

2.5 A more abstract approach

A number of data assimilation and model calibration methods have been described in the previous sections. This is only a brief overview of some of the available methods. There are numerous alternatives, variations and combinations of methods possible.

The formulation of various methods can be completely different like the variational methods and the Kalman Filters. But from a more abstract level there is a lot that all these methods have in common. These are the basic building blocks that are used to formulate the algorithms. The algorithms perform, mostly linear algebra, operations on model state-vectors $\mathbf{x}$ or model parameters $\mathbf{p}$. For all algorithms that are described it is necessary to be able to perform model timesteps, where the state-vector is propagated forward in time. All methods improve the dynamic model by using observations of the system. In order to do so, it is necessary to use an observation operator $H(\mathbf{x})$, that transforms the model state to the observed values.

Some other building blocks are not needed by all methods like an adjoint of the model or an scaling vector (see Section 4.5.1) for comparing various quantities in the state-vector. These less common building blocks are not necessary for all methods but still for a sub-set of methods. Like the adjoint for 4D-var and gradient based calibration methods and the scaling vector for the RRSQRT, COFFEE and for model reduction [Altaf et al., 2008].

Since all methods are constructed from the same basic building blocks it is conceptually possible to use the same approach for the software that implements these methods and develop a framework for data assimilation and model calibration as will be described in more detail in the next chapter.

Chapter 3

COSTA

3.1 Introduction

The COSTA framework for data assimilation and model calibration is the main result of the research presented in this thesis. The framework is discussed in detail in the three papers that are presented in the Chapters 4, 5 and 6. This chapter will present aspects of COSTA that are not, or only briefly mentioned in the three papers.

An analysis of requirements is important before any development can take place. Requirements form the basis of the whole design and the choices that are made in the development process. The requirements that are defined for COSTA in cooperation with users are presented in Section 3.2.

The fundamental setup of the COSTA framework and how this setup relates to the requirements is presented in Section 3.3. The technical aspects on the object oriented programming principles in the COSTA framework that are not discussed in the papers are presented in this section as well.

Besides COSTA there are other software packages and frameworks available for data assimilation as well. The main aspects of these packages, how they are related to COSTA and the requirements are discussed in Section 3.4.

3.2 Requirements of a generic framework for data assimilation

An important step in the development of a software system is to formulate and analyse the requirements. The set of requirements that describes what a system must be able to do are referred to as functional requirements.

Requirements that describe how a system must perform its functions are referred as the non-functional requirements. The non-functional requirements include among others the performance and the ease of use of the system.

The functional and non-functional requirements are very important because they form the basis of the system and most design decisions will be made such that they match the requirements. However this does not mean that the initial written requirements are not to be changed. It is possible that requirements are added, changed or removed but this should be based on new insight in what requirements fit best to the system. Not because some requirements are not met.

In order to get a set of requirements for COSTA that describes a useful system, the COSTA users' group is formed. The COSTA users' group consists of experts in the field of data assimilation and large scale modeling from different institutions in the Netherlands: Delft university of technology, VORtech, Deltares, TNO-MEP, TNO-NITG, and HKV. The set of requirements that are formulated in the following sections are based on the input from members of the COSTA users' group.

COSTA is an innovative system. In advance it was not known what was feasible and what not. The set of requirements is therefore initially chosen to be not too extensive and not too detailed. The requirements form a list of wishes from the members of the users' group and define the properties of an ideal data assimilation and model calibration framework. The requirements form the context in which the system is developed and they have proven to be useful for making design choices while developing the system.

To better understand the background of the requirements it is important to realize that there are various categories of users for which COSTA is a useful tool. Depending on the kind of user, the user will have different requirements. The users are divided into three categories:

1. **Developers of data-assimilation methods**, these developers are scientists with a thorough knowledge of data assimilation methods but with varying programming skills;

2. **Developers of computational models**, these are developers with various skills both on model knowledge and programming skills.

3. **End-users of computational models with data-assimilation capability**, these users have a fair understanding of both the dynamic model as data assimilation methods but are in general not programmers.

The first two kinds of users will be involved in programming using the COSTA framework. The end-users however will only use the system and do not need to be able to program at all.

3.2.1 Functional requirements

In this section we give an overview of the functional requirements of the COSTA framework. These requirements are formulated using the input of the COSTA users' group.

The term COSTA model will be frequently used in the formulation of the requirements. A COSTA model is a dynamic model that is adapted, wrapped or prepared in order to work in the COSTA framework. Similar, a data assimilation or model calibration method that is implemented in COSTA is called a COSTA method.

The following functional requirements are formulated by the COSTA users' group:

- Both data assimilation and model calibration system can be implemented with COSTA.
- COSTA models can be combined with arbitrary COSTA methods and vice versa.
- Parallel computing can be used in COSTA in order to improve the performance of the data assimilation and model calibration algorithms.
- Besides using parallel computing in the algorithms it must be possible to use models in COSTA that by them self already use parallel computing. Using these parallel models in COSTA should not be fundamentaly different to using sequential models.
- Applying a method in COSTA should provides sufficient feedback on the progress and effect of the applied method to the end user.
- COSTA must be a suitable framework both for large and complex (operational) models as well for doing research on small (academic) models.

3.2.2 Non-functional requirements

The set of non-functional requirements that are defined by the COSTA users' group are:

- A set of basic building blocks for data assimilation and model calibration must be defined for COSTA. Besides the definition it must provide generic implementations of these building blocks in order to enable the rapid development of COSTA methods.
- COSTA allows the reuse of software especially it must be possible to:
  - share COSTA models and COSTA methods among COSTA users,
  - replace the generic implementations of the COSTA building blocks by alternative implementations and share these alternative implementations amoung users and applications.
- The COSTA framework and the software developed using COSTA must be portable. Therefore it must be:
  - platform independent,
  - and possible to use COSTA in combination with (model) software that is developed in various programming languages, especially: Fortran77/90/95 and C.
- Software that is developed using COSTA must be computational efficient. The performance of COSTA should be of the same order as dedicated implementations of data assimilation and model calibration methods.
- The COSTA framework must be easy to use, both for end users and developers.
  - It must enable end-users of models with data-assimilation capability to easily experiment with data-assimilation algorithms and the settings of parameters therein
  - It does not require end-users to have any programming skills in order to use COSTA.
  - It does not require the end-user to do any programming in order to combine COSTA models with COSTA methods.
  - A method in COSTA is not more difficult to program than the same method independent from COSTA in a generic programming language like Fortran, C or Matlab.
  - The transformation of a dynamic model into a COSTA model should not be more difficult or time consuming than preparing the same dynamic model for use with a dedicated data assimilation or calibration method.

- COSTA must be an open source platform and aims at beeing a standard. The license should allow to use COSTA in combination with dynamic models and other existing software that is propriatory as well as open source. Such that it is easy accessible for a large group of potential users.

3.3 Overview of the system

An overview of the core design of the COSTA framework is presented in this section. This section does not give a detailed description of the available classes and components. The most relevant classes and components are discussed in the Chapters 4, 5 and 6.

The most fundamental choice in COSTA is the use of concepts from object oriented programming. As a result of this design choice it was possible to fulfill a number or requirements. The motivation for the object oriented approach is given in Section 3.3.1. The COSTA framework consists of various building blocks of various size and complexity. In Section 3.3.2 it is explained how these building blocks relate to each other. COSTA uses concepts from object oriented programming but can be used in imperative languages as well. The technical aspects of using object oriented concepts in imperative programming languages are explained in Section 3.3.3.

3.3.1 An object oriented approach

One of the requirements states that it must be possible in COSTA to combine dynamic models with the available data assimilation and calibration methods. There is another, rather similar requirement. This requirements states that COSTA should define basic building blocks for data assimilation and model calibration that enable the rapid development of COSTA methods.

There are roughly two approaches that enable easy replacement of parts of a software system in software devellopement. The first approach focuses on the way data is stored. In this approach a strict definition is made for the representation of data. When all data storage is standardized it is possible to use and share the data among parts of the software system. Standardizing of the data makes it possible to provide generic routines as building blocks for the data assimilation and calibration framework. In order to be able to combine arbitrary models with data assimilation and model calibration methods it is not sufficient to only standardize the data representation. It

is also necessary to define a set of functions that can be used in the code of the data assimilation method in order to interface with the model.

This approach is however not very suited for systems that are programmed in various programming languages. Basic data representation like scalars or arrays of integers, real numbers and strings can be shared among most programming languages without too much problems. Complex forms of data representation like the Fortran90 types that groups various kinds of data into a composite data representation are very language specific and it is far from trivial to share them among code that is written in different languages. This can be avoided by using only simple forms of data representation in the interface. The disadvantage is that most data cannot be represented by a single variable. As a result the functions will have a large number of arguments.

To overcome the problem of sharing complex data structures, COSTA uses concepts from object oriented programming. The building blocks in COSTA are similar to classes in object oriented languages but they can also be used in non-object oriented languages like Fortran and C. An important aspect of the object oriented approach is that the data, called the state of an instance of a class, cannot be accessed directly. The actual representation is completely hidden from the code that uses the instance of the class. The state of an instance can only be changed by using a predefined set of functions. Similarly it is only possible to get data from the state by a set of functions. These functions together form the interface of the class. The object oriented approach has a disadvantage as well. It is not possible in the object oriented approach to directly access the data of a class instance. Copying of data is needed when a class does not provide the necessary methods in the interface. This is undesirable since it will have a negative impact on the performance and memory requirements. Therefore it is important to carefully design the interface of the classes such that the overhead for the object oriented approach is reduced to marginal proportions.

The arguments of the functions of the interface only consist of instances of COSTA classes and basic data representations like scalars or arrays of integers, reals and strings. In this way it is possible to implement the COSTA classes in various programming languages without complications. The implementation of the class is completely hidden and independent from the context in which the class is used.

Most classes in COSTA are quite simple in the sense that they contain a limited amount of functionality. These simple classes are in general data containers, replacements for complex data structures. Examples of these kinds of classes are the time-, vector-, matrix- and file-class. There are

three classes in COSTA that provide a large amount of functionality. These classes are called components: the model, the stochastic observer and data assimilation or model calibration method. Typically together they form a data assimilation or model calibration system.

The object oriented approach isolates all model specific information from the outside and offers an uniform way for using models in a data assimilation or model calibration method. The object oriented approach has other applications as well. By only using the methods from the COSTA model interface it is possible to create larger composite models using a generic tool called the model combiner. In a similar way it is possible to automatically parallelize independent propagations of model state-vectors and the use parallel models in COSTA. These additional applications are discussed in detail in Chapter 6. The creation of a COSTA model component for an existing model is described in detail in Chapter 4.

The observations are represented in the form of a stochastic observer class. The stochastic observer contains the measured values and all necessary information related to the observation including the location, measurement error statistics and observation kernel of satellite measurements. The stochastic observer is discussed in more detail in Section 4.3.5.

3.3.2 Set-up of the system

The COSTA framework is built up in different layers as illustrated in Figure 3.1. The bottom layer is not specific for data assimilation but provides the basic tools for programming the COSTA classes. This layer contains a number of useful software libraries that are written by third parties and the basic classes like vectors, matrices, time, functions and handles. The function and handle classes play an important role in the object orientation in COSTA and will be discussed in more detail in Section 3.3.3. This layer contains also robust and efficient numerical methods, input/output facilities and data storage. On top of this first layer, the data assimilation and calibration specific classes and components are constructed like the tree-vector, model, observations, model combiner and black box model builder. The top layer contains complete (utility) programs including the COSTA workbench program that can be used by the end user to run any combination of model, observations and data assimilation or calibration method in parallel or sequentially. The use of this program is explained in more detail in Chapter 6.

Generic implementations are available for all COSTA classes and components. This makes it possible to quickly compose a data assimilation system.

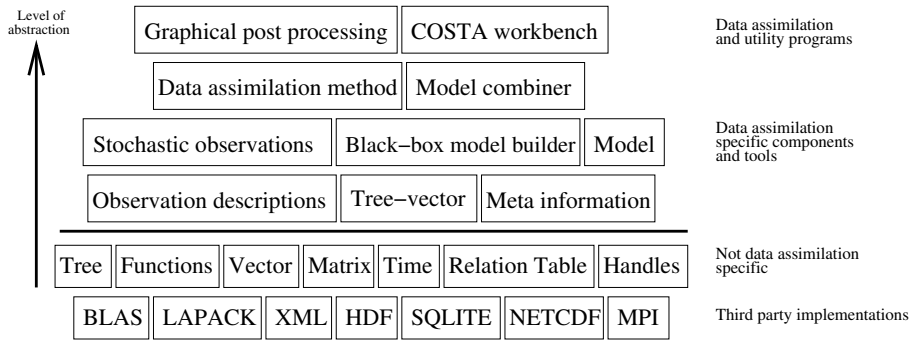

Figure 3.1: An overview of the layers in COSTA. The bottom layer contains non data assimilation and calibration specific software. The middle layer contains data assimilation components and the top layer contains utility programs

The developers of a dynamic model initially only need to create a COSTA model component and can build the whole data assimilation system using the already existing implementations of the classes. Developers of data assimilation methods only need to implement the method and various models are already available to do experiments with.

In a later stage, when necessary it is possible to use alternative implementations of COSTA classes in order to improve the performance of the system or add additional functionality like e.g. output to specific data formats.

3.3.3 Object oriented programming in imperative languages

The setup of COSTA is object oriented. However most existing model implementations are in imperative languages like Fortran and C. Therefore COSTA provides full functionality for these languages. Two COSTA classes, the function and handle provide the basis for the object oriented concepts that are available in the COSTA framework.

The COSTA handle is the base of all COSTA classes, it can be compared to the base class in object oriented languages. A COSTA handle associates an ID (unique integer value) to an instance of a COSTA class e.g. a vector or model instance. A handle has two attributes the type of the class and a block of memory. The size and content of the block of memory are dependent on the type of the class.

Figure 3.2: The standard data structure composition of a COSTA class instance. An instance of the vector class is graphically presented in this example. The generic handle holds the type of the class it represents and all related data. The data is divided in two parts; the implementation specific data that holds the content of the vector and a handle to the corresponding classimplementation. The classimplementation holds all function instances of the functions that implement the interface of the vector class.

The second basic COSTA class is the function. The function class holds a pointer to a function written in e.g. C or Fortran. Typically to a function that implements one of the methods of a COSTA class. When a function instance is created, it is possible to directly associate it to a function or to load the function from a dynamic library.

Most COSTA classes can be replaced by alternative implementations except for the handle and function classes because they form the core of the whole COSTA framework. The classes, for which it is possible to provide alternative implementations, are associated to a third basic class in COSTA called classimplementation. The classimplementation holds a list of the functions that together implement the interface of the COSTA class.

Instances of COSTA classes are constructed from these three basic classes. This is illustrated in Figure 3.2 for an instance of a COSTA vector.

The COSTA framework implements an interface layer for all COSTA classes. All methods of any class instance that are executed in the COSTA framework are direct calls to functions in the interface layer with the instance handle as one of the arguments. The interface layer performs some consistency checks and will then get the corresponding function pointer from the classimplementation instance. This function will than be called with the implementation specific data as one of the arguments. The computational overhead of the interface layer is very limited compared to the total amount of computations that are generally performed by a method. The implementation in pseudo code of the interface layer implementation of the norm2-method of the vector class is given in Figure 3.3.

```
function cta_vector_norm2(vec,norm){
  checkHandle(vec,'isVector');
  func=GetFunctionPointer(vec,'norm2');
  data=GetDataPointer(vec);
  func(data,norm);
}
```

Figure 3.3: Implementation of the interface layer of the norm2 method of the vector class. The handle of the vector instance is first checked then the implementation specific data is retrieved together with the function the implements the method. Finally this function is called with the implementation specific data block

The interface is more sophisticated for some classes than the example in Figure 3.3. Some methods in the interface are redundant, which means

that they can be derived from other methods in the interface. For example in case of the vector class, every method can be implemented using the getValues and setValues methods although not computationally efficient. The interface layer can implement some redundant methods when they are not provided in the class implementation. This enables rapid development of class implementations because it is not necessary to implement all methods. However for an optimal performance it is eventually necessary to implement them all.

A Java interface is available for COSTA as well. This interface called OpenDA is developed in collaboration with the developers at Deltares of DATools, see Section 3.4. Using this interface it is possible to develop data assimilation and model calibration methods in Java. The interfacing with Java combines the object oriented syntax and utilities of Java with the computational efficiency of C and Fortran. The Java interfacing is similar to the COSTA interface layer.

3.4 Alternative frameworks

COSTA is not the only available software that is useful for creating data assimilation applications. Implementations of data assimilation methods and model calibration methods are available in the form of software libraries and MATLAB-toolboxes. In this section an overview is given of some other frameworks, packages and software that are like COSTA useful for implementing data assimilation techniques for dynamic models.

ReBEL
A MATLAB toolbox called ReBEL (Recursive Bayesian Estimation Library) [der Merwe and Wan, 2006] is developed at the Oregon Health & Science University. The toolbox implements various variations of Kalman and particle filters. The packages focuses on relatively small models of which full error covariances can be handled in memory. In order to be able to use a model with Rebel it is necessary to implement a set of MATLAB functions. The ReBEL toolbox is not open source but is available free of change for educational purposes.

DAIHM Toolbox
The DAIHM toolbox [Drécourt et al., 2003, Drécourt, 2001, 2004] is a MATLAB toolbox developed at DHIWater & Environment. The package only implements an Ensemble Kalman filter. An important aspect of the toolbox

is the representation of the state-vector. The state-vector is not represented as a vector but it includes additional information. This information is used to make a mapping between the spatial representation of the values and the vector representation that is used in the Ensemble Kalman Filter. Various tools are available to visualize the results. The models are not programmed in MATLAB. The original model executables are started from the MATLAB code. The DAIHM Toolbox is a propriatory piece of software of DHI.

NERSC ensemble Kalman Filter
The NERSC ensemble Kalman filter by [Evensen, 2007] is suited for very large dynamic models. The interface with the model is realized by reading the initial state of the ensemble from file and after the assimilation step writing the ensemble back to file. A parallel version is available to reduce the memory requirements and computational time. The source is freely available.

DART
The Data Assimilation Research testbed (DART) [Anderson et al., 2009] is a framework for data assimilation that is developed at the National Center for Atmospheric Research (NCAR). DART contains various data assimilation methods. Models can be used in DART by writing a wrapper around the executable of the model or wrap the model directly by implementing a number of routines. DART contains an advanced ensemble Kalman filter implementation including inflation and smoothing. Some of the computations can be performed in parallel and the system is successfully linked to some large operational model including CAM, WRF, AM2. DART is not open source but it is available for research purposes.

DATools
The DATools toolbox [Serafy et al., 2007] is developed at Deltares (formerly Delft Hydraulics) to offer a range of data assimilation methods and tools for the dynamic models that are originaly developed and used at Delft Hydraulics. An important aspect of DATools is the interfacing with the end-user. The toolbox contains various tools that simplify the specification of uncertainty in the dynamic models. DATools has been successfully used in combination with various models at Deltares.

Palm
The PALM project of Cerfacs [Morel et al., 2008, Lagarde, 2000] offers a tool to compose a data assimilation application from various units that have been

created by the programmer. The units are coupled into the overall application be linking them using the PALM graphical user interface. The PALM framework will be in charge in for managing the sequences of the units and the messages that are send between them. The PALM framework is used for various operational system including the French Operational oceanography project MERCATOR. PALM is propriatory software of Cerfacs but it is available for other users as well.

The following frameworks are not specially designed for data assimilation but in principle that can be used for implementing (parts) of a data assimilation framework.

The Dutch OMS
The OMS project was initiated to migrate the two Dutch modeling systems for Delft3D of Delft Hydraulics and SIMONA of Rijkswaterstaat into a single system called the Dutch Open Modeling System (OMS). One of the products was the development of the OMS-backbone a system for combining and (parallel) running of various software components. The system has been successfully used to link models in to a larger composite model like WAQUA-SWAN and Delft3D-FLOW with morphology functionality.

HarmonIT
The HarmonIT project [HarmonIT, 2009] is a European a research project. The objective is to develop and implement the Open Modeling Interface and Environment (OpenMI). The OpenMI interface should simplify the linking of various models. OpenMI is already implemented in various operational models of various institutes including amonng others DHI, Deltares and Wallingford. OpenMI makes it possible to combine different models of various institutions in a single forecasting system.

The DATools toolbox and DART toolbox are the systems that have the most similarities with COSTA. All three systems are intended to be used with various models. Another similarity between these projects is that they were all approximately initiated at the same time. Members of the development team of DATools are part of the COSTA users group. This opportunity has been picked up to start working together in an early stage. The result of this cooperation is an almost similar design of the data assimilation building blocks. The first steps are already taken to combine DATools with COSTA into single framework for data assimilation that will be called OpenDA.

There are three fundamental differences between DART and COSTA.

DART only implements an ensemble Kalman Filter and the coupling between DART and the models is develloped for usage with this method. COSTA and the model interface in COSTA are develloped such that they can be used for a range of data assimilation methods and model calibration methods. This does imply that DART cannot be used in the future in combination with other methods but then it is necessary to extend the DART coupling with models considerably. The DART developers do not use an object oriented approach and use Fortran variables and structs for storing data and sharing data. Finally DART is not freely available what makes it currently not possible to use it in combination with propriatory models or for commercial use.

PALM, OMS and HarmonIT are mainly concerned in linking software components. These frameworks are not limited to applications on the field of data assimilation. These tools provide a mean for software components to communicate with each other but they do not define what the components must be able to do such that they can be used and reused in a data assimilation framework. These frameworks are in some sense complementary to COSTA because the focus of COSTA is to identify the generic components in data assimilation and design their interface such that they can be reused. COSTA supports various kinds of parallel computing but this is completely hidden from the component interface. Frameworks like PALM, OMS and HarmonIT can be used in combination with COSTA e.g. a COSTA model interface can be created on top of the OpenMI interface of a model.

Chapter 4

Application for Hydrodynamic Modelling [1]

4.1 Abstract

A problem solving environment for data assimilation called COSTA is developed at Delft University of Technology. The goal of COSTA is to offer a modular framework where simulation models can be combined with various data assimilation methods. COSTA defines a number of building blocks called components. Examples of components are model, method, stochastic observer and tree-vector. New data assimilation systems can be created by combining these components. This paper describes the application of COSTA to the WAQUA/TRIWAQ shallow water simulation model. In the past a model specific RRSQRT Kalman filter has been implemented for WAQUA/TRIWAQ. However, this implementation cannot be used in combination with other models. The WAQUA/TRIWAQ model is changed into a COSTA model component and the original RRSQRT Kalman filter is changed into a generic filter that can be used for other models as well. The new filter is now a part of the COSTA environment. The original filter contained a number of WAQUA/TRIWAQ specific aspects e.g. the drying and flooding of areas in the model. These model specific issues are identified, isolated and moved into the model component. A COSTA based

[1] The content of this chapter is publised in Meteorologische Zeitschrift [van Velzen and Verlaan, 2007]; COSTA a problem solving environment for data assimilation applied for hydrodynamical modelling, by Nils van Velzen and Martin Verlaan, published in Meteorologische Zeitschrift, 16:777-793, 2007. The content is slightly updated for changed terminology.

implementation of WAQUA/TRIWAQ with the RRSQRT Kalman filter is realized and compared to the original system in a number of experiments. The experiments show that the COSTA based system produces the correct results and the computational overhead for using COSTA is low. The new RRSQRT Kalman filter is also combined with other COSTA models including the LOTOS-EUROS model for atmospherical transport, chemistry (chemical reaction) and deposition of air pollution on the scale of Europe.

4.2 Introduction

COSTA (COmmon Set of Tools for the Assimilation of data) is a problem solving environment for data assimilation and model calibration that is developed at Delft University of Technology. COSTA offers a modular framework, containing data assimilation methods, calibration methods and tools that can be combined with various simulation models [van Velzen, 2006].

COSTA is an open source project released under the LGPL license and is available at sourceforge.net, project costapse. Under the LGPL license it is possible to freely use COSTA in combination with open source models as for propriatory models.

The goal of COSTA is the simplification of the application of data assimilation methods. In this way we want to make data assimilation available to a larger group of researchers and application areas. COSTA defines a set of building blocks for data assimilation that can be interchanged. This opens the way to the reuse data assimilation and calibration methods. Reversely, new data assimilation and calibration methods developed in COSTA can be tested on the available models that are already linked to COSTA.

This article describes the application of COSTA to models developed at the Dutch National Institute for Coastal and Marine Management (Rijkswaterstaat/RIKZ). Important tasks of Rijkswaterstaat are: the protection of the country from flooding, the management of shipping routes and ports and water management in terms of quantity and quality.

Furthermore Rijkswaterstaat is responsible for the construction, management and maintenance of public works, and for research and regulations related to the public works.

The models at Rijkswaterstaat are used for operational purposes, for supporting policy making, and for research, to gain better understanding of the physical processes. The most important model is WAQUA/TRIWAQ, which implements 2D and 3D simulation models for shallow waters: rivers,

estuaries and coastal areas [Stelling, 1983].

Various data assimilation methods have been implemented for WAQUA/TRIWAQ over the years. The available methods are the Chandrasekhar steady state- [Heemink and Kloosterhuis, 1990], RRSQRT- [Verlaan and Heemink, 1997a] and ensemble Kalman [Evensen, 2003] filters. The adjoint of the 2D model, called WAQAD is also developed and is used for model calibration [Mouthaan et al., 1994], [Verlaan et al., 1996]. There is however a need to apply the same assimilation methods to other simulation models at Rijkswaterstaat and to extend the number of assimilation methods, including e.g. POEnK and COFFEE [Heemink et al., 2001] in the existing software. The addition of new data assimilation techniques to WAQUA/TRIWAQ is expensive and migration of the existing assimilation techniques to other simulation software is far from trivial.

The COSTA framework offers tools for tackling these kind of problems. Therefore Rijkswaterstaat is implementing the COSTA framework in their simulation systems as it is being developed. The advantage for Rijkswaterstaat is that they can benefit from the advantages of COSTA at an early stage. At the same time the development of COSTA's concepts and its software will also benefit from this. The problems and successes when implementing COSTA in a complex operational system will give important information for improving the COSTA system.

This article describes the first steps taken, where COSTA is introduced in WAQUA/TRIWAQ. The existing software is reused and split into two main parts: a COSTA model component of the WAQUA/TRIWAQ model and an RRSQRT Kalman filter implementation that can be used in combination with other COSTA model components and that is equivalent to the RRSQRT Kalman filter that is currently available in the WAQUA/TRIWAQ software.

WAQUA/TRIWAQ is an operational system. Therefore the new version of the software that is developed using COSTA must have the following properties: the computational results must be very close to the original version. An explanation must be found for all changes in the results. The changes made to the software, especially the computational core must be kept to a minimum and, the computational performance must be similar to the original software.

The COSTA environment and the main building blocks are presented in Section 4.3. Here we will also give an overview of the available data assimilation and model calibration methods and describe the RRSQRT Kalman algorithm in detail.

Existing models have to be linked to COSTA before the available meth-

ods in COSTA can be used. In Section 4.4 we will illustrate how an existing model can be linked to COSTA. Here we use the WAQUA/TRIWAQ model as an example.

The operational WAQUA/TRIWAQ system with RRSQRT filter handles a number of issues that are specific for the WAQUA/TRIWAQ model. We will discuss these problems in Section 4.5.

In Section 4.6 we will describe from a software point of view how the existing RRSQRT Kalman filter is transformed into a reusable COSTA data assimilation method and the WAQUA/TRIWAQ model is coupled to COSTA.

The new implementation is tested for results, computational performance and reusability. The performed test and the results are presented in Section 4.7.

We will conclude with a short overview of planned developments on the COSTA environment in Section 4.8 and the conclusions in Section 4.9.

4.3 COSTA

COSTA is a problem solving environment developed at Delft University of Technology. It is intended to facilitate data assimilation and model calibration for a wide range of simulation models. It provides a number of building blocks for data assimilation and calibration systems. Combining and creating new building blocks should be possible with a minimum of effort. These building blocks are called components and are discussed in more detail in Section 4.3.1. There are building blocks for handling the observed data, different assimilation and calibration methods, (stochastic) models and noise models. Besides these major components, COSTA also defines a number of smaller more generic components like tree-vectors, vectors, matrices and time.

COSTA is not the only available software for data assimilation. There are data assimilation methods available in the form of software libraries or MATLAB-toolboxes. Among many others there are the ReBEL developed at OHSU, the DAIHM MATLAB Toolbox [Drécourt, 2004] and the NERSC ensemble Kalman filter EnKF. The ReBEL toolkit contains a number of Kalman filters and particle filters. The DAIHM and NERSC provide an ensemble Kalman filter. These available methods are in general high quality methods. The number of available methods is however limited and all methods use a different interface. This means that a model that is adjusted to work with a particular package cannot be used in an other package.

The models used for data assimilation are in general very complex and

the overall model is often a combination of several sub-models. The PALM project of Cerfacs [Lagarde, 2000] and the ESMF project [Hill et al., 2004] offer tools for handling these problems. Other projects that are very similar but not especially designed for data assimilation are OMS [Hummel et al., 2002] of the Dutch Rijkswaterstaat and WL|Delft Hydraulics and the HarmonIT project [Gijsbers, 2004]. These projects offer tools for combining different components like model and assimilation method in a (parallel) environment.

The focus of COSTA is to identify the generic components in data assimilation and design their interface. This approach makes it possible to interchange components. The other projects that are mentioned here like PALM, ESMF, OMS and HarmonIT mainly provide tools for the combination of components in a parallel environment. These projects do not define a set of components with their interface. At that point COSTA is supplementary to PALM, ESMF, OMS and HarmonIT.

4.3.1 COSTA components

COSTA provides building blocks in the form of COSTA components. COSTA components are very similar to classes in C++. Instances of a C++ class or a COSTA component are called objects.

Objects have a state, which can be seen as its value(s). For every component, COSTA defines an interface, which is a set of methods. A method is something that can be done with an object: a function that returns information on the component or changes its state.

COSTA defines various components on different levels. These components are the building blocks of the data assimilation system. An overview of the most important components is given in Table 4.1.

The components are split up into two groups. The first group are the high level components: the model, observations and data assimilation method. The combination of these three components form a data assimilation system.

The second group of components contains a number of smaller useful components like vector, matrix, time, tree-vector and pack object. These components are used for building the main components and to pass information between components.

Some of the components in Table 4.1 are marked with an asterisk. This indicates that it is possible to add alternative versions of these components to the COSTA environment. COSTA provides one or more versions of all the components that are listed. For example there are a number of academic models available in COSTA that are useful for testing assimilation methods.

Component Name	Short Description
Model(*)	Simulation model.
Stochastic observer(*)	A set of available observations, including the stochastic roperties and observation description.
Data assimilation method(*)	Data assimilation methods and model calibration methods.
Observation description(*)	The properties of a set of observations. The description describes the physical interpretation of the observations.
Tree-vector	Extension of a vector. The tree-vector can be build from other tree-vectors and optionally contains meta information for interpretation of the values in the vector.
Meta information(*)	Optional information associated to a tree-vector. It describes the physical meaning of the values in a tree-vector This information is used for carrying out interpolations between different tree-vectors.
Vector(*)	Basic vector for carrying out basic linear algebra operations.
Matrix(*)	Basic matrix for carrying out basic linear algebra operations.
Tree	Dynamic data structure in the form of a tree where all leafs contain an instance of a arbitrary COSTA component. The tree is used for grouping data in a structured way.
Pack	Component in which we can store information in a continuous block in memory. Most COSTA components can be packed. Their internal state is then added to the content of a pack object.
Time	Component for describing time in an uniform way.
Model Combiner	Create a COSTA model that is the combination of two or more existing COSTA models.
Model Builder SP	Single Processor modelbuilder. This model builder simplifies the development of new COSTA models from new and existing software for models with some special properties. The model builder handles most of the code needed to support the object oriented property of COSTA models where multiple instances of a model can be created.
Model Builder PAR	PARallel model builder. This modelbuilder extends a COSTA model for parallel computing. The computation of timesteps various model instances are of the performed in parallel when this model builder is used.

Table 4.1: Overview of the major components in the COSTA environment including a short description. Components that are marked with an asterisk (*) can be extended with alternative versions. For these components it is possible to code and use an alternative version of this component instead or as addition to the existing versions of the component.

However it is possible to extend the set of available models.

The available components are a starting point for relatively quickly building a data assimilation system. It is possible to provide an alternative version of some of the components e.g. for improving performance or functionality. These alternative versions of components can be used in other data assimilation systems as well because the interface is not changed.

The model, tree-vector, stochastic observer and available data assimilation and calibration methods will be described in detail in the next sections. After that we will give a detailed description of the RRSQRT Kalman filter that is now part of the COSTA environment.

4.3.2 Mathematical description of a COSTA-model

COSTA models are COSTA components and therefore have a state (value) and an interface. COSTA models are intended to describe stochastic models, which means that a model is available for the uncertainties (differences between model results and reality). Deterministic models are seen as a special case of a stochastic model, in which the uncertainties are ignored (assumed zero).

The state of a COSTA-model is denoted as $\mathbf{x}(t)$ and the 'formal' notation for a model becomes,

$$\frac{d\mathbf{x}(t)}{dt} = \mathcal{M}\left(t, \mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \mathbf{w}(t), t\right) \tag{4.1}$$

where $\mathcal{M}$ denotes the model operator, which is in practice often an existing piece of software that solves the model-equations numerically. In most cases an existing model will not describe uncertainties and thus a stochastic extension is necessary. The inputs to the model are the initial state $\mathbf{x}(0)$, time-independent parameters $\mathbf{p}$, time dependent forcing $\mathbf{u}(t)$, and a stochastic forcing $\mathbf{w}(t)$.

Besides a model, observations are sometimes available as well. The observations that available at time t are denoted by $\mathbf{y}(t)$. The corresponding observation operator is:

$$\mathbf{y}(t) = H\left(\mathbf{x}(t)\right) + \mathbf{v}(t) \tag{4.2}$$

the operation H interpolates and transforms the state variables to the observations. The observation error $\mathbf{v}(t)$ includes both the instrumental and representation error and is supposed to have zero mean and covariance $\mathbf{R}(t)$.

4.3.3 Interface functions of the COSTA model component

The previous section has discussed the structure of COSTA models. The interface of the COSTA model component that contains all the functions necessary to support data assimilation methods is discussed in this section.

The 'value' of a COSTA-model consists of $\{\mathbf{x}, \mathbf{u}, \mathbf{p}\}$. The model interface contains methods for changing the value of the model or getting information from the model's value. Table 4.2 gives an overview of all the methods that form the interface of the model component.

An important property of the `SetState`, `GetState` and `AxpyState` operations is that they can be directly performed between different model instances. This will avoid unnecessary copying of data.

The `AxpyState` operation between two model instances can be used to nicely handle some model specific problems inside the model component and not in the filter implementation.

The `AxpyState` can be used by assimilation methods to update the state of the model or to compute the difference between model states. Addition and subtraction of model states does not always yield results that can be used without problems in the following steps of the data assimilation methods. An updated state can have no physical meaningful value e.g. negative concentrations or unstable stratification. The computed difference between states of two (nonlinear) models sometimes needs some special treatment in order to avoid unstable data assimilation methods. These model specific issues can be handled inside the `AxpyState` method of the model when necessary.

The list of methods in the COSTA model interface will grow in time when more assimilation methods will be added to COSTA. However this does not mean that the creation of a COSTA model component for a model will become more and more time consuming.

Some of the methods can be derived from other methods. The various `Axpy` methods can be realized using the corresponding `Get` and `Set` methods. These methods will automatically be realized by COSTA when they are not provided.

There are however methods that cannot be realized automatically when they are not provided. In that case it will not be possible to use the model in combination with all the available data assimilation and calibration methods. For example when the model does not implement the methods for changing the model parameters it cannot be used in combination with the available model calibration methods.

When a model is linked to COSTA it is at first only necessary to im-

`Create`	Create a new instance of a model having its own internal state. Depending on the model all kinds of initialization will be performed when this method is called.
`Free`	Free a model instance, deleting its model state and performing all tasks to stop the model instance in a regular way and clear the used resources.
`Compute`	Carry out the time steps necessary to step through a given time span (Equation 4.1).
`GetObsValues`	Interpolate the model prediction that corresponds to a given set of observations. The information on the observations are provided in the form of an observation description component.
`GetObsSelect`	Given a set of observations, described by an observation description component, return the selection of observations that can be provided by the `GetObsValues` method of the model. This method is used to filter out observations of quantities that are not described by the model or are measured outside the modelled area.
`AddNoise`	The addition of random noise $\mathbf{w}$ in the `Compute` can be switched on and off using this method.
`GetNoiseCount`	Return the number of noise parameters of the stochastic model.
`GetNoiseCovar`	Return the (root of) noise the covariance matrix $\mathbf{G}$.
`SetState`	Set the state $\mathbf{x}$ of the model instance.
`GetState`	Get a copy of the model state $\mathbf{x}$.
`AxpyState`	Modify the model state by adding α times a tree-vector to the state of the model. This operation can be performed between a model and a tree-vector as well between two model instances.
`SetParam`	Set the model parameters $\mathbf{p}$.
`GetParam`	Get a copy of the model parameters $\mathbf{p}$.
`AxpyParam`	Add a offset to the model parameters $\mathbf{p}$.
`SetForc`	Set the forcings of the model to a given value for a specified timespan.
`GetForc`	Get the forcings of the model for a given time
`AxpyForc`	Add a constant offset to the model forcings for a given timespan.

Table 4.2: Overview of the methods of the interface of the COSTA model component.

plement the methods that are needed for the type of data assimilation or model calibration method that will be applied first for the model. The missing methods can be added in a later stage when usage in combination with other methods is needed.

4.3.4 Tree-vector

The tree-vector is an extension of a vector component. A tree-vector either contains a single vector or is a concatenation of a number of tree-vectors, called sub-tree-vectors in this context. A COSTA tree-vector represents a tree structure where the leafs are vectors. The usage of sub-tree-vectors makes it possible to concatenate models where the state of the two models are the sub-states of the state of the concatenated model. An example of such a concatenation is the extension of a deterministic model into a stochastic model (see Equation 4.15). The sub-states are also very useful if the state of the model is not represented as a single array inside the model source code but is distributed over a number of arrays. Each array can be associated with a single sub-tree-vector combined into the tree-vector of the whole model.

Every (sub-) state has a tag. Tree-vectors having the same tag are considered to be the same, meaning they have the same structure of sub-states, length and data-types.

The Tree-vector of a model has a unique structure. Knowledge of the structure is not necessary for performing manipulations on the state. A tree-vector can always be manipulated like a normal vector. COSTA offers a set of methods for manipulating tree-vectors.

4.3.5 Stochastic observer and observation description

The observed values are available in the form of a stochastic observer in COSTA. An instance of a stochastic observer holds a number of observations including their distribution and covariance. The stochastic observer also includes the available additional information on the observed values like location, unit and physical meaning. This additional information is represented in the form of an observation description component.

The information in the observation description component can be represented by a table where each column has a unique name called key. A row in this table contains all information on a single observed value. The keys can be freely selected and depend on the type of observation. The only mandatory key is time.

It is possible to create a new instance of a stochastic observer that is a subset of an existing stochastic observer. The selection can be based on the time associated with the observations or more generally based on a selection criterion formulated in a subset of SQL.

The information in the observation description component is conceptually represented as a table. It does not imply that it needs to be stored in memory in the form of a table. COSTA includes a default implementation of the stochastic observer and observation description components. The observations and their description are stored in a database containing two tables. One table contains time independent information that can be shared by various observations. The other table contains all time dependent information for each observation. The data is externally represented as being one large table but it is stored in a space efficient way.

In order to be able to use the default implementation of the stochastic observer it is necessary to convert existing sources of observations into a COSTA observation database. An alternative approach is to implement the interface of the stochastic observer and observation description component for an existing storage format or database.

4.3.6 Data assimilation and calibration methods in COSTA

The data assimilation and model calibration methods form the core of COSTA. The methods are developed such that they do not contain model specific information. All interaction between the method, the model and the observations takes place by only using the defined methods in the interface of the components. A model component that implements the COSTA model interface can be combined with all the available methods in COSTA.

However this does not guarantee that the combination between a given model and a data assimilation method will always be successful. The success of the application of a specific method depends on the model properties e.g. the nonlinearity of the model. In some cases it is also necessary to handle some computations within the method in a model specific manner in order to avoid problems like instability of the method.

This does however not mean that we cannot use a model independent method as offered by COSTA. We will illustrate in Section 4.5.2 that a necessary model specific computation can be handled within the appropriate method of the COSTA model interface in specific cases.

Currently COSTA contains the following sequential data assimilation methods: Ensemble Kalman Filter (EnKF) [Evensen, 2003], Ensemble Square Root Filter (EnSRF) [Whitaker and Hamill, 2002], Complementary Orthog-

Data assimilation methods	
EnKF	Ensemble Kalman Filter.
EnSRF	Ensemble Square Root Filter.
COFFEE	Complementary Orthogonal subspace Filter for Efficient Ensembles.
RRSQRT	Reduced Rank Square Root filter.
Model calibration methods	
CG	Conjugate Gradient method.
LBFGS	Quasi Newton methods using the LBFGS method for approximating the Hessian matrix.
Simplex	Simplex method.

Table 4.3: Overview of the data assimilation and model calibration methods that are currently available in the COSTA environment.

onal subspace Filter For Efficient Ensembles (COFFEE) [Heemink et al., 2001] and RRSQRT Kalman filter [Verlaan and Heemink, 1997a].

The RRSQRT Kalman filter is added to the COSTA environment as part of the work that is presented in this article. The algorithm is described in Section 4.3.7.

COSTA also offers methods for model calibration where the time independent model parameters $\mathbf{p}$ are calibrated. The available methods are: the Conjugate Gradient (CG) method [Fletcher and Reeves, 1964] , a Quasi Newton methods using the LBFGS method [Nocedal, 1990],[Byrd et al., 1994] and the Simplex method [Nelder and Mead, 1965].

All data assimilation and model calibration methods that are currently available in COSTA are presented in Table 4.3.

4.3.7 The RRSQRT Kalman filter

The RRSQRT Kalman filter is an approximation of the Kalman filter equations:

$$\begin{aligned}
\mathbf{x}^f_{k+1} &= \mathbf{M}_k \mathbf{x}^a_k && (4.3)\\
\mathbf{P}^f_{k+1} &= \mathbf{M}_k \mathbf{P}^a_k \mathbf{M}^T_k + \mathbf{Q}_k && (4.4)\\
\mathbf{x}^a_{k+1} &= \mathbf{x}^f_{k+1} + \mathbf{K}_{k+1}\left(\mathbf{y}_{k+1} - \mathbf{H}_{k+1}\mathbf{x}^f_{k+1}\right) && (4.5)\\
\mathbf{K}_{k+1} &= \mathbf{P}^f_{k+1}\mathbf{H}^T_{k+1}\left(\mathbf{H}_{k+1}\mathbf{P}^f_{k+1}\mathbf{H}^T_{k+1} + \mathbf{R}\right)^{-1} && (4.6)\\
\mathbf{P}^a_{k+1} &= \mathbf{P}^f_k - \mathbf{K}^o_{k+1}\mathbf{H}_{k+1}\mathbf{P}^f_{k+1} && (4.7)
\end{aligned}$$

where $\mathbf{M}$ is the model operator, $\mathbf{x}^f$ the forecasted state vector, $\mathbf{x}^a$ the analyzed state vector, $\mathbf{Q}$ the error covariance matrix of added noise to the model, $\mathbf{y}$ the observed values and $\mathbf{H}$ the interpolation operator between model state and observations.

The model error covariance matrix $\mathbf{P}$ is substituted by the product of the root matrix $\mathbf{L}\mathbf{L}^T$. Equation 4.4 is replaced by

$$\mathbf{L}^w_k = [\mathbf{M}_k \mathbf{L}^a_k, \mathbf{Q}^r_k] \tag{4.8}$$

where $\mathbf{Q}^r_k$ is a root matrix of $\mathbf{Q}_k$ such that $\mathbf{Q}_k = \mathbf{Q}^r_k \left(\mathbf{Q}_k\right)^T$.

To avoid an ever-increasing number of columns in the $\mathbf{L}$-matrix $\mathbf{L}^w_k$, a new square-root matrix $\mathbf{L}^f_{k+1}$ will be calculated from $\mathbf{L}^w_k$, with a fixed number of columns. This narrower $\mathbf{L}$-matrix reduces the rank of the approximated covariance matrix to (at most) the chosen number of columns. Hence the name 'Reduced Rank Square Root' filter.

The new square-root matrix $\mathbf{L}^f_{k+1}$ is chosen to minimize the error-measure

$$\Delta\mathbf{P} := \|\mathbf{P}_{k+1} - \mathbf{L}^f_{k+1}\left(\mathbf{L}^f_{k+1}\right)^T\|_{sF}. \tag{4.9}$$

The norm used, $\|\mathbf{A}\|_{sF} := \sqrt{\sum_{i,j}(s_i a_{i,j})^2}$, is called the scaled Frobenius norm. The scaling parameters s_i form a diagonal matrix, indicated by the letter $\mathbf{S}$. The determination of the scaling is important and model dependent.

The $\mathbf{L}$-matrix $\mathbf{L}^f$ is found by

$$\mathbf{L}^f = \mathbf{L}^w \mathbf{U}_{:,1:n} \tag{4.10}$$

where $\mathbf{U}_{:,1:n}$ is used to indicate the first n columns of $\mathbf{U}$, with n the chosen maximum number of columns in $\mathbf{L}^f$. The columns of $\mathbf{U}_{:,1:n}$ are the eigenvectors of $(\mathbf{SL}^w)^T(\mathbf{SL}^w)$ that correspond to the largest eigenvalues, where $\mathbf{S}$ denotes the diagonal scaling matrix corresponding to scaling vector $\mathbf{s}$. Since the matrix $(\mathbf{SL}^w)^T(\mathbf{SL}^w)$ is symmetric positive definite, the matrix $\mathbf{U}$ is orthonormal ($\mathbf{UU}^T = \mathbf{I}$), and its eigenvalues are real and nonnegative. Therefore an eigendecomposition exists of the following form:

$$(\mathbf{SL}^w)^T(\mathbf{SL}^w) = \mathbf{UDU}^T \tag{4.11}$$

where $\mathbf{D}$ is a diagonal matrix with decreasing, positive elements ($d_{1,1} \geq d_{2,2} \geq \cdots \geq 0$).

4.3.8 Propagation of the L-matrix

The columns of the root covariance matrix $\mathbf{L}$ contain in general nonphysical values. For nonlinear models it is not possible to compute the propagation of the columns of $\mathbf{L}$ (Equation 4.8) meaningfully using:

$$(\mathbf{L}_k^w)_{:,j} = M_k\left(\left(\mathbf{L}_k^f\right)_{:,j}\right) \tag{4.12}$$

where the subscript $:,j$ indicates the j-the column of the $\mathbf{L}$ matrix. A tangent linear model (TLM) can be used for the propagation of the columns of $\mathbf{L}$. However, the creation of a TLM is not trivial. The RRSQRT implementation will therefore use a finite difference approximation around the central model state $\mathbf{x}_k^f$. The columns of $\mathbf{L}$ are propagated according to:

$$(\mathbf{L}_k^w)_{:,j} = \frac{M_k\left(\mathbf{x}_k^f + \delta\left(\mathbf{L}_k^f\right)_{:,j}\right) - M_k\left(\mathbf{x}_k^f\right)}{\delta} \tag{4.13}$$

assuming that $\mathbf{x}_k^f + \delta\mathbf{L}_k^f$ for a selected δ is near a physical value and can be propagated meaningfully by the model M_k .

4.4 Linking models to COSTA

There are various ways to create a COSTA model component from an existing model. The necessary approach depends on the type of model and whether the source code of the model is available and can be changed. In this section we will illustrate how a COSTA model component can be created from an existing model. For this purpose we use the WAQUA/TRIWAQ

model. The approach can be used for a whole class of models that have the same timestepping structure as WAQUA/TRIWAQ.

4.4.1 WAQUA/TRIWAQ

The Dutch National Institute of for Coastal and Marine Management (Rijkswaterstaat/RIKZ) uses various simulation models for operational purposes. The most important application is WAQUA/TRIWAQ.

WAQUA [Stelling, 1983] is used for two-dimensional and TRIWAQ for three-dimensional hydrodynamic and water quality simulation of estuaries, coastal seas and rivers. WAQUA/TRIWAQ is based on the shallow water equations.

WAQUA/TRIWAQ is able to interact with other models for short waves, sediment transport, morphodynamics and ecological processes. WAQUA/TRIWAQ can simulate the hydrodynamics and the distribution of dissolved substances in geographical areas based on rectilinear, curvilinear or spherical coordinates.

The geographical areas in WAQUA/TRIWAQ are bounded by any combination of closed boundaries (land) and open boundaries. Open boundaries force the flows in the model by water levels, velocities, Riemann invariants, discharges or distributed discharges. The system accounts for sources of discharge, such as rivers or outfalls, for tidal flats, for islands and dams, movable barriers or sluices and weirs.

WAQUA/TRIWAQ is by itself not a model but software implementing a numerical method that makes it possible to do simulation of shallow water. The actual models are defined in the form of an input file, that describes all aspects of the model like geometry, forcings etc. In this article we will however often refer to the WAQUA/TRIWAQ model. In this case we mean the WAQUA/TRIWAQ software in combination with an arbitrary input file specifying the model.

Rijkswaterstaat has developed a large number of WAQUA/TRIWAQ models of rivers, estuaria and parts of the North Sea. Examples of models simulated with WAQUA/TRIWAQ are the Dutch Continental Shelf Model (DCSM), modelling a large part of the north sea and the Kuststrook model, a model of the coastal region of the Netherlands. The grids of these models are shown in Figure 4.1 and 4.2.

Figure 4.1: The grid of the Dutch Continental Shelf Model (DCSM). The DCSM model is a 2-dimensional model that is being used for day-to-day sea level forecasts.

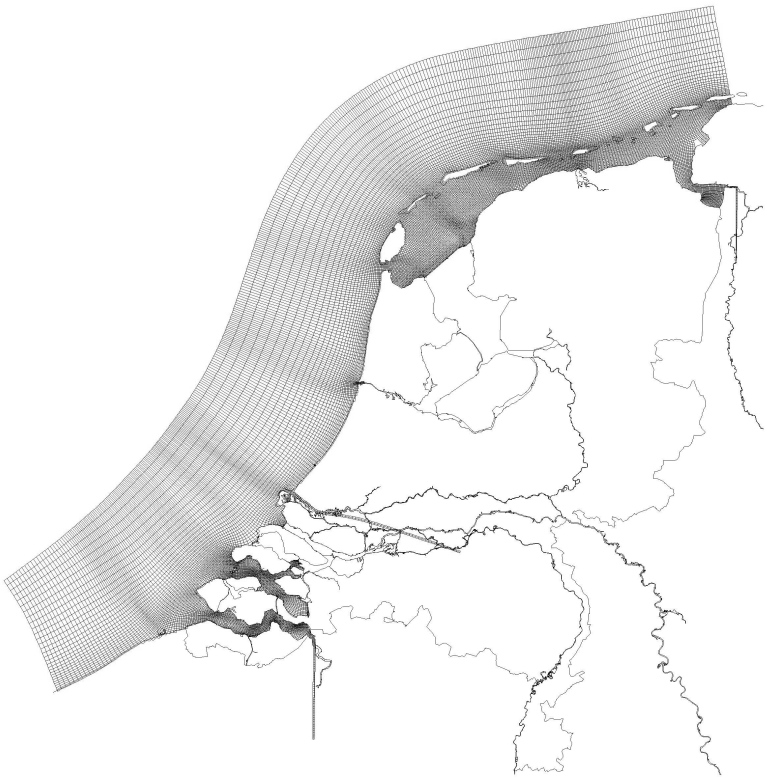

Figure 4.2: The grid of the Kuststrook model, a detailed 3-dimensional model of the coastal region of the Netherlands. The model is used for various applications.

4.4.2 Creating a COSTA model component of an existing model

In COSTA, all interaction between a data assimilation method and a model takes place by use of the COSTA model interface. Therefore in order to use a model in COSTA, it is necessary to adjust or wrap an existing model such that it implements the COSTA model interface. We will illustrate how a model of which the source code is available can be wrapped. We use the WAQUA/TRIWAQ model as an example.

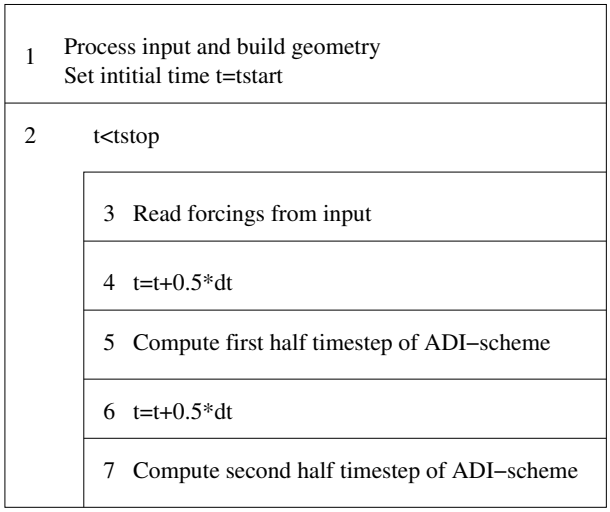

Figure 4.3: Nassi-Shneiderman diagram of the stand alone WAQUA/TRIWAQ program. After the initialization where the model is constructed from the input, the model is simulated with fixed timesteps of length dt. The shallow water equations are solved using an ADI-scheme. Each timestep is therefore spit up into two half time steps. The necessary data needed for computing the model forcings is read from the input at each timestep.

The program flow of WAQUA/TRIWAQ is given in the form of a Nassi-Shneiderman diagram [Nassi and Shneiderman, 1973] in Figure 4.3. the shallow water equations are solved in WAQUA/TRIWAQ using an ADI-scheme. Each timestep is therefore divided into two half timesteps. At the start of the global timestep the necessary model forcings are read from the input.

In this example we will first create a COSTA model component of the deterministic WAQUA/TRIWAQ model. The deterministic model is extended into a stochastic model afterwards. For application with the forward

assimilation methods in COSTA it is sufficient to implement the methods; `Create`, `GetState`, `SetState`, `Compute`, `AxpyForc` and `GetObsValues`.

I	Copy COSTA−state of model instance to internal variables Set t
3	Read forcings from input
II	Add offset to forcings (as set with axpyforc)
4	t=t+0.5*dt
5	Compute first half timestep of ADI−scheme
6	t=t+0.5*dt
7	Compute second half timestep of ADI−scheme
III	Copy internal variables to COSTA−state of model instance

Figure 4.4: Nassi-Shneiderman diagram of the `Compute` method. Most of the program code is reused. Three steps are added in order to copy the state of the model instance into the variables of WAQUA/TRIWAQ and back and to add the specified offset on the forcings of the model.

The `Compute` method computes a model timestep of a model instance. The Nassi-Shneiderman diagram in Figure 4.4 illustrates how the `Compute` method is created using the existing code of WAQUA/TRIWAQ. Each model instance holds its own state, time and the offset to the forcings if these are specified by the `AxpyForc` method. Three parts of code, denoted by Roman numerals, have to be added in order to create the `compute` method.

The first part of added code (I) will copy the state of the model instance into the variables that hold the state in the existing WAQUA/TRIWAQ code. At the end of the timestep the model state is copied from the internal variables into the state of the model instance (III). The model component of WAQUA/TRIWAQ also implements the `AxpyForc` method that is used to add noise to the forcings of the model. In order to implement this, we add the specified change to the forcings after the model forcings are computed (II).

The `Create` method will create a new instance of the COSTA model with its own internal state. The first time `Create` is called it will perform the general WAQUA/TRIWAQ initialization, Step 1 in Figure 4.3, building the geometry of the model and initializing WAQUA/TRIWAQ. The model

state is defined by the time, the state-vector and a specified change to the forcings. These items are allocated at the `Create`.

The methods `GetState`, `SetState` and `AxpyForc` can be created quickly since they involve the copying of the data from and to the data that is associated with a model instance.

Finally the method `GetObsValues` needs to be created. This method interpolates the model state towards the observations. The observations are described in the form of an observation description component. The observation holds the description in "key"-"values" pairs. The amount of work needed to implement this method depends on what information is provided in the observation description component and what interpolation methods are already available. The observation description components in WAQUA/TRIWAQ contains the name of the station that is also specified in the model input. The matching between the model state and the observation can therefore be resolved relatively easy.

4.4.3 Extending the deterministic model into a stochastic model

In the previous section we have given a description of how a COSTA model component can be created from an existing simulation model, using the WAQUA/TRIWAQ model as an example. The created model is however deterministic and does not describe the stochastic forcings $\mathbf{w}(t)$.

Before we can use the model in combination with one of the available data assimilation methods in COSTA we need to add the stochastic forcings to the model. We will use the WAQUA/TRIWAQ model as illustration how the stochastic forcings can be added to a deterministic model from a software engineering point of view.

For most data-assimilation algorithms the stochastic forcing $\mathbf{w}(t)$ needs to be uncorrelated in time. For WAQUA/TRIWAQ we want to model time-correlated noise $\mathbf{n}(t)$. This time correlated noise can be modelled as uncorrelated noise that is passed through a filter. For WAQUA/TRIWAQ we use the popular AR(1) filter:

$$\frac{d\mathbf{n}}{dt} = -\frac{1}{T}\mathbf{n}(t) + \mathbf{w}(t) \tag{4.14}$$

which transforms white noise into a series with exponential temporal correlation.

The deterministic model and an AR(1) model for the stochastic forcing of boundary conditions can be combined into one model. Mathematically

this is done by extending the state

$$\tilde{\mathbf{x}}(t) = [\mathbf{x}(t); \mathbf{n}(t)]. \tag{4.15}$$

The deterministic COSTA model can be extended into a stochastic model by implementing the noise model directly into the existing deterministic model. An alternative approach is not to change the deterministic model component but to create a new model component that is the combination of a noise model and the unchanged deterministic model. The two models can be combined into a larger model in COSTA using a component called the ModelCombiner. The coupling of a deterministic model and a noise model into a stochastic model is illustrated in Figure 4.5.

In the case of WAQUA/TRIWAQ the noise is modelled using an AR(1) model and added to the forcings $\mathbf{u}(t)$. The AR(1) model is by itself a stochastic COSTA model component also implementing the methods `AddNoise`, `GetNoiseCount` and `GetNoiseCovar`.

A timestep of the stochastic WAQUA/TRIWAQ model is given by

$$\begin{aligned} \tilde{\mathbf{x}}(t+\Delta t) &= \begin{bmatrix} \mathbf{x}(t+\Delta t) \\ \mathbf{n}(t+\Delta t) \end{bmatrix} \\ &= \begin{bmatrix} M\left(t, \mathbf{x}(t), \mathbf{u}(t) + \mathbf{n}(t)\right)) \\ N\left(t, \mathbf{n}(t)\right) + \mathbf{w}(t) \end{bmatrix} \end{aligned} \tag{4.16}$$

where M denotes the deterministic WAQUA/TRIWAQ model and N a discrete the AR(1) noise model. The only interaction between the AR(1) model and the deterministic model is the addition of the noise $\mathbf{n}(t)$ to the forcings $\mathbf{u}(t)$ of the model. This interaction is realized using the `AxpyForc` method of the model interface of the WAQUA/TRIWAQ model.

The tree structure of the COSTA state-vector simplifies the coupling of the noise model and the WAQUA/TRIWAQ model. The state of the combined model is a state with two sub-states, the WAQUA/TRIWAQ state and the noise state. The methods of the combined model are trivial and hardly produce overhead. For example the `GetState` method will return the combined state of the model by using the `GetState` method of the deterministic and noise model. The `Compute` method will also execute the `Compute` methods of the two models. It will however use the `AxpyForc` method of the WAQUA/TRIWAQ model in order to add the noise to WAQUA/TRIWAQ before it performs the `Compute` method.

The model that combines WAQUA/TRIWAQ to the AR(1) model is independent of the model that is specified in the input of WAQUA/TRIWAQ. The noise model however, does depend on the model and geographical area

that is simulated. The AR(1) model has its own configuration and needs to be specially configured for the model that is simulated using WAQUA/TRIWAQ.

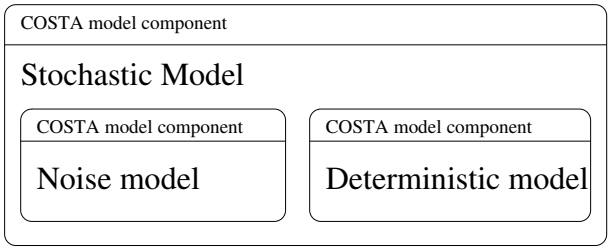

Figure 4.5: Example of a coupled model. A deterministic simulation model which implements the COSTA model interface is coupled with a noise model. The combination is a new COSTA model implementing a stochastic version of the deterministic model. Since the noise model and deterministic model are kept as two separate models it is possible to replace the noise without changing the deterministic model.

4.5 The WAQUA/TRIWAQ data assimilation system

Various data assimilation methods have been added to WAQUA/TRIWAQ in the past including the RRSQRT-filter. The data assimilation system where WAQUA/TRIWAQ is combined with the RRSQRT-Kalman filter is used in an operational environment. This version of the software is not based on COSTA and there is also no clear separation in the software between the various building blocks like deterministic model, stochastic model and data assimilation method. The lack of separation makes the software very complicated. An other disadvantage is that the available data assimilation code cannot be reused for other applications and it is hard and expensive to add new alternative data assimilation methods to WAQUA/TRIWAQ.

In the following subsections we will discuss some of the model specific properties and features of the original version of the data assimilation system of WAQUA/TRIWAQ and the model specific RRSQRT Kalman filter implementation. In Section 4.6 we will explain how this model specific properties are handled in the new COSTA based system.

4.5.1 Scaling of the state-vector

The determination of the narrower **L**-matrix $\mathbf{L}^f$ as explained in Section 4.3.7 uses the scaling matrix **S**. This matrix, which is taken in the form of a diagonal matrix, describes the relative importance of each of the state-vector's entries. The matrix **SL** is the square-root matrix of the covariance matrix of the scaled state-vector $\mathbf{z} := \mathbf{S}\mathbf{x}$.

In the scaled state-vector, it should be possible to compare its vector entries meaningfully. The state-vector $\mathbf{x}$ of WAQUA/TRIWAQ, however, contains dissimilar quantities like water levels, flow velocities, salt concentrations, turbulent quantities k and ϵ and noise parameters.

The scaling is determined according to the amount of energy that is related to the variables in the state-vector.

The scaling factors for the noise parameters are determined differently. The idea behind the scaling is 'conservation' of uncertainty. The scaling factor of the j-th noise parameter is determined such that:

$$\left\| M_0\left(\mathbf{x}_0^f + \mathbf{e}_j p\right) - M_0\left(\mathbf{x}_0^f\right)\right\|_s = \|\mathbf{e}_j p\|_s \tag{4.17}$$

where we use the scaled Euclidean norm $\|\mathbf{x}\|_s := \sqrt{\sum_i (s_i x_i)^2}$ and $\mathbf{e}_j$ denotes a unit vector with 1 at the position of the j-th noise parameter in the state and p a selected perturbation on the noise parameter. The scaling parameter s_j for the j-th noise parameter is then defined by:

$$s_j^2 = \frac{\left\| \mathbf{1}^\phi \left(M_0\left(\mathbf{x}_0^f + \mathbf{L}_0^f\right) - M_0\left(\mathbf{x}_0^f\right)\right)\right\|_s^2}{(1-\alpha^2)\, p^2} \tag{4.18}$$

where $\mathbf{1}^\phi$ denotes an identity matrix with zeros on the diagonal at the positions of the noise parameters in the extended state-vector and α the noise time correlation parameter.

4.5.2 Drying and flooding

During the simulation period of a WAQUA/TRIWAQ model, it is possible that dry areas will flood and flooded areas will become dry. The numerical solution method cannot handle this in a continuous way. Almost dry cells are taken out of the computations by putting screens between computational cells. No water can flow trough a screen. When areas are flooded these screens are removed. Figure 4.6 shows a computational area where screens are placed to isolate temporarily dry areas.

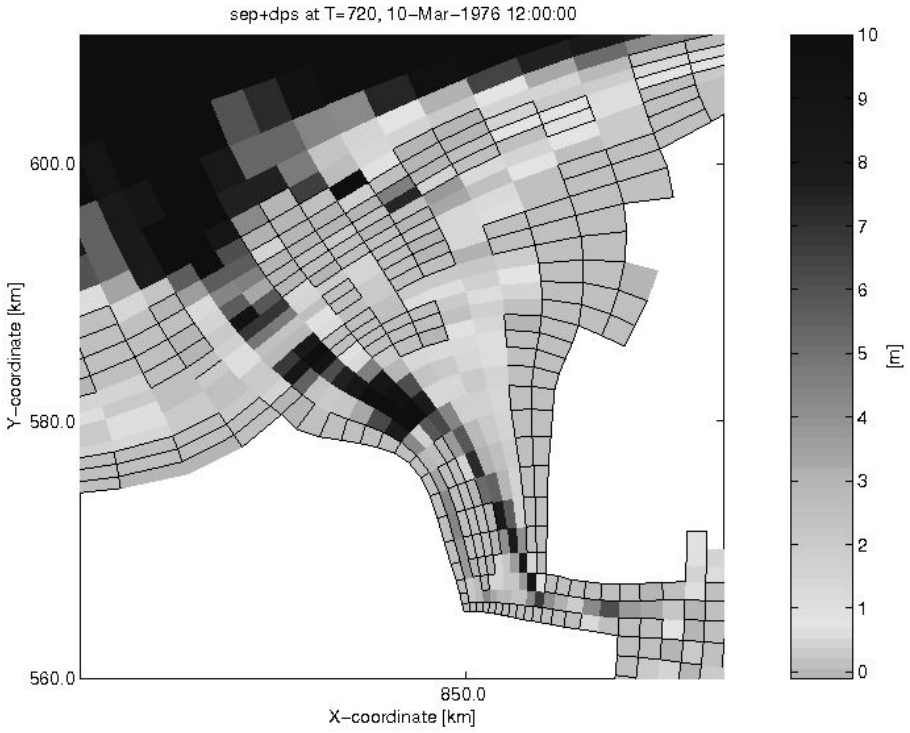

Figure 4.6: part of the Kuststrook-model where the river Ems discharges into the Wadden sea. Some areas are dry and taken out of the computations by screens that are placed between computational cells. The screens are denoted by the black lines in the figure.

The addition and removal of screens is handled during the iterative process and the screens cannot be determined directly from the model state. The screens must therefore be seen as a part of the model state:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_{IR} \\ \mathbf{x}_{IN} \end{pmatrix} \tag{4.19}$$

where $\mathbf{x}_{IR}$ denotes the real part of the model state containing the waterlevels, velocities, salinities, etc. and $\mathbf{x}_{IN}$ the discrete part of the state containing the screens.

Data assimilation methods can handle $\mathbf{x}_{IR}$ but cannot handle $\mathbf{x}_{IN}$. The screens are therefore ignored in most computations in the data assimilation method.

When the difference is computed between two model states, as in the propagation of the $\mathbf{L}$ matrix (Equation 4.13) and the computation of the noise scaling parameters (Equation 4.18) the screens cannot be ignored. Changes in the screens can locally result in large differences between states. The RRSQRT Kalman filter will become unstable without special handling at these gridcells.

To solve this problem the RRSQRT Kalman filter implementation in WAQUA/TRIWAQ will take the screens into account when subtracting two model states, setting the difference to 0 for all grid cells where the screens are different. The RRSQRT implementation is therefore model specific.

4.5.3 Noise model

The stochastic model of WAQUA/TRIWAQ is similar to the stochastic model that is described in Section 4.4.3. An AR(1) process is used to describe the noise on the forcings $\mathbf{n}$. The noise is defined on a courser grid than the model and is therefore interpolated before being added to the model in order to reduce the number of noise parameters.

WAQUA/TRIWAQ implements an ADI method for solving the 2D and 3D shallow-water equations. Using the ADI method, the time discretization is split into an implicit step for d/dx and then another implicit step for d/dy, where x and y denote the horizontal and vertical grid directions respectively. The noise time correlation α is applied to the noise parameters, after each half time-step. The noise model is made part of the ADI solution scheme as a result.

4.6 The COSTA model component of WAQUA/TRIWAQ and the RRSQRT implementation

The existing data assimilation system of WAQUA/TRIWAQ that contains the RRSQRT method is split into parts. We have created a COSTA model component according to the strategy that is presented in Section 4.4. We have also transformed the existing RRSQRT-filter implementation into a COSTA data assimilation method that can be used in combination with other models that implement the COSTA model interface.

In this section we will explain how this spitting up in parts is performed. In Section 4.5 we have described the model specific issues including the handling of the drying and flooding. Appendix 1 gives a detailed description on the handling of drying in the COSTA component of WAQUA/TRIWAQ.

4.6.1 Stepwise transformation

The breaking up in parts of the existing software is performed in various steps. The steps are selected such that we have a working data assimilation system at the end of each step. In this way we can do tests in order to track down errors at an early stage.

In the first step we have restructured the existing code. The filter code and model code are isolated as much as possible and the model specific handling in the filter code is identified.

In the second step we have introduced the COSTA components for holding the data that is exchanged between model and data assimilation method. In this stage we introduce the COSTA tree-vector for holding the model state and the observations are handled using the stochastic observer.

The COSTA model component of WAQUA/TRIWAQ is realized in the third step. All interaction between the model and the RRSQRT filter is performed using the available methods in the model interface.

The final step consists of splitting up the software. The RRSQRT filter is integrated into the COSTA environment whereafter it is available for all users of COSTA. The WAQUA/TRIWAQ model that now implements the COSTA model interface is compiled into a library.

At all stages we have taken special care concerning changes to the model code of WAQUA/TRIWAQ. At each point we have tried to minimize the changes such that changes to the official version of the model can be integrated easily into the COSTA version.

4.6.2 The COSTA model component

The size and content of the state-vector in WAQUA/TRIWAQ is not fixed. The dimension and even the existence of parts of the state are dependent on the model that is specified in the input. For example, the part of the state that holds the concentrations of dissolved substances is only present when these are specified in the input. The structure of the state vector including all elements that can be part of the state is given in Figure 4.7.

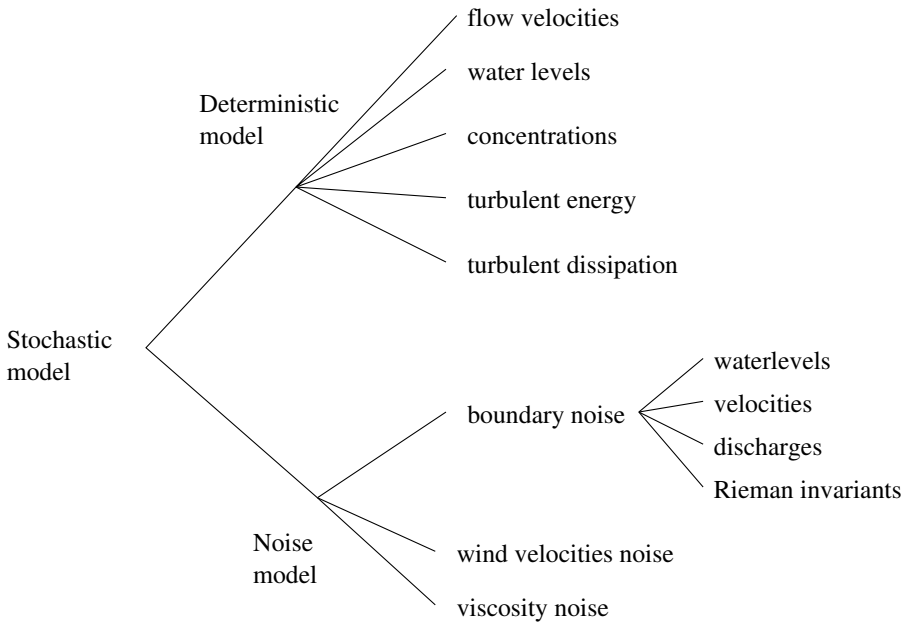

Figure 4.7: Overview of the sub-states that together form the state of the stochastic WAQUA/TRIWAQ model.

From the structure of the state-vector in Figure 4.7 we can identify the part of the state that belongs to the noise model and the part that belongs to the deterministic model. The state of the stochastic model of WAQUA/TRIWAQ is the concatenation of the noise parameters and the state of the deterministic model.

The COSTA state-vector of the deterministic WAQUA/TRIWAQ model is build up from sub-states. Each of the sub-states is associated with the corresponding array in the WAQUA/TRIWAQ source code.

Noise is added to forcings of the WAQUA/TRIWAQ model by use of the `AxpyForc` method in the model interface. The various forcings that can be specified in the input are divided in groups.

In the original software there is no separation between the deterministic model and the noise model. The data assimilation method only interacts with the stochastic model and has therefore no knowledge on the existence of the deterministic model. Since the data assimilation does not need to have knowledge on the deterministic model, it is possible to transform the existing combined model directly into a COSTA model component. However we have chosen to create a separate deterministic model and combine it with an AR(1) model into a stochastic model, as described in Section 4.4.3, for more flexibility.

4.6.3 COSTA RRSQRT Kalman filter

The RRSQRT Kalman filter of WAQUA/TRIWAQ is now a generic filter and a part of the COSTA environment. The RRSQRT filter can be used with other COSTA models as well. The filter algorithm is presented as a Nassi-Shneiderman diagram in Figure 4.8.

Some of the steps are marked as an optional user function. It is possible to provide a routine for performing these tasks. A generic implementation will be used whenever a user function is not provided. A proper scaling vector for the RRSQRT algorithm is necessary for most models to work. When no routine is provided for computing a scaling vector the RRSQRT algorithm will scale by identity. This will however only be useful for a very special class of problems. In order to have good results with any truncation based assimilation algorithm like RRSQRT, COFFEE and POEnK it is important to provide a good scaling procedure for the model.

For WAQUA/TRIWAQ we have implemented all user functions. the scaling of the deterministic model is based on the energy principle as described in Section 4.5.1. This scaling method involves knowledge of the model itself and can therefore not be handled in a generic way. The output routines of step 6 and 8 prints uncertainty in the various quantities in the the model as represented by the **L**-matrix. The end of timestep is used for writing all the simulation results after assimilation to the WAQUA/TRIWAQ data-files.

4.7 Results

In this section we will present the test we have carried out with the COSTA version of the RRSQRT filter and the COSTA model component of the WAQUA/TRIWAQ model. The results are split into two sections.

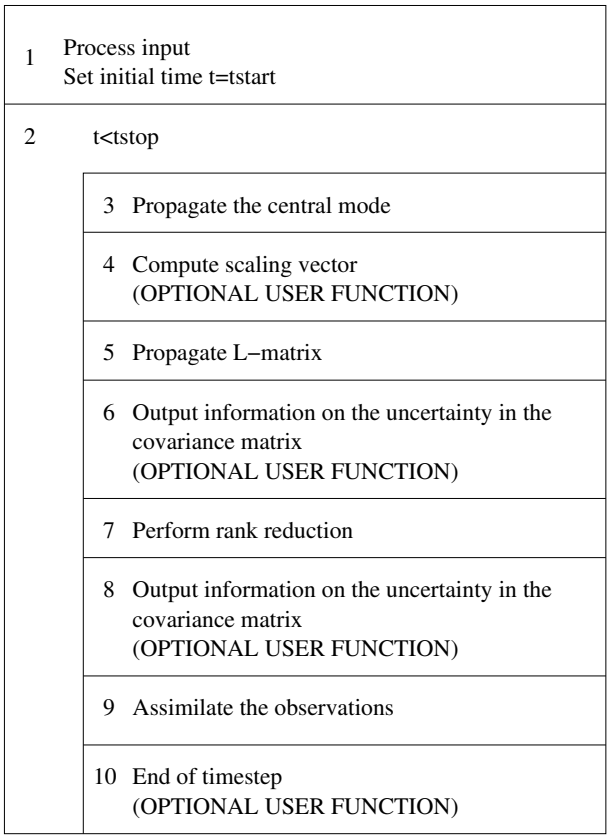

Figure 4.8: Nassi-Shneiderman diagram of the RRSQRT method as available in the COSTA environment. Some of the steps are marked as an optional user function. These steps can be replaced by code of the user. Addition of a proper user implementation of the scaling will be necessary for most models. The other two user routines can be used for customizing the output of the filter.

In Section 4.7.1 we want to show that the code of the COSTA version of the RRSQRT filter and the COSTA version of the WAQUA-TRIWAQ model that is now used within COSTA produces the same results as the original version of the software. We will also compare the computational performance of the new system with the operational system.

In Section 4.7.2 we discuss tests were we have combined the new RRSQRT filter with other COSTA models in order to show that the new COSTA version of the RRSQRT filter can be combined with other COSTA models.

4.7.1 Testing the COSTA based version of WAQUA/TRIWAQ

The tests are performed using the Dutch Continental Shelf Model (DCSM). DCSM is used for storm surge forecasting in the Netherlands. The simulation area and assimilation data locations of the DCSM model are presented in Figure 4.9. The DCSM is a relatively small model with a state-size of the order of 10^5 elements. The model can be run on a desktop computer in moderate time. For that reason it is selected as test model.

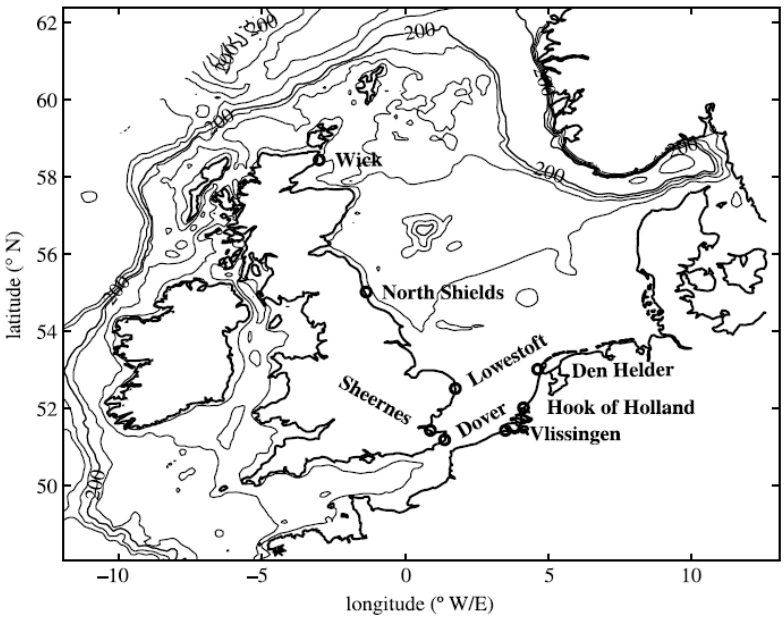

Figure 4.9: DCSM area with the locations of the observation stations.

We have performed several runs in order to check whether the new system, where we use the COSTA RRSQRT filter and the COSTA model component of WAQUA/TRIWAQ produces the correct results. This new system is denoted by "COSTA version". We have made the choise to change the noise model such that it is no longer a part of the ADI-scheme. As a result it was possible to create a deterministic model component of WAQUA/TRIWAQ model and extend this into a stochastic model component.

The operational version of WAQUA/TRIWAQ with RRSQRT filter, denoted by "original version", cannot be directly compared to the "COSTA version" since we know that the results will be slightly different due to the changed noise model. To be able to check the "COSTA version" we have created a special version of the operational version called "Original version with adjusted noise model". This version is identical to the "original version" except for the adjusted noise model.

In Figures 4.10 and 4.11 we have plotted the results of these three different versions at the location of two observation stations. We can see that the "original version with adjusted noise model" produces exactly the same results as the "COSTA version". All differences between the operationally used version and the new COSTA version of WAQUA/TRIWAQ and the RRSQRT filter are the result of the change in the noise model and not the result of design or programming errors.

The created COSTA RRSQRT implementation is currently not fully optimized for speed, however we have set up a small experiment in order to get some insight in the extra computational costs of the new generic system based on COSTA compared to the original operational version of the software.

The DCSM model is used in this experiment where we compare the computational performance of the "COSTA version" based implementation to the "original version". We simulate a period of 12 hours with time-steps of 10 minutes using the RRSQRT Kalman filter. The test is repeated a number of times with various numbers of modes, i.e. the number of columns in the $\mathbf{L}^f$-matrix. The observations of 8 observation stations are assimilated at every timestep.

The computation time of the various runs is presented in Table 4.4. The additional computational cost of the COSTA version is in the order of 10% and seems to be decreasing for a larger number of modes.

The amount of extra computation time is therefore not a bottleneck for applying the COSTA version of WAQUA/TRIWAQ with RRSQRT filter in an operational environment.

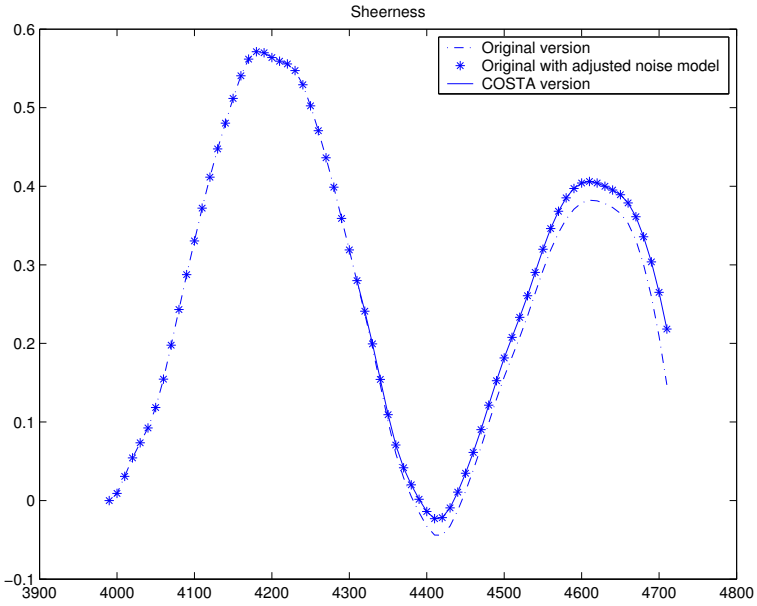

Figure 4.10: Water level after analysis at Sheerness for three different versions of the software; the original version of WAQUA/TRIWAQ, the original version where only the time correlation of the AR(1) process is applied at the whole time steps and the new COSTA version of the system. We can see that the results of the adjusted original version is exactly the same as the COSTA version.

Number of modes	COSTA (s)	Original (s)	Extra time (%)
10	82.7	74.3	+ 11
20	149.8	136.2	+ 10
40	295.3	271.9	+ 9
80	623.4	586.0	+ 6

Table 4.4: Computation time of a run with the WAQUA/TRIWAQ DCSM model with RRSQRT filter for various numbers of modes, columns of the **L** matrix, of the COSTA and original version of the software.

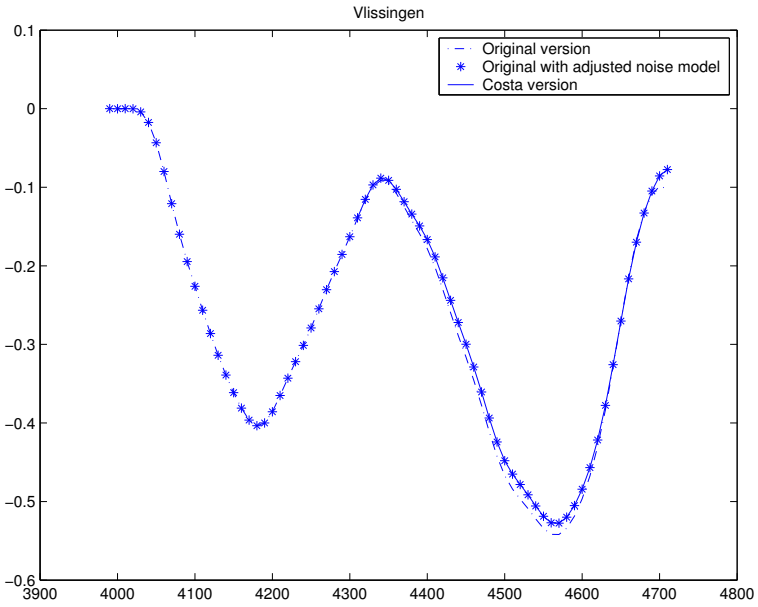

Figure 4.11: Water level after analysis at Vlissingen three different versions of the software; the original version of WAQUA/TRIWAQ, the original version where only the time correlation of the AR(1) process is applied at the whole time steps and the new COSTA version of the system. We can see that the results of the adjusted original version is exactly the same as the COSTA version.

Operational models	
WAQUA/TRIWAQ	2D and 3D Shallow water simulation model
LOTOS-EUROS	Air pollution pollution model
SobekRe	1D river model
COSTA test models	
advec-1d	1-dimensional advection model
oscill model	Spring mass system
heat model	2-dimensional heat model
Lorenz96	A very non-linear test model

Table 4.5: Overview of models that can be used with COSTA. These models are split up into operational models and test models. Note: at this time the SobekRe model can only be used for model calibration.

4.7.2 Applying the RRSQRT method to other COSTA models

The new RRSQRT filter implementation is tested in combination with the available test models that are part of the COSTA environment. At this time there are three big operational models available in the form of a COSTA component. These models are the WAQUA/TRIWAQ model, the LOTOS-EUROS model and SobekRe [Stelling and DuinMeijer, 2003]. All test models are presented in Table 4.5. The SobekRe model is currently only available as a deterministic model and has only be used for model calibration. It is therefore not yet tested in combination with the RRSQRT Kalman filter.

The LOTOS-EUROS model is an operational computational model for atmospherical transport, chemistry and deposition of air pollution on the scale of Europe developed by the TNO Institute of Environmental Sciences, Energy and Process Innovation and the Netherlands Environmental Assessment Agency Institute of Public Health and the Environment (RIVM). It is an Eulerian grid model with a resolution of 7.5 times 7.5 km and 4 vertical layers. The model focuses on modelling the lower part of the troposphere. It predicts the air quality and deposition of pollutants on an hourly basis. Figure 4.12 illustrates computational results of the LOTOS-EUROS model for the ozone concentration in Europe.

LOTOS-EUROS is a new model that is the combination of the air quality models LOTOS (LOng Term Ozone Simulation) [Schaap et al., 2004] and EUROS [Hanea et al., 2004].

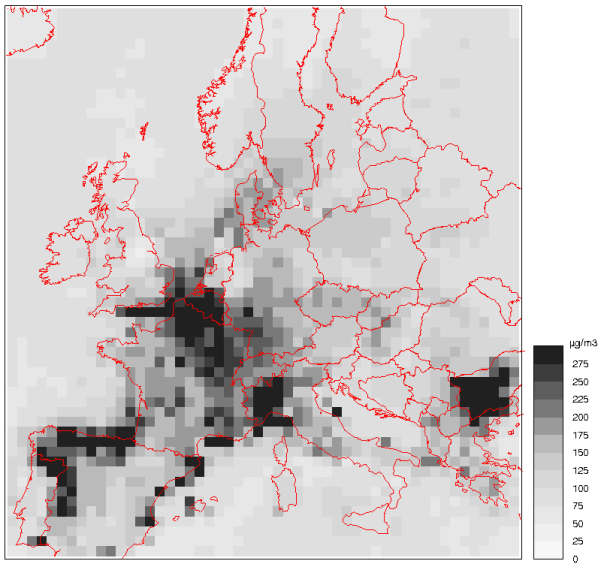

Figure 4.12: Example of the results that are computed by the LOTOS-EUROS model. Here we have plotted the concentration of ozone (μgm^{-3}) at the observation height in Europe for June, 6 1996, 15:00 hour

The creation of the COSTA model component of the LOTOS-EUROS model is similar to the procedure we have described in Section 4.4.2. The deterministic model is extended into a stochastic model by adding noise to the emmisions.

Without the need to change the LOTOS-EUROS model or the RRSQRT filter implementation in COSTA is was possible to combine both into a new working data assimilation system.

4.8 Future developments

COSTA is a system under development. The basis of the system is realized but a lot of work still needs to be done. In this section we will mention some research and development issues that are currently carried out.

The scaling of the model state is not only necessary for the RRSQRT filter. Other methods like COFFEE and POEnK also need a scaling vector for the model state. The model interface will be extended with a method that provides a scaling vector for the state. This will make it possible to handle the scaling in a uniform way in the various data assimilation methods.

The computational speed of the new system will be improved in the future when the code of the RRSQRT Kalman filter is more optimized.

LOTOS-EUROS is combined with the COSTA RRSQRT filter. There is still work to do on the determination of a suitable scaling vector and the validation of the results.

The current version of COSTA only supports sequentional methods. We are now working on the extension of the model interface and the addition of variational methods.

In this article we have illustrated how a COSTA model component can be created for a model of which the source code is available. A tool called the Blackbox model builder is being developed that provides tools for linking models to COSTA of which the source code is not available.

More operational models from different application fields need to be added in order to improve the design and prove intended usability and flexibility of COSTA. We are currently working on a COSTA model component of SWAN [Booij et al., 1999] a wave model for coastal regions.

We are working together with WL|Delft Hydraulics in the `openda.org`-project in order to have a mutual accepted interface to the main components, and to share algorithms, that will become available in Java. As a result, in the near future, COSTA compliant models can run in the DATools data assimilation environment of WL|Delft Hydraulics and the models that are available for DATools can be run in COSTA. These models include HBV-96 rainfall runoff model [Lindström et al., 1997], the representative elementary watershed (REW) model [Reggiani and Schellekens, 2004] and the SobekRe 1D river model.

4.9 Conclusions

We have shown that the original WAQUA/TRIWAQ software can be split up into a COSTA model component and a generic RRSQRT Kalman filter.

Model specific issues that were part of the original RRSQRT filter like drying and flooding did not need to be part of the data assimilation method and are handled by the COSTA model component of WAQUA/TRIWAQ.

Tests show that the computational overhead for creating generic reusable data assimilation components is relatively small and can even be reduced in the future.

A scaling vector that makes it possible to compare the various values in the state of the model is model specific and should be made a part of a COSTA model component since it is also necessary for other data assimila-

tion methods.

The new RRSQRT Kalman filter is now part of the COSTA environment. We have shown that it produces the correct results when combined with the model component of the WAQUA/TRIWAQ model.

The new RRSQRT Kalman filter is combined with the LOTOS-EUROS model for atmospherical transport, chemistry and deposition of air pollution in order to show that new filter implementations can be used for other models as well.

4.A The AXPY operation between model instances

The `AxpyState` method of the COSTA model interface changes the state of the model instance $\mathbf{y}$ by adding a scaled state $\mathbf{x}$ to it.

$$\mathbf{y} := \alpha\mathbf{x} + \mathbf{y} \tag{4.20}$$

The state $\mathbf{y}$ is the state of model instance for which the method `AxpyState` is applied. For $\mathbf{x}$ there are two options. It is allowed to supply $\mathbf{x}$ in the form of an instance of a COSTA tree-vector as well as directly in the form of a model instance.

In general it is not necessary to supply the `AxpyState` method when creating a COSTA model component. The method is automatically available whenever the methods `GetState` and `SetState` are provided by the model.

However it is possible to provide a model specific implementation of the `AxpyState` method.

The assimilation method needs to take special care of drying and flooding for the WAQUA/TRIWAQ model as explained in Section 4.5.2. The special handling for drying and flooding is handled in the `AxpyState` operation between two model instances. When the `AxpyState` method is applied where $\mathbf{x}$ is provided in the form of a COSTA model we have the drying and flooding information at hand and can therefore compute the subtraction properly.

In order to make it work it is necessary that the data assimilation method uses the `AxpyState` method between two model instances and not between a model and a state. A general rule when developing a data assimilation for COSTA is to use operations between models when possible and only work with tree-vector instances when there is no other option. In this way it is possible to handle model specific issues like drying and flooding by the model component. Applying this rule will also limit the memory usage and copying overhead of the data assimilation method since it will limit the copying of data by limiting the usage of the `GetState` and `SetState` methods.

4.B List of acronyms

ADI	Alternating Direction Implicit
AR	AutoregRessive
CG	Conjugate Gradients
COFFEE	Complementary Orthogonal subspace Filter For Efficient Ensembles
COSTA	COmmon Set of Tools for the Assimilation of data
DAIHM	Data Assimilation In Hydrological and hydrodynamic Modelling
DATools	Data Assimilation Tools
DCSM	Dutch Continental Shelf Model
ESMF	Earth System Modeling Framework
EUROS	EURopean Operational Smog
EnKF	Ensemble Kalman Filter
EnSRF	Ensemble Square Root Filter
GNU	GNU's Not Unix
HBV	Hydrologiska Byråns Vattenbalansavdelning
LBFGS	Limited memory Broyden Fletcher Goldfarb Shanno
LGPL	(GNU) Lesser General Public License
LOTOS	LOng Term Ozone Simulation
NERSC	National Energy Research Scientific Computing center
OMS	Open Modellen Systeem
OHSU	Oregon Health & Science University
PALM	Projet d'Assimilation par Logiciel Multi-methodes
POEnK	Partially Orthogonal Ensemble Kalman
REW	Representative Elementary Watershed
RIVM	RijksInstituut voor Volksgezondheid en Milieu
RRSQRT	Reduced Rank SQuare RooT
ReBEL	Recursive Bayesian Estimation Library
SQL	Structured Query Language
SWAN	Simulating WAves Nearshore
TLM	Tangent Linear Model
TNO	(Nederlandse organisatie voor) Toegepast Natuurwetenschappelijk Onderzoek
TRIWAQ	3-Dimensional WAQUA
WAQAD	WAQua ADjoint
WAQUA	WAter QUAlity

Chapter 5

Comparing Kalman filtering techniques for air quality modelling [1]

5.1 Abstract

A generic toolbox for data assimilation called COSTA (COmmon Set of Tools for the Assimilation of data) makes it possible to simplify the application of data assimilation to models and to try out various methods for a particular model. Concepts of object oriented programming are used to define building blocks for data assimilation systems that can be exchanged and reused. The main building blocks in COSTA are the model component and the stochastic observer component. These components can be created by wrapping existing code. The LOTOS-EUROS air quality model will be used for operational smog and aerosol forecasts in the Netherlands in the near future. The COSTA framework will be used in this operational environment to implement the data assimilation techniques. As a first steps towards this operational system the model component of LOTOS-EUROS is created and the performance of various Kalman Filter based data assimilation techniques are compared for a real live case study. The Ensemble Kalman filter and the Ensemble Square Root filters converged faster than the other tested techniques for the selected case study setup.

[1] The content of this chapter is publised in Environmental Modelling & Software [van Velzen and Segers, 2009]; A problem-solving environment for data assimilation in air quality modelling, by Nils van Velzen and Arjo Segers, Environmental Modelling & Software, 2009, DOI: 10.1016/j.envsoft.2009.08.008

5.2 Introduction

Most applications of data assimilation make use of a dedicated implementation of the data assimilation algorithm and processing of the observations for a particular simulation model. Implementing a single data assimilation method like an Ensemble Kalman Filter for a single model is in general not really a problem since it does not take a huge amount of work. Data assimilation algorithms like 4D-Var require an adjoint. In that case most work will be the construction of the adjoint. In general it is not known in advance what data assimilation method will perform best for a given model. The other way around, it is desirable to try out newly proposed data assimilation methods on various models and to compare them to existing data assimilation methods.

In order to find the best suited data assimilation method for a model or to investigate the performance of newly proposed algorithms it is necessary to be able to perform simulations with a range of data assimilation methods. The implementation of a number of different data assimilation algorithms is a significant amount of work, not only because of the programming and debugging of the data assimilation algorithms, but also because of the need to adapt the model such that it can be used in combination with various algorithms.

A generic toolbox for data assimilation called COSTA (COmmon Set of Tools for the Assimilation of data) [van Velzen, 2006, van Velzen and Verlaan, 2007] provides a framework wherein it is possible to combine models with various data assimilation methods. Within this framework it is possible to try out alternative data assimilation methods for a given model without a lot of work. The strategy of COSTA is to define generic building blocks for data assimilation. Building blocks of a similar type can be interchanged. This makes it possible to reuse them in an alternative data assimilation system. Models and data assimilation methods can be freely combined and compared. The software will be better tested and contain less errors because it can be tested in more than one applications. In addition, the reuse will decrease the development time and costs of a data assimilation application.

The models that are used in data assimilation systems can be very complex and sometimes they are a combination of several sub models. There are various software tools available that simplify the connection of models to other software building blocks like e.g. a data assimilation method. Examples of these tools are PALM of Cerfacs [Lagarde, 2000], ESMF [Hill et al., 2004], OMS [Hummel et al., 2002] of the Dutch Rijkswaterstaat and WL|Delft Hydraulics and the HarmonIT project [Gijsbers, 2004]. The fo-

cuses of these tools is to provide means for linking arbitrary software components together and to share or send data between these software components.

The focus of COSTA is not to link together arbitrary software components but the identification and definition of the generic components that are needed to construct a data assimilation application. COSTA is therefore supplementary to PALM, ESMF, OMS and HarmonIT since these tools can be used in combination with COSTA.

In this paper we present the results of a case study with COSTA and the LOTOS-EUROS model. LOTOS-EUROS is a regional air quality model used to simulate concentrations of pollutants on European scale [Schaap et al., 2008]. The model focuses on the lower 2-3 km of the atmosphere, and is therefore able to simulate pollution levels that have a direct impact on human health. Simulated concentrations include oxidants and organic compounds that are responsible for episodes of elevated ozone levels, often referred to as summer smog. The model is able to perform simulations over multiple years in reasonable time to support scenario studies for the impact of emission regularisations and climate changes. Data assimilation systems around LOTOS-EUROS and its predecessor have been implemented in different forms of low-rank Kalman filters [van Loon et al., 2000, Hanea et al., 2004, Denby et al., 2008].

In the near future, the LOTOS-EUROS model will be used for operational smog and aerosol forecasts in the Netherlands. Initially, the resolutions are set to 25 km for the European domain in zoom mode over the Netherlands at double resolution. The COSTA framework will be used in this operational environment to implement the data assimilation techniques. The case study we present in this paper is the first step taken towards the operational system. The development of the COSTA toolbox and the development of the operational modelling system are currently disjunct. The example given in this paper is an illustration of how the application could look like.

The COSTA framework is used in order to compare the performance of various Kalman filtering techniques for a selected real life case study, in this case for the European domain and resolution. The goal of this study is to investigate whether a successful coupling between LOTOS-EUROS can be established, to get insight in what Kalman filtering technique is most likely best suited for the LOTOS-EUROS model and what aspects of the LOTOS-EUROS model and the uncertainty model need to be improved. The case study is used to illustrate how a comparisons between various methods can be performed using COSTA. It is not in the scope of this paper to formulate any substantive conclusions on the general performance of the various data

assimilation methods. For this more research is needed in the future.

In Section 5.3 we give an overview of the COSTA framework and what work needs to be done in order to use an existing model in COSTA. The used data assimilation methods in the comparison are briefly described in Section 5.4. The setup of the case study and the comparison of performance of the various methods are presented in Section 5.5. The conclusions of the presented work will be formulated in Section 5.6.

5.3 The COSTA environment for data assimilation

COSTA is a problem solving environment (PSE) initially developed at Delft University of Technology. It provides a framework for developing data assimilation and model calibration systems and the reusage and exchange of data assimilation software.

The COSTA software is freely available under the LGPL license and can therefore be used in combination with both open source as propriatory software.

The basic design philosophy of COSTA is illustrated in Figure 5.1. The key elements in this picture are: a deterministic or stochastic model, collection of observations, several data assimilation procedures and the core of the COSTA-system that connects the different building blocks. The data assimilation method is implemented on top of existing model software. In order to do so it is necessary to wrap the model and observations in an appropriate way.

COSTA uses the concepts of object oriented software. It defines building blocks called classes. An instance of a class is a single variable of a class, also called an object. The value of an object cannot be changed directly, but only by a given set of functions called the interface of a class.

Components in COSTA are classes that involve a large amount of functionality. Currently COSTA defines two components; the model and the stochastic observer which are described in more detail in Section 5.3.1 and Section 5.3.2. The interface of the model component consists of the list of methods/routines that a model must provide. Examples are “perform a time-step”, “deliver the model state”, or “accept a modified state”. This interface of the model component is visualized through the shape of the empty space in the bricks in Figure 5.1.

Usually an existing model code does not yet provide the required routines, and certainly not in the prescribed form. Therefore some additional

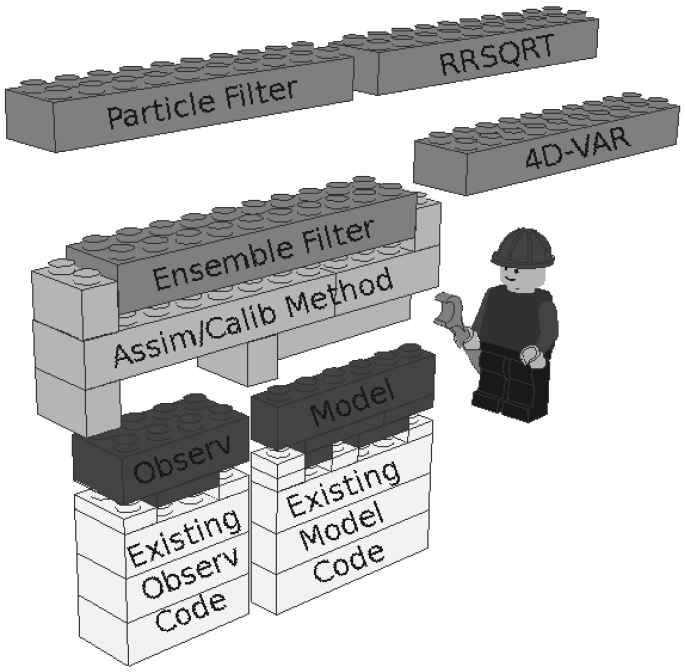

Figure 5.1: Basic design of the COSTA system. Model and observation components (dark/white bricks) are plugged into the core of the COSTA environment ("Assim/Calib Method"), and can then exploit the available data assimilation methods (upper bricks).

code has to be written to convert between the existing code and the COSTA model components interface. This is illustrated in Figure 5.1 using dark on top of the white bricks: the white bricks stand for the original model code, whereas the dark bricks concern the wrapping of the model in order to provide it in the required form.

COSTA similarly prescribes the interface of the COSTA observation component. Similar to the model it is usually necessary to wrap existing sources of observations in the required form such that it can be used in the COSTA framework.

The data assimilation algorithms in Figure 5.1 implement different data assimilation techniques with the elements provided by COSTA as the building blocks (models, observations, state vectors etc.).

The basic design of COSTA may seem disadvantageous at first, because it appears to require additional programming work in comparison to an approach where data assimilation is added to a computational model in an ad-hoc way. This is usually not the case. Most of the work in restructuring of the existing model code is needed in an ad-hoc approach too. This is because data assimilation itself puts requirements on the way in which the model equations are implemented in software routines: one must be able to repeat a time step, to adjust the model state, to add noise to the forcings or the model state, and so on, which is often not true for computational models that are not implemented with data assimilation in mind.

The design of COSTA has the advantage over an ad-hoc approach for adding data assimilation to an existing program that the different aspects of a data assimilation algorithm are clearly separated. The algorithmic of the filtering algorithm that is used are separated from the uncertainty model, which in turn may be largely separated from the deterministic model and the processing of observations. This makes it easy to experiment with different choices for each of the parts: adjusting the noise model, adding observations, testing another data assimilation algorithm and so on. This is the major benefit of using the COSTA approach. Using COSTA introduce some computational overhead. This overhead of COSTA mainly caused by copying data from the generic COSTA data structures to the data structures that are used in the original (model) code. However this is often necessary as well in the case of a direct implementation of a data assimilation method in existing model code. The overhead of using COSTA is only in the order of a few percent [van Velzen and Verlaan, 2007].

COSTA is both a philosophy on how to set up data assimilation software as well a software toolbox. The COSTA software consists of implementations of the defined components, various implementations of data assimilation and

model calibration methods and a number of (test) models and various tools to simplify creation of a model component from an existing model code and a tutorial. The COSTA workbench program, that is part of the software as well, is used to start up the simulations. The input of this program is an XML-file in which the user specifies what model, method and data assimilation method is used and their configuration.

Interfaces of the COSTA classes are defined both for Fortran and C/C++. An interface for Java is defined separately in the OpenDA project. This allows data assimilation algorithms to be implemented in Java, and be used together with model components made in Fortran and C.

5.3.1 Description of a COSTA-model

The first task in the application of COSTA is to define a model component. The interface of the model component can be used both for stochastic and deterministic models. Stochastic models contain a model for the uncertainties (differences between model results and reality). Deterministic models are seen as a special case of a stochastic model, in which the uncertainties are ignored (assumed zero).

The 'formal' notation of a model in COSTA is

$$\frac{d\mathbf{x}(t)}{dt} = \mathcal{M}\left(\mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \mathbf{w}(t), t\right) \tag{5.1}$$

where M denotes the model operator, which is in practice often an existing piece of software that solves the model-equations numerically. In most cases an existing model will not describe uncertainties and thus a stochastic extension is necessary.

The input of a model consists of the model state $\mathbf{x}(0)$, time-independent parameters $\mathbf{p}$, time dependent forcing $\mathbf{u}(t)$, and a stochastic forcing $\mathbf{w}(t)$. It is very important to notice that the state of the COSTA model component contains more than only the model state $\mathbf{x}$. The state of the model component also include the parameters $\mathbf{p}$, forcings $\mathbf{u}(\mathbf{t})$ and the time t.

As an example of how the COSTA software should be used a number of example codes have been included. The first example in Table 5.1 illustrates the definition of a COSTA state that consists of a model state $\mathbf{x}$ and a noise input $\mathbf{w}$. In this particular example the model state consists of the four dimensional concentration array of the LOTOS-EUROS model, described in detail later on.

The Kalman filtering techniques that have been used for our case study do not change the forcings or the parameters of the model. It is therefore

```
subroutine my_create(datablk)
use my_model, only:nx,ny,nz,nspec,nnoise, tstart
! Datablock to store the state of the model
integer, dimension(*), intent(out) :: datablk

   ! Handles to costa objects for
   integer :: x       !State vector
   integer :: conc    !Vector with concentrations
   integer :: w       !State vector (noise part)
   integer :: noise   !Vector with noise
   integer :: t       !Time instance of the model
   integer :: ierr    !Error code

   ! Create the state-vector (x)
   call cta_treevector_create('lotos-euros-state',x, ierr)
   call cta_vector_create(nx*ny*nz*nspec, conc, ierr)
   call cta_treevector_setvec(x,conc, ierr)

   ! Create the noise-vector (w)
   call cta_treevector_create('lotos-euros-noise',w, ierr)
   call cta_vector_create(nnoise, noise, ierr)
   call cta_treevec_setvec(w,noise, ierr)

   ! Create the time that is associated to the model instance
   call cta_time_create(t, ierr)
   call cta_time_set(t, tstart, ierr)

   ! Store the handles of x and w in datablk
   datablk(1)=x
   datablk(2)=w
   datablk(3)=tmodel

end subroutine
```

Table 5.1: Example code in Fortran90 of an implementation of the Create routine of a COSTA model. This routine creates the state of a model instance. In this example, the state contains of the state-vector ($\mathbf{x}$) of a concentration array augmented with the noise values ($\mathbf{w}$) and the time (t).

possible the implement the interface of the model with a simplified form

$$\frac{d\mathbf{x}(t)}{dt} = M\left(\mathbf{x}(t), \mathbf{w}(t), t\right) \tag{5.2}$$

An example code of how this timestep operator should be implemented in the COSTA environment is given in Table 5.2.

```
subroutine my_compute(datablk, tend)
use my_model, only:nx,ny,nz,nspec,nnoise,dt
  integer, dimension(*), intent(inout) ::datablk !State of model
  integer,               intent(in)    ::tend !target time

  real, dimension(nx,ny,nz,nspec) ::conc    !Concentration array
  real, dimension(nnoise)         ::noise   !Noise array
  integer                         ::numstep !Number of timesteps
  integer                         ::ierr    !Error code
  real                            ::t       !time of model timestep

  ! Extract model arrays
  call cta_treevector_getvals(datablk(1), conc, ierr)
  call cta_treevector_getvals(datablk(2), noise, ierr)

  ! Determine the number of timesteps to reach tend
  call cta_time_numstep(datablk(3), tend, dt, numstep, ierr)
  call cta_time_gettime(datablk(3), t, ierr)

  ! Perform timesteps (call existing model code)
  do i=1,numstep
     t=t+dt
     call my_model(conc,noise,t)
  enddo

  ! Restore model arrays and new time
  call cta_treevector_setvals(datablk(1), conc, ierr)
  call cta_treevector_setvals(datablk(2), noise, ierr)
  call cta_time_copy(tend, datablk(3), ierr)

end subroutine
```

Table 5.2: Example code in Fortran90 of an implementation of the Compute routine. The result is the propagation of the model state vector for a number of timesteps. This routine in general consists of three parts. First the state of the model is copied to the model arrays, then the existing model routines are called and finally the propagated state is saved.

Besides the definition of the model state and the timestep operator, other methods need to be implemented too. Table 5.3 summarizes the most essential methods of a COSTA model. For example the method `AxpyState`

`Create`	Create a new instance of a model having its own internal state.
`Free`	Free a model instance
`Compute`	Carry out the time steps necessary to step through a given time span (Equation 5.2).
`GetObsValues`	Interpolate the model prediction that corresponds to a given set of observations.
`GetObsSelect`	Given a set of observations, return the selection of observations that can be handled by the `GetObsValues` method of the model. This method is used to filter out observations of quantities that are not described by the model or are measured outside the modeled area.
`AddNoise`	The addition of random noise $\mathbf{w}$ in the `Compute` can be switched on and off using this method.
`GetNoiseCount`	Return the number of noise parameters of the stochastic model.
`GetNoiseCovar`	Return the (root of) noise the covariance matrix $\mathbf{Q}$.
`SetState`	Set the state $\mathbf{x}$ of the model instance.
`GetState`	Get a copy of the model state $\mathbf{x}$.
`AxpyState`	Modify the model state by adding α times a state-vector to the state of the model.

Table 5.3: Overview of the methods that need to be implemented such that the model component can be used in combination with the Kalman filtering techniques in COSTA.

is used by assimilation methods to update the state $\mathbf{y}$ of the model by $\mathbf{x}$ or to compute the difference between model states $\mathbf{x}$ and $\mathbf{y}$ according to:

$$\mathbf{y} := \alpha\mathbf{x} + \mathbf{y} \tag{5.3}$$

By default this method does not need to be implemented since it is automatically implemented for a COSTA state. However in some cases the user might extend the default functionality when necessary. For example, addition and subtraction of model states does not always yield results that can be used without problems in the following steps of the data assimilation methods. An updated state can have physically meaningless values e.g. negative concentrations or an unstable stratification. The computed difference between states of two (nonlinear) models sometimes needs some special treatment in order to avoid unstable data assimilation methods. These model specific issues can be handled inside a user specific implementation of the `AxpyState` method of the model.

The transformation of an existing model is discussed in detail in van Velzen and Verlaan [2007]. The implementation of the `Compute` method will in general be the most time consuming when the model component is created from an existing simulation model. Depending on the model, some of the code needs to be restructured in order to be able to isolate the model operator M from the code. However, this needs to be done as well when a dedicated implementation of a data assimilation method is implemented. The amount of work for creating a COSTA model component is therefore of the same order as the work that needs to be performed to adjust the model for a dedicated data assimilation method.

5.3.2 Observations handling

The formal notation of an observation operator is:

$$\mathbf{y}(t) \;=\; \mathbf{H}\mathbf{x}(t) \;+\; \mathbf{v}(t) \tag{5.4}$$

where $\mathbf{y}$ denotes the observed values and the operator $\mathbf{H}$ interpolates the state variables to the observations. The observation error $\mathbf{v}$ includes both the instrumental and representation error and is supposed to have zero mean and covariance $\mathbf{R}$.

Table 5.4 illustrates how the operator $\mathbf{H}$ can be implemented in the COSTA framework. This example makes use of the so called observation description that will be explained below.

The observed values are available in COSTA in the form of a so called stochastic observer. An instance of a stochastic observer holds a number of

```
subroutine my_get_obs_values(datablk,obsdescr,vpred)
   use my_model, only:nx,ny,nz,nspec,nnoise

   implicit none
   integer, dimension(*), intent(in)    ::datablk
   integer,               intent(in)    ::obsdescr
   integer,               intent(inout)::vpred

   real, dimension(nx,ny,nz,nspec) ::conc
   real, dimension(nnoise) ::noise
   ....

   ! Get properties of the observations from the observation description
   call cta_obsdescr_getpropertyvalues(obsdescr,'LON',vlon,ierr)
   call cta_obsdescr_getpropertyvalues(obsdescr,'LAT',vlat,ierr)
   call cta_obsdescr_getpropertyvalues(obsdescr,'LEVEL',ilevel,ierr)
   call cta_obsdescr_getpropertyvalues(obsdescr,'COMP',icomp,ierr)

   ! Get the model concentrations
   call cta_treevector_getvals(datablk(1),conc,ierr)

   ! Call model interpolation routine
   call my_interpolation(conc,vlon,vlat,ilevel,icomp,y)

   ! Set the output values
   cta_vector_setvals(vpred,y,ierr)

 end subroutine
```

Table 5.4: Example code in Fortran90 of an implementation of the GetObsValues routine. The result is the simulated value of the model concentration array at the observation sites specified by longitude, latitude height (level) and component name.

observations including their error distribution and covariance. The stochastic observer also includes the available additional information on the observed values like location, unit and physical meaning. This additional information is represented in the form of an observation description component.

The information in the observation description component can be represented by a table where each column has a unique name called key. A row in this table contains all information on a single observed value. The keys can be freely selected and depends on the type of observation. The only mandatory key is time.

It is possible to create a new instance of a stochastic observer that is a subset of an existing stochastic observer. The selection can be based on the time associated with the observations or more generally based on a selection criterion formulated in a subset of SQL, a standard querying language for managing databases.

The information in the observation description component is conceptually represented as a table. This does not imply that it needs to be stored in memory in the form of a table. COSTA includes a default implementation of the stochastic observer and observation description components. The observations and their description are stored in a database containing two tables. One table contains time independent information that can be shared by various observations. The other table contains all time dependent information for each observation. The data is externally represented as being one large table but it is stored in a space efficient way.

In order to be able to use the default implementation of the stochastic observer it is necessary to convert existing sources of observations into a COSTA observation database. An alternative approach is to implement the interface of the stochastic observer and observation description component for an existing storage format or database.

5.4 Kalman filtering techniques in COSTA

The core of the COSTA environment are the available data assimilation and model calibration methods. These methods can be used with arbitrary models that implement the COSTA model interface. This simplifies the comparison of various methods for a given model. In the case study with the LOTOS-EUROS model that is presented in Section 5.5, we compare the performance of various Kalman filtering techniques. The compared methods are the Ensemble Kalman Filter (EnKF) [Evensen, 2003], Ensemble Square Root Filter (EnSRF) [Whitaker and Hamill, 2002], Complementary

Orthogonal subspace Filter For Efficient Ensembles (COFFEE) [Heemink et al., 2001] and the RRSQRT Kalman filter [Verlaan and Heemink, 1997b, Verlaan, 1998].

5.4.1 Kalman filter

The classical Kalman filter (KF) algorithm for computing a time step from time instance k to $k+1$ consists of the following filter equations:

$$\mathbf{x}^f_{k+1} = \mathbf{M}_k \mathbf{x}^a_k \tag{5.5}$$

$$\mathbf{P}^f_{k+1} = \mathbf{M}_k \mathbf{P}^a_k \mathbf{M}^T_k + \mathbf{Q}_k \tag{5.6}$$

$$\mathbf{x}^a_{k+1} = \mathbf{x}^f_{k+1} + \mathbf{K}_{k+1}\left(\mathbf{y} - \mathbf{H}_{k+1}\mathbf{x}^f_{k+1}\right) \tag{5.7}$$

$$\mathbf{K}_{k+1} = \mathbf{P}^f_{k+1}\mathbf{H}^T_{k+1}\left(\mathbf{H}_{k+1}\mathbf{P}^f_{k+1}\mathbf{H}^T_{k+1} + \mathbf{R}\right)^{-1} \tag{5.8}$$

$$\mathbf{P}^a_{k+1} = \left(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1}\right)\mathbf{P}^f_k \tag{5.9}$$

where $\mathbf{M}$ denotes the model operator, $\mathbf{P}$ the model error covariance matrix, $\mathbf{R}$ the observation error covariance, $\mathbf{x}$ the state vector, $\mathbf{Q}$ the error covariance matrix of added noise to the model, $\mathbf{y}$ the observed values and $\mathbf{H}$ the interpolation operator between model state and observations. The superscript f denotes the forecasted value and a denotes assimilated value of the variable.

The classical KF approach gives the BLUE (best linear unbiased estimator) under the assumption that both the model and observation operator are linear and the model and observation error are independent white noise with known covariance. In addition, if model and measurement errors are Gaussian, the KF is the maximum a posteriori or maximum likelihood estimator. However, these assumptions are only valid for a limited set of applications. The Extended Kalman filter (EKF) is the first order extension of the KF for non-linear models. The only difference between KF and EKF is the usage of a tangent linear approximation of the non-linear model operator $M(x(t_k))$ according to

$$\mathbf{M}_{:,j} = \frac{\partial M_k\left(\mathbf{x}_k\right)}{\partial (x_k)_j}, \tag{5.10}$$

where $(x_k)_j$ denotes the j-th element of the vector $\mathbf{x_k}$. The applicability of the filter however depends on the non-linearity of the model.

The tangent linear model needs to be available in order to apply the EKF as well as the model error covariance matrix $\mathbf{P}$. For large dynamical models it is infeasible to compute $\mathbf{M}$ and store the matrices $\mathbf{M}$ and $\mathbf{P}$.

Approximations of the EKF have been proposed for applications where the dimension of the state is too large to store the covariance matrix, or that involve a model that is too complicated for formulation of a tangent linear model. One of these, the reduced rank square root filter will be discussed in one of the following sections. These EKF approximations avoid the requirement of a tangent linear model, and use a low rank approximation of the model error covariance matrix to bypass the storage problem. The low rank approximation is also employed by the Ensemble Kalman filter, that is discussed in the next section

In general all low rank methods operate on a low dimensional sub-space such that storage and necessary computations become feasible.

5.4.2 Ensemble Kalman filter

The Ensemble Kalman filter [Evensen, 1994] is a popular example of a low-rank filter. The method has been successfully applied in mainly geophysical applications, for example in oceanography [Evensen and van Leeuwen, 1996] and meteorology [Houtekamer and Mitchell, 1998].

The ensemble Kalman filter (EnKF) is based on a representation of the probability density of the state estimate by a finite number N of randomly generated system states $\mathbf{X} = [\xi_1, \xi_2, ..., \xi_N]$.

The forecast of the covariance (Equation 5.6) is replaced by a propagation in time of each of the ensemble members by the model. This propagation is for most practical applications computationally dominant. As a result the computational effort required for the EnKF is approximately N model simulations. The standard deviation of the errors in the state estimate are of a statistical nature and converge very slowly with the sample size ($\approx N$).

The update of the state (Equation 5.7) is replaced in EnKF by

$$\mathbf{X}^a_{k+1} = \mathbf{X}^f_{k+1} + \mathbf{K}_{k+1}[\mathbf{Y} - \mathbf{H}_{k+1}\mathbf{X}^f_{k+1}] \tag{5.11}$$

where $\mathbf{Y} = [\mathbf{y} + \nu_1, \mathbf{y} + \nu_2, ..., \mathbf{y} + \nu_N]$ denotes a matrix with realization of the observed values with ν_i a realization of the observation noise. The observation noise must be added in order to prevent underestimation of model error covariance. It should be noted that for many atmospheric data assimilation problems the analysis step is also a very time consuming part of the algorithm in case of assimilation of large amounts of remote sensing observations from satellites.

The major advantage of the EnKF is that the result will always converge to the 'true' solution of the filter problem for growing ensemble size, even in case of a strongly non-linear model. For large ensembles, the sampling

errors involved with use of random numbers will vanish. For small scale problems where computation time is no issue, the EnKF will always provide the correct result of the filter problem, although it might require a very large ensemble. For practical application to large scale models, the required ensemble size might become too large, in which case the filter problem should be reconsidered and for example the problem should be reduced by taking a simplified model or an uncertainty description with less degree of freedom.

5.4.3 Ensemble square-root filters

There is a fundamental problem associated with the use of EnKF. Small ensembles have only a few degrees of freedom available to represent errors, and suffer from sampling errors in the observations that will further degrade the forecast error covariance representation. The addition of noise ν in the update of the ensemble (Eq. 5.11) is omitted in the ensemble square-root filter (EnSRF) [Tippett et al., 2003, Evensen, 2004]. Since no random sample is generated, this extra source of sampling error is eliminated. Therefore, these methods are expected to perform better than the ones with perturbed observations for a certain type of applications. The mean of the ensemble of the EnSRF filter is first updated using the standard Kalman filter analysis equation. The ensemble is then transformed such that the analyzed covariance matches the theoretical solution. The transformation is not unique and some of the suggested transformations like the one by Evensen [2004] are not mean preserving. This can result in a bias in the results for some applications. This problem is identified and alternative transformations are suggested by Sakov and Oke [2008] and Livings et al. [2006]. The EnSRF algorithm that is implemented in COSTA and used in the presented case study uses the transformation as suggested by Sakov and Oke [2008].

The EnSRF method will, like the EnKF method, converge to the 'true' solution for a large ensemble size.

5.4.4 Reduced-rank square-root Kalman

The reduced-rank square-root (RRSQRT) filter algorithm that is proposed by Verlaan and Heemink [1997b] is based on a factorization of the covariance matrix $\mathbf{P}$ of the state estimate according to $\mathbf{P} = \mathbf{L}\mathbf{L}'$, where $\mathbf{L}$ is a matrix containing the leading eigenvectors $\mathbf{l}_i$ (scaled by the square root of the eigenvalues) .

The model error covariance matrix $\mathbf{P}$ is substituted by the product of

the root matrix $\mathbf{L}\mathbf{L}^T$. Equation 5.6 is replaced by

$$\mathbf{L}_k^w = [\mathbf{M}_k \mathbf{L}_k^a, \mathbf{Q}_k^r] \tag{5.12}$$

where $\mathbf{Q}_k^r$ is a root matrix of $\mathbf{Q}_k$ such that $\mathbf{Q}_k = \mathbf{Q}_k^r \left(\mathbf{Q}_k^r\right)^T$ and the superscript w is used to indicates that the matrix $\mathbf{L}^w$ contains more columns than $\mathbf{L}$.

To avoid an ever-increasing number of columns in the $\mathbf{L}$-matrix $\mathbf{L}_k^w$, a new square-root matrix $\mathbf{L}_{k+1}^f$ will be calculated from $\mathbf{L}_k^w$, with a fixed number of columns. This narrower $\mathbf{L}$-matrix reduces the rank of the approximated covariance matrix to (at most) the chosen number of columns. Hence the name 'Reduced Rank Square Root' filter.

The new square-root matrix $\mathbf{L}_{k+1}^f$ is chosen to minimize the error-measure

$$\Delta\mathbf{P} := \|\mathbf{P}_{k+1} - \mathbf{L}_{k+1}^f \left(\mathbf{L}_{k+1}^f\right)^T\|_{sF}. \tag{5.13}$$

The norm used, $\|\mathbf{A}\|_{sF} := \sqrt{\sum_{i,j}(s_i a_{i,j})^2}$, is called the scaled Frobenius norm. The performance of the method depends very strong on a proper selection of the, possibly time depending elements s_i of the scaling vector $\mathbf{s}$. The scaling is model dependent; for LOTOS-EUROS it is described in Segers et al. [2000]. Determining a proper scaling vector for a model is not necessary for EnKF and EnSRF, therefore it requires more work to use the RRSQRT-filter for a model.

The RRSQRT is deterministic in the sense that it does not depend on random numbers as the EnKF. Results are therefore always reproducible. A drawback is that the approximations on which the RRSQRT filter is based are only valid for weakly-nonlinear models. For this type of models, the RRSQRT might be a cheap alternative for the EnKF, which requires a minimum number of modes to vanish the impact of using random numbers even if the filter problem is linear and of limited rank. For strong non-linear models, the EnKF remains the best option.

5.4.5 COFFEE filter

COFFEE (Complementary Orthogonal subspace Filter For Efficient Ensembles) is a hybrid filter, which combines RRSQRT filter and EnKF [Heemink et al., 2001, Hanea et al., 2007]. One problem of the RRSQRT algorithm is that repeated projection on the leading eigenvalues leads to a systematic bias in forecast errors. Because of the truncation, the covariance matrix is always underestimated, which may result in filter divergence [Hanea et al.,

2004]. The truncated part of the covariance matrix does not contribute to the improvements of the state estimate. The COFFEE filter attempts to solve this problem by representing the truncated part of the covariance matrix as random ensembles and to add them to the EnKF part. The RRSQRT part acts as a variance reductor for the ensemble filter, thus reducing the statistical errors of the Monte Carlo approach. Moreover, by embedding the reduced-rank filter in an EnKF the covariance is not underestimated, eliminating the filter divergence problems of the reduced-rank approach. Like the RRSQRT filter it is necessary to provide a proper scaling vector for the model state.

5.5 Comparison of filter performance

5.5.1 The LOTOS-EUROS model air quality model

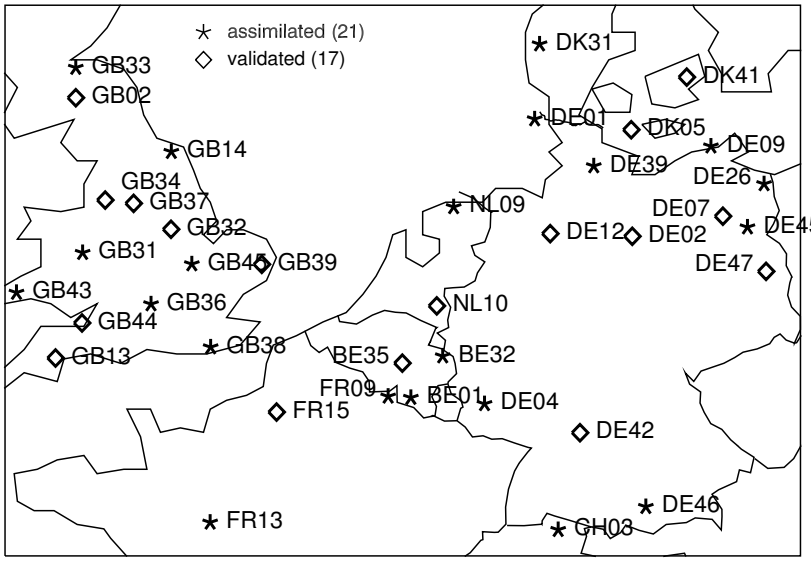

Figure 5.2: LOTOS-EUROS model domain used during the assimilation experiments, and locations of the assimilated and validated measurement sites.

The LOTOS-EUROS model [Schaap et al., 2008] is an operational air quality model of intermediate complexity focused on modelling the lower part of the troposphere. The model domain is bounded to Europe or regions within; in this study, the domain is bounded to the north western part of Europe (Figure 5.2). The horizontal resolution is 0.50 deg. longitude by 0.25 deg. latitude, which leads to a horizontal grid of 40 by 40 cells . In the vertical, the model takes into account a surface layer of 25 m, a boundary layer with a height varying during the day, and 2 residual layers with a top at 3.5 km or more in case of a strongly elevated boundary layer. Boundary layer height is obtained as a part of the meteorological input, and determines together with among others wind, temperature, pressure and humidity the dynamics in the model. The trace gases included and chemical reactions between them are based on the CBM-IV mechanism [Whitten et al., 1980].

To compare the performance of the various Kalman filtering techniques, the LOTOS-EUROS model has been configured to simulate ozone concentrations for July 2003.

5.5.2 Observations

For the selected period, hourly surface measurements of ozone have been collected from the EMEP network [Hjellbrekke and Solberg, 2005]. The EMEP network contains sites that are representative for background concentrations, and the measurements could therefore be compared to the simulated concentrations by the LOTOS-EUROS model. Since the model is unable to represent concentrations at higher altitudes, sites above 700 m have been excluded. With this selection criterium, a total number of 38 measurement sites is available in the domain (Figure 5.2). The set of sites was randomly split into a group of 21 locations from of which the observations are assimilated, and another group of 17 locations that are used for validation.

The observation error is assumed to be normally distributed with a standard deviation of 10% of the measured value with a maximum of 2.5 ppm. This includes instrumental errors as well as the representation error between simulated concentrations in a grid box and the point measurements. The value used here (10% of the observed value) is a pragmatic choice given the grid resolution and the type of observations (hourly values from background sites).

5.5.3 Stochastic model

The simulations made by the model are uncertain for many reasons. These uncertainties include for example modelling errors that are a consequence of the parameterization of the processes in the model, e.g. chemistry, transport, and deposition, which are only an approximation of the actual physical processes. Another important uncertainty is the model input. Input errors are present in for example emissions, meteorological fields and boundary conditions, which are usually temporal and spatial averages that might not describe the actual values at any time and location.

Since the focus of this study is on the performance of different filter implementations, the stochastic model used to describe the model uncertainties is kept rather simple. The only uncertain parameter considered here are the emissions. Although not all errors in the model can be addressed to the emission model, the actual emissions can differ largely in space and time from the modelled emission which is based on yearly totals and average time profiles, and therefore form one of the major uncertainties.

In the stochastic model, the emissions of nitrogen oxides (NO_x) and volatile organic compounds (VOCs) are described by:

$$e_s^j(t_k) = \bar{e}_s^j(t_k)\left[1 + \eta_{\mathbf{s}}^j(t_k)\right], \tag{5.14}$$

where $s = \{NO_x, VOC\}$ describes the emitted component; j identifies a region, $\bar{e}_s^j$ is the deterministic value of the emissions, and $\eta_{\mathbf{s}}^j$ a normal distributed noise parameter. The regions considered are (1) Belgium, The Netherlands, and Luxembourg; (2) Germany and Denmark; (3) France; and (4) Great Britain. Each of the 8 noise parameters have the same standard deviation $\sigma = 0.25$. The noise parameters are not correlated explicitly in time or space.

Some form of spatial correlation is included in terms of the different source regions: within each region, the emission uncertainty is spatially correlated. Temporal correlation could be omitted since observations are available in each source region on hourly basis. The filter state will therefore be adapted quickly (each hour) into a direction of the emissions which are most likely. This choice is suitable to investigate in what direction the model is adjusted. For quantitative estimates of the true emissions an improved stochastic model should be used.

5.5.4 Assimilation results

The COSTA assimilation environment has been used to test the performance of the various filter implementations. In each experiment, the hourly

ozone concentrations from the 21 analysis sites are assimilated with LOTOS-EUROS simulations during a single month (July 2003).

The following filter implementations, as described in section 5.4 have been applied:

- EnKF with either 5, 10, 15, 20, 30, 40 or 50 ensemble members. To test the impact of the use of random numbers in the EnKF, the experiments were performed twice with different random seeds in the number generator. In addition, a run with 250 ensemble members was performed to serve as the 'true' solution of the filter.
- EnSRF with either 5, 10, 15, 20, 30, 40, or 50 ensemble members.
- RRSQRT with reduction to either 10, 20, 30, 40 or 50 modes
- COFFEE with either 7, 13, 20, 27 or 33 modes in the RRSQRT part and either 3, 7, 10, 13 or 17 ensemble members in the EnKF part.

In addition, a single run with the deterministic model is available for comparison. The total number of ensemble members and/or RRSQRT modes correspond to the minimum number of model propagations that have to be computed each timestep of the filter in order to propagate the covariance approximation forward in time. These model timesteps dominate the computation time of the simulations we have performed. Therefore we will compare the filter implementations that have the same total number of modes and members. This assumption is only valid for a moderate number of RRSQRT modes (≤ 50) like we have used in the experiment. The rank reduction of the RRSQRT-filter can take a considerable amount of computational time when a large number of modes is used. For simplicity, the term 'mode' will be used to denote ensemble members as well.

For a first impression of the performance of the different filter techniques, Figure 5.3 shows the timeseries of ozone concentrations in one of the assimilation sites (BE32) during the first 10 days of the simulation period. The figure shows the hourly ozone measurements, the simulation by the deterministic model, and the analyzed mean for 4 different filter types. Each of the applied filters used a total of 50 modes. It is clearly visible that most filters provide more or less the same results: either all are close to the observations or none. From this short time series of a single station no quantitative conclusions about the performance of the individual filters can be drawn, but the general behavior was found to be representative for other locations and periods too.

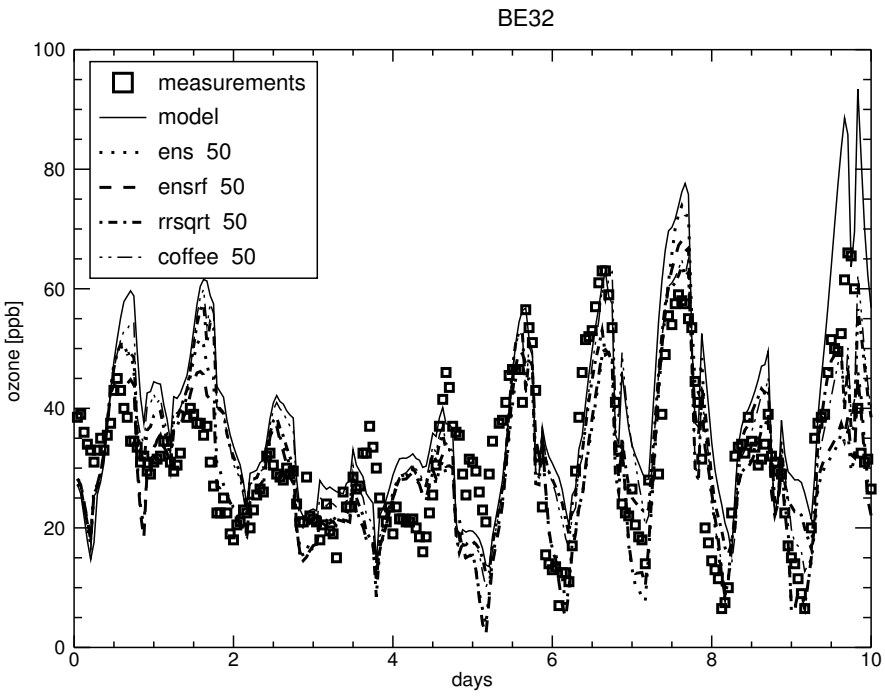

Figure 5.3: Example of ozone time series at EMEP site BE32 (Eupen, Belgium) of measured values, model simulation, and analyzed filter mean.

The most important feature in ozone time series is the height of the afternoon ozone maximum, since this is incorporated in keynumbers describing air quality. For the time series shown in this figure, the model captures the general pattern visible in the measured maxima of higher values during days 6-8 and 10 and lower values at days 3-5 and 9. The concentrations of the afternoon maximum are overestimated on days 1-2 and 8 and 10. Assimilation of the ozone measurements is able to reduce these overestimations. There are clearly differences between the filter implementations however. For the time series shown, the ensemble type of filters (EnKF and EnSRF) provide the best results during the first two days and for the ozone minima during the night. RRSQRT and COFFEE provide better results for the high values during days 6-10, but are unable to decrease the error during the first days. The difference in performance of the various filters could be related to the different treatment of nonlinearities. This will be discussed at the end of this section.

For an overview of the average decrement in the error after assimilation, Figure 5.4 shows the root-mean-square (rms) errors between analyzed mean and observations with and without assimilation. The rms errors are computed for each individual station over all measurements available during the assimilation period. As expected, the rms errors decrease after assimilation. This holds for the assimilated sites as well as for the validation sites, although in general the errors in the validation sites exceed the values computed for the analysis sites. The lowest rms errors after assimilation are obtained for filter experiments using the EnKF and EnSRF implementations, where for some sites the values are decreased by 50% . For the EnKF implementation with 50 ensemble members as shown in this figure, the impact of the use of random numbers seems to be limited. The results obtained differ slightly between the experiments using different random seeds, but do not change the general picture.

To judge the filter performance as a function of the computational costs, Figure 5.5 and Figure 5.6 shows the rms error between model or filter mean averaged over all sites (analyzed and validated), versus the total number of modes. The total number of modes is a minimum for the number of model evaluations required to propagate the covariance structure in time, and therefore a suitable number for the amount of computation time to be spent. The figure shows that, in the chosen experiment, the initial error obtained with the deterministic model could be decreased with 25% . The largest improvements are obtained for the EnKF and EnSRF filters, which provide similar results if at least 10 or more modes are used. Both filters converge quickly; for the chosen setup, 30 modes is enough to obtain similar

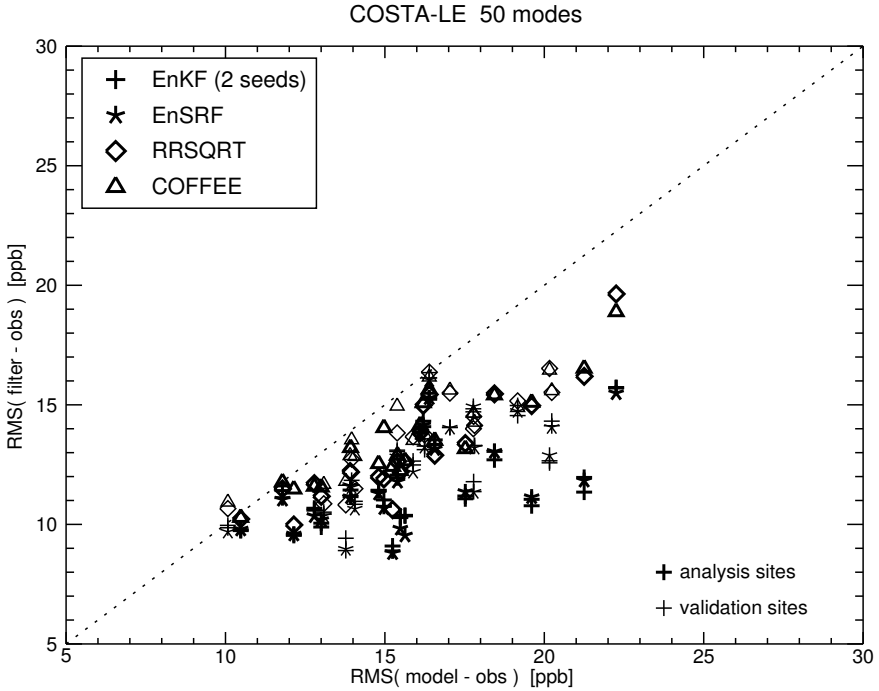

Figure 5.4: Root-mean-square error between filter mean and observations for all measurement sites (used for analysis (thick marks) or validation), versus the error obtained using the deterministic model. Results for filters with 50 modes each.

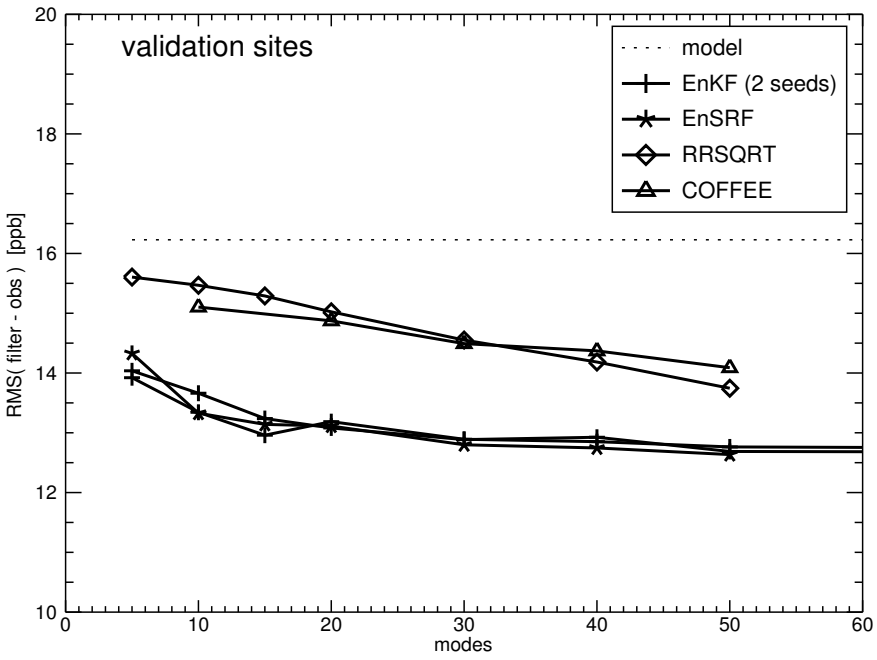

Figure 5.5: Root-mean-square error between measurements and model or filter, for different numbers of modes at the stations that are assimilated. The line for the EnKF implementation is continued at the right side of the figure to the value obtained using 250 ensemble members.

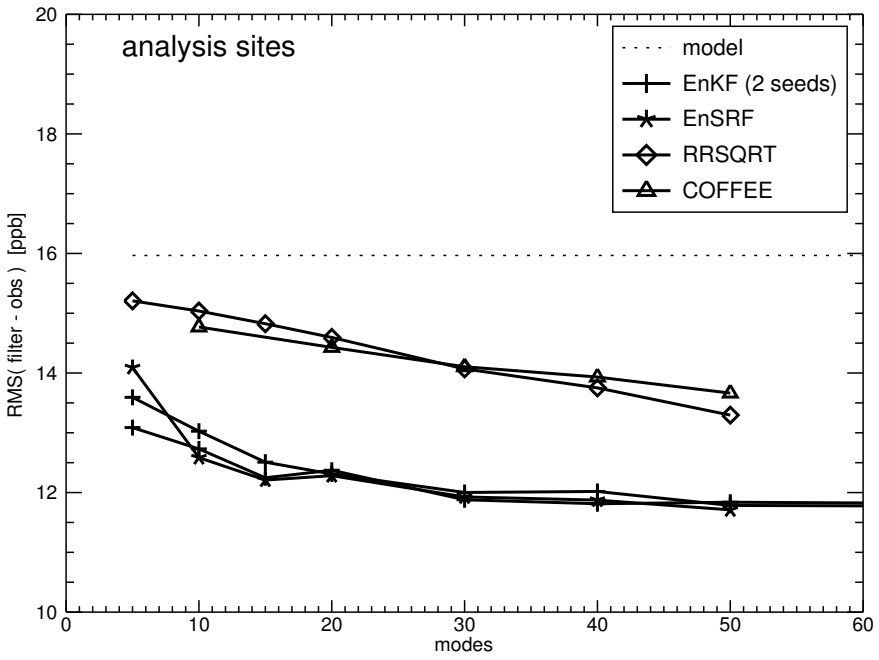

Figure 5.6: Root-mean-square error between measurements and model or filter, for different numbers of modes at the validation stations that are not assimilated. The line for the EnKF implementation is continued at the right side of the figure to the value obtained using 250 ensemble members.

results as what could be obtained using an EnKF implementation with 250 modes, which is large enough to serve as the 'true' converged value. Convergence is much slower for the RRSQRT and COFFEE implementations. Both implementations provide smaller errors with growing numbers of modes, but have not converged to the EnKF-250 results even using 50 modes. These results suggest that the stated filter problem is too non-linear to be treated by filters based on the RRSQRT approach. The propagation from ozone-precursor emissions to ozone concentrations includes non-linear processes in chemistry and removal, which make the model strongly non-linear even on small time scales.

Another difference between the ensemble type and reduced rank type of filters is that the latter include a reduction of the covariance. Newly introduced noise might be discarded in this reduction step. This can result in a different error covariance matrix and a longer spin up period to provide a reasonable state covariance. This needs to be further investigated in long term simulations.

For the validation sites, the errors are slightly larger than for the assimilation sites, but the general conclusions are the same for both sets.

5.6 Conclusions

COSTA is a flexible environment for creating data assimilation systems. COSTA provides basic building blocks as well as a number of data assimilation methods that make it possible to relatively quickly transform an existing model to COSTA and use existing parts of software.

A COSTA model component is created for the operational LOTOS-EUROS model. Using this model component we could perform a range of experiments with the Kalman filtering techniques that are provided by COSTA. We have compared the performance of the EnKF, EnSRF, RRSQRT, and COFFEE filters for various numbers of modes.

The stochastic model used in this study is far from perfect. Experiments will be continued to identify a more extensive stochastic model that captures all differences between model and measurements. Similar to the comparison of the various Kalman filtering techniques we can use the COSTA framework for comparing the results of the experiments using alternative noise models.

An EnKF implementation seems to be a proper choice for these experiments, since it requires hardly any other configuration parameters than the number of ensemble members. If the stochastic model is found to be suitable for a particular application, other filter implementations should be applied

too in order to test whether these filters could reduce the computational costs without significantly changing the results.

The RRSQRT and COFFEE implementations converge slower than the EnKF and EnSRF filter implementations. The performance of these methods depends on the used scaling of the model state in the rank reduction step, improvement of the used scaling might improve the performance of these methods. An other possibility for the slow convergence of the RRSQRT and COFFEE method is that the nonlinearity of the model cannot be captured well by the RRSQRT and COFFEE filter implementations. A detailed investigation of what causes this slow convergence is left for future research.

Software

The COSTA software is freely available under the GNU LGPL license. Documentation an software is available at:

`www.costapse.org`.

Chapter 6

Parallel computing and model coupling [1]

6.1 Abstract

Extending an existing (large scale) dynamical model with respect to data assimilation is an elaborate task. This is particularly so because data assimilation schemes, such as Kalman filtering or calibration methods, have consequences for the way in which the model operator is accessed. This usually requires that the data assimilation method is built on top of the existing model code, instead of under the existing code in a library.

COSTA is a generic toolbox that simplifies the application of data assimilation techniques. It provides interfaces between the assimilation algorithms and the model and observation handling code. It provides well-tested implementations of various data assimilation techniques as well. Concepts of object oriented programming are used to define building blocks for data assimilation systems that can exchanged and reused.

In this work we describe COSTA's parallel computing and model coupling capabilities. Using COSTA, the parallelization of multiple model runs is provided for free, i.e. does not require additional modification of the existing model code. Further COSTA provides templates for the easy incorporation of existing model codes that employ parallel computing themselves. These capabilities are demonstrated by the application of COSTA to three

[1] The content of this chapter is submitted to Scientific Programming; A generic object oriented approach towards parallel computing and combining models in the COSTA framework for data assimilation, by Nils van Velzen, Hai Xiang Lin, Edwin Vollebregt and Erwin Loots.

large scale models in which parallel computing and domain decomposition are used.

6.2 Introduction

Data assimilation concerns the incorporation of measured data into dynamical models [Evensen, 2006]. It is used for instance in numerical weather prediction (NWP). The present state of the atmosphere is known only up to a certain accuracy. By combining observations and short-range forecasts, an improved initial condition for NWP may be found. Data assimilation can be used in order to improve the forecast performance of the model, realize a better reconstruction of the past or to calibrate the model. For this different techniques may be used, such as Kalman filtering [Evensen, 2003, Verlaan and Heemink, 1997b, Whitaker and Hamill, 2002, Heemink et al., 2001], 3DVAR or 4DVAR [Talagrand and Courtier, 1987, Elbern et al., 2000].

The implementation of a data assimilation method for a model can be a significant programming effort. This is particularly so because data assimilation methods change the way in which a model is used. When developing a dynamical model, a straight-forward approach is to initialize the model, perform time steps and output results along the way, and finalize the computation when the end-time is reached. This is not a suitable approach when for instance an Ensemble Kalman filter is used. Within the Ensemble Kalman filter, several different model states are computed simultaneously. The model code must be adapted for this. It must incorporate noise processes that characterize the uncertainty in the model forcings or operator, must be able to solve multiple model states simultaneously or one by one, and be able to perform a so-called analysis step where the predictions are combined with the available measurements.

Up to recently, data assimilation was often implemented for each model code separately. The adaptations of model code were designed for each dynamical model separately, and the data assimilation methods were implemented for each model anew, dedicated to work together with the specific model code at hand. The disadvantage of this approach is the inflexibility. It is not trivial to replace a dedicated data assimilation method by an alternative method. The other way around, it is not possible to apply and test the data assimilation method with alternative models. The same data assimilation algorithms are therefore programmed over and over again and it is difficult to thoroughly test them for programming errors since they can only work in combination with a single dynamic model.

COSTA [van Velzen, 2006, van Velzen and Verlaan, 2007] is a generic toolbox for data assimilation initially developed at Delft University of Technology. It provides a framework wherein dynamic models can be combined with various data assimilation methods and new data assimilation methods can be developed and tested for various models [van Velzen and Segers, 2009]. COSTA provides implementations of data assimilation methods, example models and software building blocks that simplify the programming of data assimilation methods and the adaption of models such that they can be used in COSTA. COSTA is made available under the LGPL license and can therefore be used in combination with both open source and propriatory software.

COSTA uses a different approach than to more traditional packages for scientific computing such as BLAS [Dongarra, 1989], LAPACK [Anderson et al., 1999] or PETSc [Balay et al., 1997, 2004], or even PVM [Geist et al., 1994] or MPI [Snir et al., 1998]. These traditional packages provide libraries of routines that may be called from a user's program. COSTA on the other hand provides a framework and a set of assimilation techniques, and calls routines from the original model code. For this the original model code has to be modified. This approach is inevitable because data assimilation schemes inherently require intervention in the way that the model operator is applied. In its approach COSTA is not unique. Its way of working is comparable to for instance OpenMI [Gijsbers and Moore, 2003] or the Common Component Architecture CCA [CCA, 2009].

COSTA uses concepts of object oriented software development, see e.g. [Szyperzki, 1997, Kruchten, 1998]. COSTA defines various building blocks which are called classes. An object is an instantiation of a class. It comprises its internal (member) variables and methods by which these variables may be accessed or modified. The specification of the methods is called the interface of a class. The idea is that classes can be substituted by alternative implementations. Classes that involve a large amount of functionality are called components. COSTA's object oriented approach is further a powerful method to standardize the way that dynamical models may be used in data assimilation schemes.

In this work we concentrate on large scale dynamical models for which parallel computing is required. We demonstrate that COSTA behaves favorably in this respect. On the one hand, COSTA provides parallel evaluation of different model instances for free. Even if the model has not been parallelized, large portions of a data assimilation run may be performed in parallel. On the other hand, COSTA can easily accommodate model codes that are internally parallelized or use other forms of model coupling as well.

These advantages are achieved by the object oriented approach that is used. They are illustrated by experiments for three large computational packages: the LOTOS-EUROS air quality model [Schaap et al., 2008], the shallow water model WAQUA/TRIWAQ [Stelling, 1983, Vollebregt et al., 2003], and the air quality model Chimere [Bessagnet et al., 2008].

The rest of this paper is organized as follows. In Section 6.3 we will give a basic sketch of the COSTA model component, which defines the operations by which a dynamical model is used. Section 6.4 describes the so-called "tree-vector", by which information in COSTA is moved around. The tree-vector allows for the easy construction of COSTA models and for creating composite models as well, as is explained in Section 6.5. In Section 6.6 we discuss general aspects of parallel computing for dynamic models and data assimilation applications. Section 6.7 describes how "mode-parallelization" is achieved automatically using COSTA's object oriented approach. Section 6.8 concentrates on the use of parallel model code inside COSTA's model component, and shows that this is largely similar to the incorporation of sequential model code. Finally Section 6.9 describes the experiments with the three models mentioned above. The conclusions and the path towards future developments are presented in Section 6.10.

6.3 Introduction of the COSTA model component

The model component in COSTA can be used both for stochastic and deterministic models. Stochastic models contain a model for the uncertainties (differences between model results and reality). Deterministic models are seen as a special case of a stochastic model, in which the uncertainties are ignored (assumed zero).

The 'formal' form of a model in COSTA is

$$\frac{d\mathbf{x}(t)}{dt} = \mathcal{M}\left(\mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \mathbf{w}(t), t\right) \tag{6.1}$$

where $\mathcal{M}$ denotes the model operator, $\mathbf{x}(t)$ the time dependent state-vector, $\mathbf{p}$ the time-independent parameters, $\mathbf{u}(t)$ the time dependent and $\mathbf{w}(t)$ the stochastic forcing. It is very important to notice that the state of the COSTA model component contains more than only the model state-vector $\mathbf{x}$. The state of the model component also include the parameters $\mathbf{p}$, forcings $\mathbf{u}(t)$, noise $\mathbf{w}$ and the time t.

Multiple model instances, each with their own state, can be created. The state of the model cannot be accessed or changed directly. It is only possible to obtain information of the model state or to change the model state using

the methods of the model interface. All methods of the model interface are presented in Appendix 6.A.

The formal model operator $\mathcal{M}$ is in general not present in the code that implements the model. Instead the code numerically propagates the state-vector forward in time, often only for a given fixed stepsize Δt, according to:

$$\begin{aligned} \mathbf{x}(t+\Delta t) &= \int_t^{t+\Delta t} \mathcal{M}\left(\mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \mathbf{w}(t), t\right) dt \\ &= M\left(\Delta t, \mathbf{x}(t), \mathbf{p}, \mathbf{u}(t), \mathbf{w}(t), t\right) \end{aligned} \tag{6.2}$$

The `compute`-method of the model interface propagates the state-vector forward in time for a given value of Δt according to Equation 6.2.

The transformation of an existing model is discussed in detail in [van Velzen and Verlaan, 2007]. The general approach is to identify the variables in the model that represent the state of the model and to isolate the code that implements the model operator M. These two steps are in general the most time consuming, compared to the implementation of the other methods of the model interface. These steps need also be taken for the implementation of a dedicated data assimilation method. Therefore creating a model component takes approximately the same amount of work as preparing model code for a dedicated data assimilation method.

6.4 Description of the COSTA tree-vector

The model state-vector $\mathbf{x}$, parameters $\mathbf{p}$, forcings $\mathbf{u}$ and stochastic forcings $\mathbf{w}$ are usually represented as a single vector and stored in a continuous block of memory in data assimilation algorithms.

This continuous storage format is practical from a computational point of view but it does often not correspond to the representation of the corresponding values in the model. Inside the model, these values can be scattered over various vectors e.g. each holding a field with values of a given quantity and in the case of a parallel model they can be distributed over various processes.

Some kind of administration is necessary in order to make a translation between the values in the vector representation and the representation in the model.

The tree-vector class of COSTA is used for representing the vectors $\mathbf{x}$,$\mathbf{p}$, $\mathbf{u}$ and $\mathbf{w}$. The important additional properties of a tree-vector compared to a normal vector are its structure and the possibility to add meta information. These properties will be discussed in the following subsections.

6.4.1 Structure of the tree-vector

The tree-vector is an extension of a normal vector. The tree-vector has a tree structure with nodes and leaves. The values are stored at the leaves and they are represented as normal vectors. The nodes of a tree-vector are a concatenation of tree-vectors, in this context called sub-tree-vectors. All nodes and leaves have a unique tag for identification. It is possible to directly access a sub-tree-vector by using its tag. Parts of the tree-vector can therefore be accessed without any knowledge of the structure of the overall tree-vector.

We illustrate the structure of the state vector using a state vector of a deterministic shallow water model. On the highest level, the state contains the velocity $\mathbf{v}$, the concentration $\mathbf{c}$, and the water level $\mathbf{s}$. The velocity itself is built up from two vectors $\mathbf{v_x}$ and $\mathbf{v_y}$ each holding a single directional component of the velocity. The resulting state $\mathbf{x}$ is then defined by Equation 6.3.

$$\mathbf{x} = \left\{ \left\{ \mathbf{v_x}^{\text{xvel}}, \mathbf{v_y}^{\text{yvel}} \right\}^{\text{velocity}}, \mathbf{s}^{\text{waterlevel}}, \mathbf{c}^{\text{concentration}} \right\}^{\text{detmodel}} \tag{6.3}$$

The tags are denoted by the super script. Leaving the tags out, we see from Equation 6.3 that the tree-vector represents the concatenated vector of all model variables.

$$\mathbf{x} = \{\{\mathbf{v_x}, \mathbf{v_y}\}, \mathbf{s}, \mathbf{c}\} = \left[\mathbf{u}^T, \mathbf{v}^T, \mathbf{s}^T, \mathbf{c}^T\right]^T \tag{6.4}$$

All operations that can be performed with a normal vector, like addition and computation of norm and inner product can be performed with the tree-vector as well. The vectors at the leaves of the tree-vector are instances of a vector class. Therefore it is possible to use arbitrary vector implementations for the vectors that are used in combination with the tree-vector.

The operations with a tree-vector involve visiting the nodes and leaves of the tree-vector in depth-first order and performing the operations on the leaves. The involved computational overhead is neglectable when the number of leaves is small compared to the total dimension of the tree-vector.

6.4.2 Meta information

The tree-vector can optionally contain useful additional information on the values. At all levels of the tree-vector (thus in nodes and in leaves) it is possible to add meta information. The meta information describes the physical interpretation of the values in the sub-tree-vector. The meta information

includes among others the quantity, unit and spatial information. Meta information of a leaf gives a description of the values of a single vector while meta information at nodes describes the physical interpretation of all the sub-tree-vectors of the node. This is necessary to describe e.g. vector fields where the different spatial components of a vector value are distributed over separate sub-tree-vectors, like the velocity in the example tree-vector of Section 6.4.1.

Automatic interpolation between values of two tree-vectors can be performed in the `axpy` operation (Equation 6.5) of two tree-vectors when the meta information is provided for both vectors.

$$\mathbf{y} := \alpha\mathbf{x} + \mathbf{y} \tag{6.5}$$

The interpolation is especially useful when various models are combined into a larger model like a noise model and a deterministic dynamic model as will be explained in section 6.5.

The meta information of tree-vectors is used as well when tree-vectors are written to NETCDF files for postprocessing and further analysis. The arrays on the NETCDF files are reshaped and gridding information is automatically added if meta information is available. In this way it is possible to directly view the data in the tree-vector with generic NETCDF viewers and post processing software.

6.5 Combining COSTA model components

The models that are used in a data assimilation framework are sometimes a combination of various models. This can be a combination of various dynamic models like a runoff model and a river model in a flood forecast system or a combination of a deterministic dynamic model with a noise model, together forming a stochastic model.

The properties of the COSTA model component and the tree-vector class make it possible to combine models into larger composite models using a generic tool, called the model combiner. The model combiner can be used to combine arbitrary COSTA models. The model combiner is itself a COSTA model and it is configured using an XML-configuration file. This is illustrated in Figure 6.1.

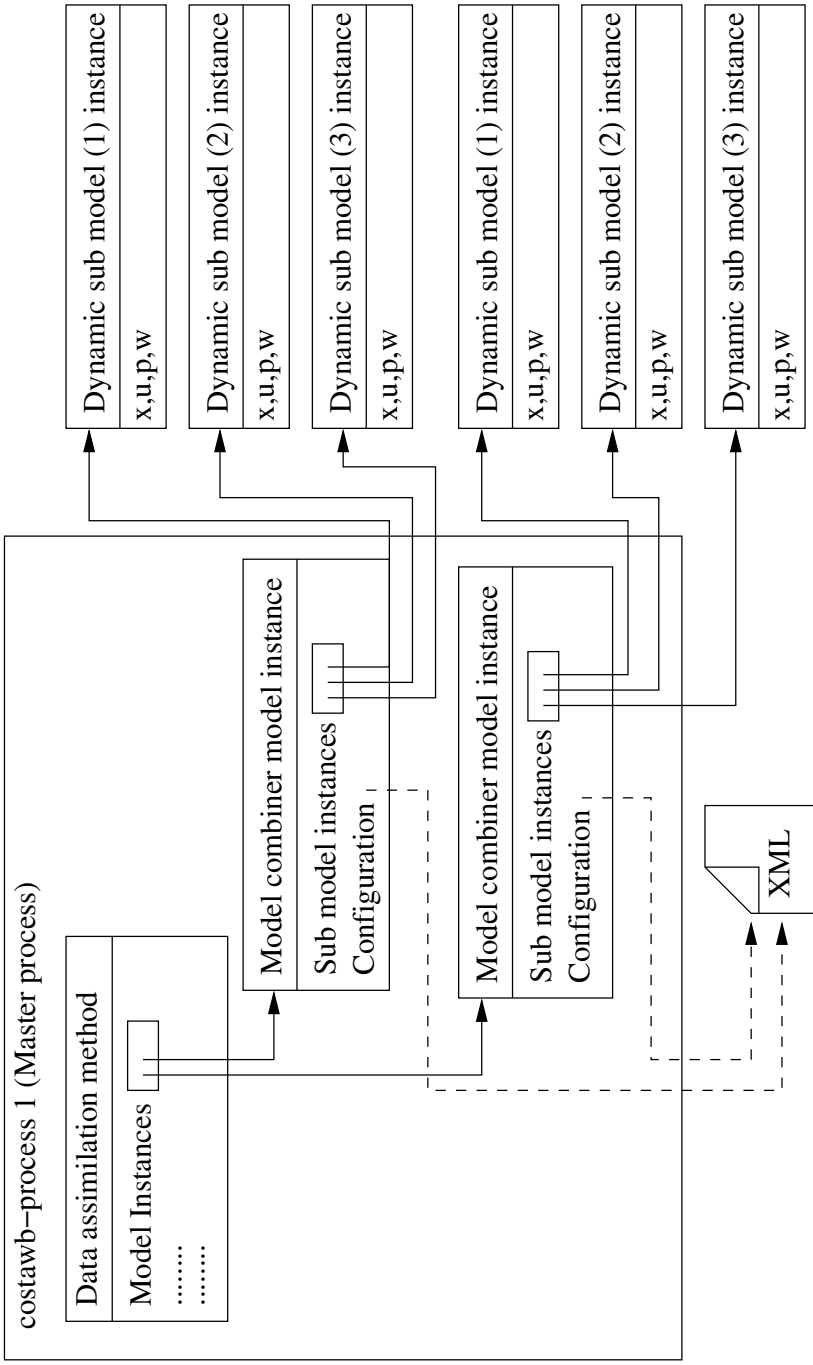

Figure 6.1: Example of a combined model that is created with the COSTA model combiner. The model combiner is a COSTA model in itself. In this example it combines three dynamical COSTA models. The model combiner is configured by an XML-file. This file describes which sub-models are combined and the coupling algorithm between the sub-models.

The model combiner can only be used to create an explicit coupling between various models. This means that there should not be a recursive relation between the propagated state-vectors of the sub-models. A combined model is constructed using the model combiner from m models (Equation 6.2) with a selected fixed overall timestep Δt. For these m model timestep operators we use the notation:

$$\mathbf{x}_{i,k+1} = M_i\left(\mathbf{x}_{i,k}, \mathbf{u}_{i,k}, \mathbf{p}_i, t_k\right), \tag{6.6}$$

where subscript i denotes the index of the sub-model and k the time instance. A timestep of the combined model consists of sequentially propagating the state-vector of all sub-models ($i = 1, .., m$) one at a time. Before the state-vector of a sub-model is propagated it is possible to change the state-vector, parameters and forcings of the sub-model using values from the state-vectors of the previously propagated sub-models. The propagation of the state-vector of the i-th sub model is given by Equation 6.7.

$$\begin{aligned}\mathbf{x}_{i,k+1} = M_i \Bigg(& \mathbf{x}_{i,k} + \mathbf{G}_0^x \mathbf{x}_{.,k} + \sum_{j=1}^{i-1} \mathbf{G}_j^x \mathbf{x}_{j,k+1}, \\ & \mathbf{u}_{i,k} + \mathbf{G}_0^u \mathbf{x}_{.,k} + \sum_{j=1}^{i-1} \mathbf{G}_j^u \mathbf{x}_{j,k+1}, \\ & \mathbf{p}_i + \mathbf{G}_0^p \mathbf{x}_{.,k} + \sum_{j=1}^{i-1} \mathbf{G}_j^p \mathbf{x}_{j,k+1}, t_k \Bigg) \\ & + \mathbf{G}_0^M \sum_{k=1}^{i-1} \mathbf{G}_k^M \mathbf{x}_{i,k+1}\end{aligned} \tag{6.7}$$

The matrices $\mathbf{G}$ denote the selection and interpolation of values from the state-vectors of the other sub-models. The selection of the sub-states, using the tags, is part of the configuration of the model combiner. The interpolation is performed automatically using the meta information of the involved tree-vectors.

The state-vector, forcings and parameters of the combined model (Equation 6.8, 6.9 and 6.10) are represented by a tree-vector with the values of the various models as sub-tree-vectors.

$$\mathbf{x}_k = \left\{x_{s^x(1),k}, x_{s^x(2),k}, ..., x_{s^x(m^x),k}\right\}^{combined} \tag{6.8}$$

$$\mathbf{u}_k = \left\{u_{s^u(1),k}, u_{s^u(2),k}, ..., u_{s^u(m^u),k}\right\}^{combined} \tag{6.9}$$

$$\mathbf{p} = \left\{ p_{s^p(1)}, p_{s^p(2)}, ..., p_{s^p(m^p)} \right\}^{combined} \tag{6.10}$$

Not all models implement the model interface to the parameters and forcings. The combined model can therefore only combine the parameters and forcings of the sub-models that do implement them. The number of sub-models that implement interfacing with the parameters and forcings are denoted by m^p and m^u, where the selection operators s^p and s^u denote these sub-models.

Similar to the forcings and parameters, m^x and s^x denote the number and index of the models that form the combined state. This is necessary because it is possible that the state-vectors of some of the sub-models are only used for changing the input of other models and have no meaning in the combined model. This is the case for models that can be written in the form of Equation 6.11:

$$\mathbf{x}_{i,k+1} = M_i \left(\mathbf{u}_{i,k}, \mathbf{p}, t_k \right) \tag{6.11}$$

Examples of these kind of models are an AR(0)-noise model and a model that performs some (complex) interpolation that cannot be handled by the automatic tree-vector interpolation.

The combined model that is created with the model combiner is a COSTA model. An instance of a combined model holds the instances of the sub-models that are combined, the timestep algorithm, and the administration on the structure of the combined state-vector, forcings and parameters (m^x, m^u, m^p, s^x, s^u and s^p). The values of the combined state-vector, forcings and parameters are not directly administrated in the combined model but are part of the state of the instances of the sub-models.

6.6 Parallel computing in data assimilation applications

Large simulation models can require a huge amount of memory as well as computational time for performing a simulation. The memory requirements per computer and the total simulation time can be reduced by using parallel computing.

Data assimilation algorithms require even more computational time and memory than normal simulations. The overall computational time of a simulation with data assimilation is often a factor 10 to 100 more than a normal simulation. Parallel computing is therefore often required to reduce the computational time and memory requirements to reasonable amounts.

There are various paradigms and programming models to incorporate parallel computing in a simulation model and data assimilation application. Multithreading is a paradigm for parallel computing where within a single executable multiple threads are started that execute parts of the code in parallel. These threads use a shared memory. This form of parallel computing can e.g. be implemented using OpenMP [Chapman et al., 2007].

Multiprocessing is another paradigm for parallel computing. The computations and data are split up over different processes. The processes cannot directly access each other's data. Information is passed between the processes by sending messages to each other. Multiprocessing programs can e.g. be developed using MPI [Snir et al., 1998] or PVM [Geist et al., 1994].

In parallel computing, a computation is first divided into a number of (parallel) tasks. For instance, one of the commonly used approach is domain decomposition in which a spatial domain is sub-divided into a number of subdomains. The computation of the problem on the entire domain is then treated as a number of interacting subproblems each defined by a subdomain. Two common programming models for multiprocessing are the master-worker and worker-worker (process farm) programming models. In a master-worker approach there is one special process called the master. The master process manages the overall coordination and hands out tasks to the worker processes. Typically, after finishing a task a worker process communicates its results to the master process and gets a new task (if there is any). All the communication go through the master process. In the worker-worker approach, the tasks to be computed are either put in a pool (dynamic scheduling) or pre-assigned to the workers (static scheduling, e.g., parallelization based on domain decomposition), all worker processes then work together in order to perform the tasks assigned to them. Workers processes communicate with each other directly in a worker-worker programming model. All forms of parallel computing that have been mentioned in this section can be used to parallize the models that are used in combination with COSTA. In Section 6.8 we explain how models that use parallel computing themselve can be used in COSTA.

COSTA automatically parallelizes some of the computations in data assimilation methods as well. For this the master worker approach is used, where all model computations are handled by the worker processes. In Section 6.7 we will discuss how parts of the algorithms are parallelized in COSTA.

6.7 Automatic parallelization of model timesteps

Data assimilation and model calibration algorithms often need to perform a large number of propagations of the model state-vector forward in time according to Equation 6.2. These timesteps are often computationally very expensive and they can dominate the computational time of the data-assimilation or model calibration run.

The state-vector propagations in the filter algorithm are often independent from each other. For example the propagation of all ensemble members in an Ensemble Kalman filter [Evensen, 1994], the propagation of the columns of the root covariance matrix in an RRSQRT-filter [Verlaan and Heemink, 1997b, Verlaan, 1998], the propagation of the particles of a particle filter [Weerts and Serafy, 2006] or the determination of a gradient by finite difference in a calibration algorithm.

The model state-vectors that are to be propagated in time are represented in COSTA data assimilation methods by instances of the model component. The object oriented approach does not allow direct access to the model state, therefore it is possible to store the state in the memory of another process.

COSTA uses a master-worker programming model for automatically parallelizing the independent model timesteps of the model instances. A special executable called `costawb` can be used to start up a data assimilation run with one of the available data assimilation methods in COSTA and an arbitrary COSTA model component.

A sequential run is started by starting up a single `costawb` executable. All model instances will be created in the memory of the `costawb` process as illustrated in Figure 6.2. The automatic parallelization is activated by starting up multiple `costawb` executables. One `costawb` process will act as the master process and will perform all data assimilation computations. The other processes will become worker processes. The model instances that are created by the data assimilation algorithm will be created on the worker processes. For each model instance that is created on a worker process a special model instance of the parallel interface model is created on the master process as illustrated in Figure 6.3.

The parallel interface model handles all communication between the master process and worker processes. In this way all parallel computing is hidden from the data assimilation algorithm. Whenever the master process uses a method of the interface model, the interface model will send a message to the worker process on which the corresponding model instance is created. This message contains the unique model instance ID, the name of the method and

optionally the input arguments. Methods that return arguments are blocking, in which case the interface model instance must wait until the results are received from the worker process. Otherwise the method is non-blocking and the worker process can continue after the initial message has been sent to the worker process.

The `compute` method that propagates the state-vector, changes the internal state of the model instance ($\mathbf{x}$ and t) but it does not return any results. This method is therefore non-blocking and it is not necessary for the master to wait until the propagation is completed. The master process can therefore ask multiple model instances that are created on different worker processes to propagate their state-vector simultaneously.

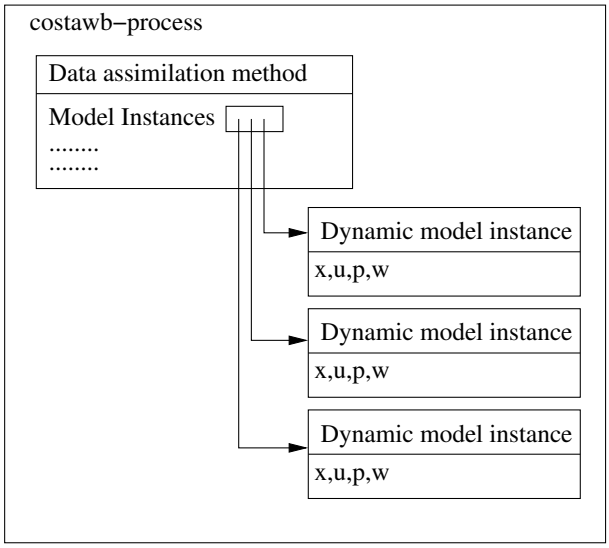

Figure 6.2: Example of sequential run with a single `costawb` executable. The data assimilation method uses a number of model instances of a dynamical model.

The automatic parallel propagation of model state-vectors is available in COSTA for all models that properly implement the COSTA model interface. In this way COSTA provides the possibility to reduce the computational time without the need to make any additional changes to the model or data assimilation code. There are, however, limitations to the parallel performance. The speedup of a parallel algorithm is defined by $s_p = \frac{t_1}{t_p}$ where p is the number of processors, t_1 the execution time of the sequential algorithm and t_p the execution of the parallel algorithm with p processes. The ideal

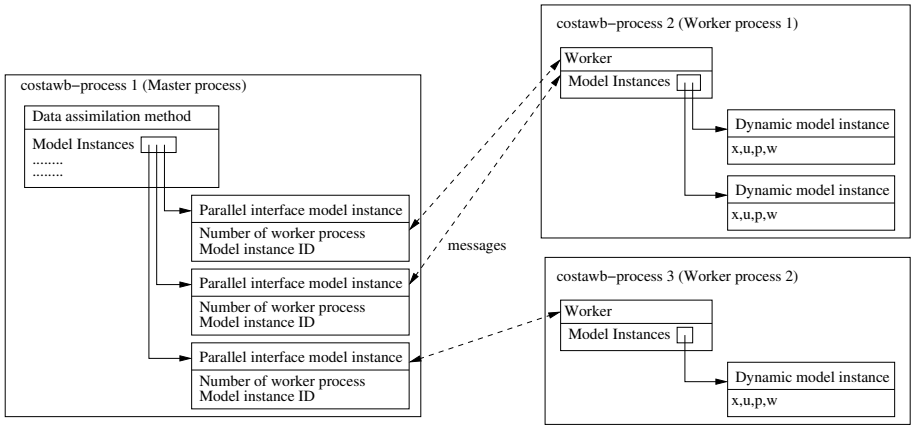

Figure 6.3: Example of a parallel run with one master and two worker processes. The data assimilation method uses a number of instances of a dynamical model. These instances are created on the worker processes and for each model instance a parallel interface model instance is created on the master processes. The parallel interface models handles all communication between the master process and the worker processes.

speedup is obtained when $s_p = p$. The efficiency of the parallel algorithm is given by $e_p = \frac{s_p}{p}$.

The automatic parallelization will only parallelize the independent propagation of model state-vectors. Therefore it will only execute a part of the data assimilation algorithm in parallel. The sequential execution time of the data assimilation algorithm is given by

$$t_1 = t_{\text{seq}} + nt_{\text{model}} \tag{6.12}$$

where t_{seq} denotes the execution time of the sequential part of the algorithm, often the assimilation procedure, t_{model} the execution time of all timesteps of a single model instance and n the number of model instances.

The data assimilation process can be run on the same processor as one of the worker processes since it will not perform computations at the same time as the workers. The optimal parallel execution time of the parallel data assimilation algorithm using the automatic parallelization is therefore

$$t_1 = t_{\text{seq}} + \lceil np^{-1} \rceil t_{\text{model}}. \tag{6.13}$$

As a result of the sequential bottle neck, this part of the parallelization is not ment for massively parallel computing. However, it provides a practicle

form of parallelization that reduces the ammount of computational time for data assimilation and model calibration methods to feasable ammounts for many real life applications.

The actual computation time will in practice be larger than Equation 6.13 due to the additional overhead that is caused by sending messages and data between the master and worker processes.

6.8 Using parallelized model components

Simulation models that use parallel computing for propagating their state-vector can be used in COSTA as well. The used approach for parallel computing that is used in the model is important, since it determines the approach for creating the COSTA model component of the model. A model that is parallelized using threads is in general not different to a sequential model with respect to creating the COSTA model component because this form of parallelism is likely to take place at a low level within the numerical computations and does not influence the program structure at the level where the interface with COSTA is to be implemented. Models that are parallelized by the master-worker programming model can be transformed into a COSTA model component similar to a sequential model as well. Only the master process needs to be adjusted such that it implements the model interface and the worker processes can be kept intact. The only fundamental difference is the start-up of the system where the worker processes of the model need to be started up as well.

In the case of the worker-worker approach, the state-vector of the model is distributed over the various worker processes. This form of parallelism has similarities to a combined model as described in Section 6.5 where each worker process implements a sub-model. The used approach allows models to be couples to parallel models to COSTA that only use a limite number of processors up to models that use massive parallel computing.

The coupling between COSTA and the sub-models of a worker-worker model is at two points different to a combined model that is created with the model combiner. In general it is not possible to perform the timesteps on the sub-models one at a time since it is likely that the computation of a timestep will involve communication between the various worker processes. Another difference is that splitting up the model state-vector over the various worker processes does not mean that the parameters and forcings are split-up as well. The model that is represented by a single sub-domain has therefore

the form of Equation 6.14.

$$\mathbf{x}_{i,k+1} = M_i\left(\mathbf{x}_{i,k}, \mathbf{u}_k, \mathbf{p}, t_k\right). \tag{6.14}$$

The overall model state-vector is given by:

$$\mathbf{x}_k = \left\{\mathbf{x}_{1,k}, \mathbf{x}_{2,k}, ..., \mathbf{x}_{m,k}\right\}^{par\_combined}. \tag{6.15}$$

The parallel interface model is not only used in COSTA for the automatic parallelization. The parallel interface model makes it possible to use models in COSTA that are parallelized according to the worker-worker programming model. A `costawb` worker process is started for each of the sub-domains of the parallel model. The creation of a model instance of the parallel model results in the creation of a model instance for each sub-domain on the worker processes and the creation of a parallel interface model instance on the master process as illustrated in Figure 6.4. The methods from the model interface of the parallel interface model instance will optionally split up the input arguments for the various sub-domains and simultaneously send a request to all worker processes to execute the method for the corresponding model instances of the sub-domains. For methods that return a result, the parallel interface model will combine the returned results of the sub-domains into a single result. The tree-structure of the tree-vector are used to identify the parts of the global state-vector and the state-vectors of the sub-domains.

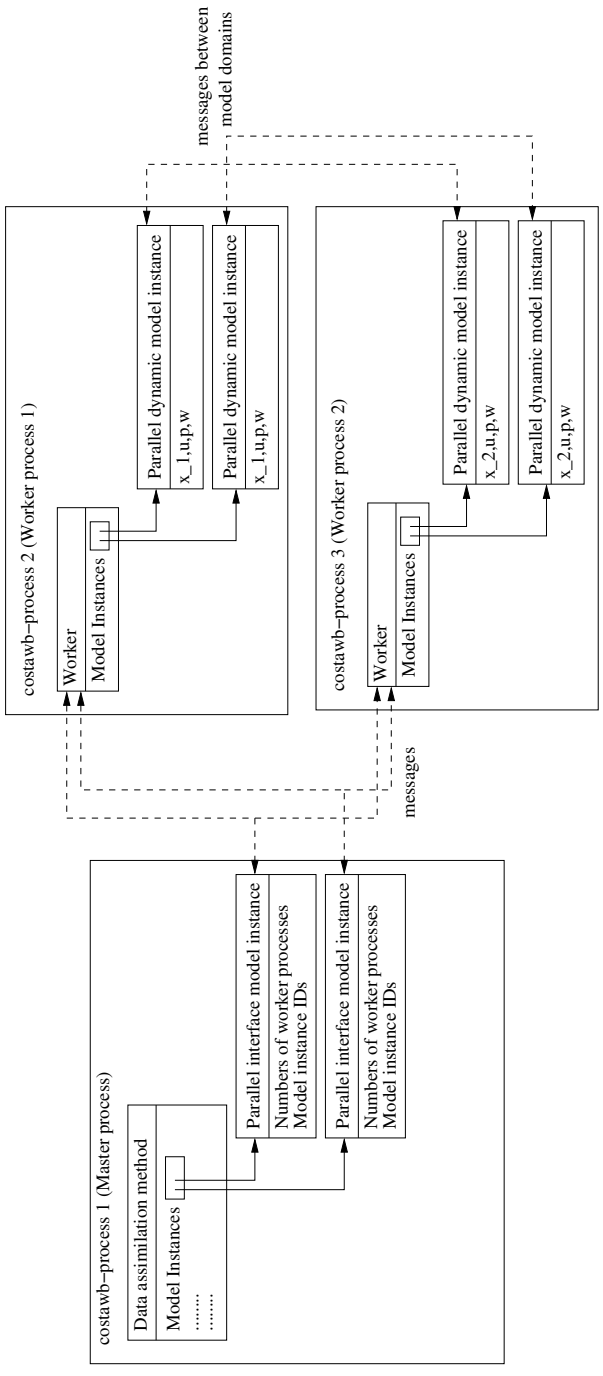

Figure 6.4: Example of a parallel model (2 sub-domains) that is parallelized by the worker-worker programming model. A single instance of the parallel dynamic model consists of an instance of the parallel interface model on the master process and a model instance for each computational sub-domain on the worker processes. The parallel interface model communicates to the sub-domain models and the sub-domain models communicate with each other.

The automatic parallelization of model propagations can be used for parallel models as well. In that case a multiple of the number of sub-domains of COSTA worker processes must be started up.

6.9 Experiments

6.9.1 Automatic mode-parallelization for the LOTOS-EUROS model

We have tested the automatic parallelization capabilities of COSTA with the LOTOS-EUROS model. The LOTOS-EUROS model [Schaap et al., 2008] is an operational air quality model of intermediate complexity. The model focuses on the lower 2-3 km of the atmosphere, and is therefore able to predict the pollutant levels that have a direct impact on human health. The model simulates the concentrations of oxidants and organic components that are responsible for elevated ozone levels, also known as summer smog.

For the experiment we have simulated the air quality of the north western part of Europe (Figure 6.5). This model has a grid of 40×40 cells with a grid size of $\frac{1}{2}$ deg. longitude by $\frac{1}{4}$ deg. latitude. In the vertical the model has a surface layer of 25 m, a boundary layer with a height varying during the day, and 2 residual layers with a top at 3.5 km or more in case of a strongly elevated boundary layer. Boundary layer height is obtained as a part of the meteorological input, and these determine together with (among others) wind, temperature, pressure, and humidity the dynamics in the model. The included trace gases and chemical reactions between them are based on the CBM-IV mechanism [Whitten et al., 1980].

The LOTOS-EUROS models has been configured to simulate ozone concentrations for July 2003. From the EMEP network [Hjellbrekke and Solberg, 2005] we have selected 38 measurement sites (Figure 6.5). The set of sites was randomly split into a group of 21 locations that was used for assimilation of the observations, and another group of 17 locations that was used for validation.

For the noise model we have selected the emissions as being uncertain. Although not all errors in the model can be addressed to the emission model, the actual emissions can differ largely in space and time from the modeled emission which is based on yearly totals and average time profiles, and therefore form one of the major uncertainties.

In the original set-up in [van Velzen and Segers, 2009] we have simulated a period of a month. Because the overall computational time is linear dependent from the simulation period we have shortened the simulation period

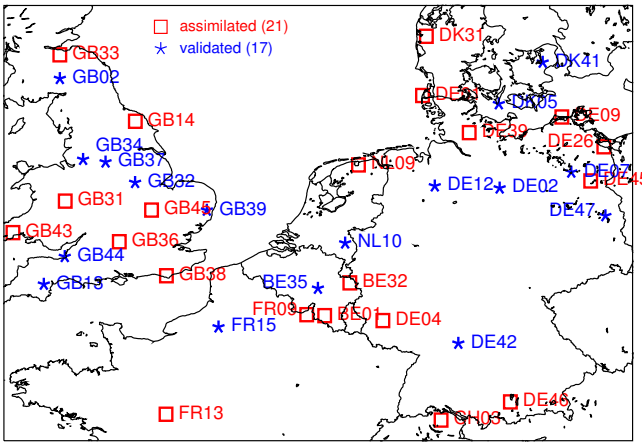

Figure 6.5: LOTOS-EUROS model domain used during the assimilation experiments, and locations of the assimilated and validated measurement sites.

to a single day in this experiment.

The Ensemble Kalman and Ensemble Square Root filters performed best for the given model setup. These Kalman filter techniques converged with approximately 30 members [van Velzen and Segers, 2009]. Therefore we have selected to use the COSTA Ensemble Kalman filter with 32 members for our parallel experiment.

We have run the simulation with varying number of worker processes on a Linux workstation with two Intel Xeon quadcore processors. Each experiment is performed a number of times in order to account for variation in computation time. In figure 6.6 we have plotted the average relative computational time of the simulation for a varying number of worker processes. The dashed line in Figure 6.6 denotes the optimal theoretical computation time according to Equation 6.13. The difference between the optimal and realized computation time is the due to the communication overhead of messages sent between the master and worker processes. This difference is plotted in Figure 6.6 by the dotted line. We can see that the overhead first decreases and then again slightly increases, having a value between 3 and 5% of the sequential computational time. The realized speedup and the theoretical speedup are presented in figure 6.7. The maximum realized speedup

with 8 worker processes is almost 5. The difference between the realized and optimal speedup increases when the number of worker processors increases. This is due to the the fact that the communication overhead as plotted in Figure 6.6, does not decrease under the 3% of the sequential computational time when the number of processes is increased.

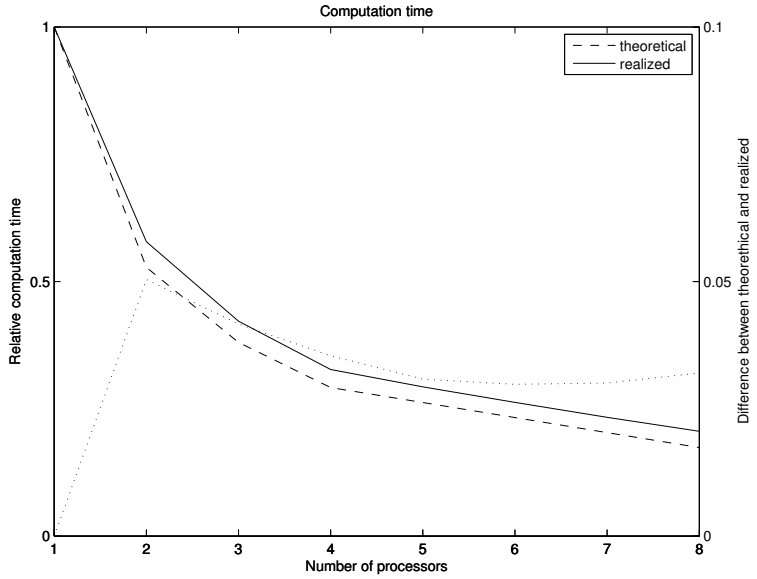

Figure 6.6: Realized relative computation time of a one day LOTOS-EUROS run using the COSTA Ensemble Kalman filter with 32 modes for varying number of worker processes. The dotted line denotes the theoretical optimal computation time.

The speedup of the auto parallelization in COSTA is close to the theoretical speedup. For a further improvement it is necessary to parallelize the sequential part of the data assimilation algorithm as well.

6.9.2 Using COSTA for WAQUA/TRIWAQ with domain decomposition

The WAQUA/TRIWAQ model

WAQUA/TRIWAQ [Stelling, 1983, Vollebregt et al., 2003] is a simulation program that is used for simulating two and three dimensional hydrodynamic and water quality of estuaries, coastal seas and rivers. WAQUA/TRIWAQ is based on the shallow water equations. It is used both for operational and

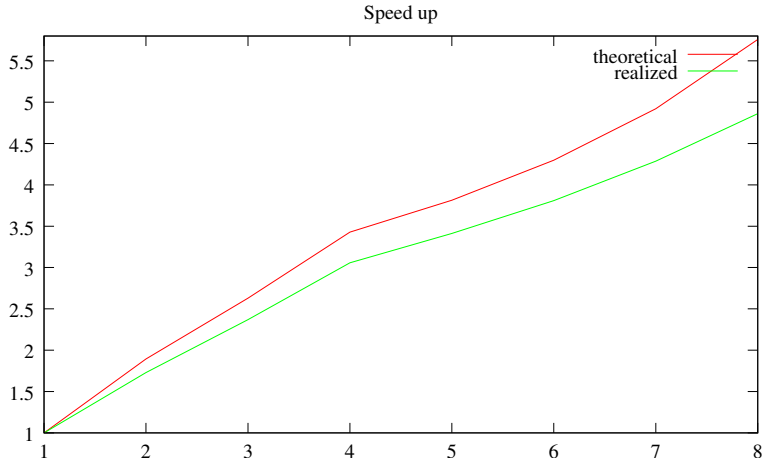

Figure 6.7: Realized and theoretical optimal speedup of a one day LOTOS-EUROS run using the COSTA Ensemble Kalman filter with 32 modes for varying number of worker processes

research purposes by the Dutch National Institute of for Coastal and Marine Management (Rijkswaterstaat/RIKZ).

WAQUA/TRIWAQ is by itself not a model but software implementing a numerical method that makes it possible to do simulations of shallow water. The actual models are defined in the form of an input file that describes all aspects of the model. Various coordinate systems and kinds of grids can be used to model the geographical areas. The modeled areas are bounded by any combination of closed boundaries (land) and open boundaries. Open boundaries force the flows in the model by water levels, velocities, Riemann invariants, discharges or distributed discharges. The system accounts for sources of discharge, such as rivers or outfalls, for tidal flats, for islands and dams, movable barriers and sluices and weirs.

We will however refer to the WAQUA/TRIWAQ model. In this case we mean the WAQUA/TRIWAQ software in combination with an arbitrary input file specifying the modeled geographical area.

The COSTA model component that has been created for WAQUA/TRIWAQ [van Velzen and Verlaan, 2007] is now part of the official distribution of WAQUA/TRIWAQ. The various data assimilation methods that are available in COSTA can therefore be used in combination with WAQUA/TRIWAQ by all users.

Parallel computing

The computation time of a simulation with WAQUA/TRIWAQ can be huge when the simulated period is long or when very detailed models are used. In order to be able to perform these kind of simulations within reasonable time, WAQUA/TRIWAQ has been extended for parallel computing [Roest, 1997, Vollebregt, 1997]. The chosen approach is illustrated in Figure 6.8. A parallel simulation starts with a pre-processing stage where the computational domain is split up into smaller sub-domains. An input file for each sub-domain is automatically generated from the original input file of the global domain by the *partitioner*. After the preprocessing stage, a WAQUA/TRIWAQ process is started for each sub-domain and an additional process called `coexec`. The WAQUA/TRIWAQ processes are exactly the same as those used for a sequential simulation, however when used in parallel they will communicate with the processes simulating neighboring sub-domains in order to solve the global non-linear systems. The `coexec` process only plays a role at the initialization stage and keeps track of the progress of the worker processes and therefore needs a neglectable amount of CPU-time. After the simulation, the output of the various sub-domains is collected into a single file that is exactly the same as the result file of a sequential simulation.

Domain decomposition

There is an important limitation to the grid that can be specified in the WAQUA/TRIWAQ input file. The number of layers must be constant in the whole domain and the regular and curvilinear grids in the vertical cannot be locally refined. As a consequence, when a high horizontal or vertical resolution is necessary at a particular area of the model, the whole geographical area must be modeled at high resolution.

WAQUA/TRIWAQ is extended with domain decomposition both in the horizontal as vertical direction in order to overcome this limitation, increasing the flexibility in grid definition and reducing the computational time. The geographical area of interest in not described by a single grid but by a number of grids that together describe the whole domain.

A domain decomposition simulation is similar to a parallel simulation as explained in Section 6.9.2. The main difference is however that the preprocessing stage does not start with a single input file but with a number of input files, one for each sub-domain. The partitioning step will cut parts from the sub-domain grids, as specified in the configuration of the partitioner, such that all sub-domains fit together like a puzzle. Domain decomposition

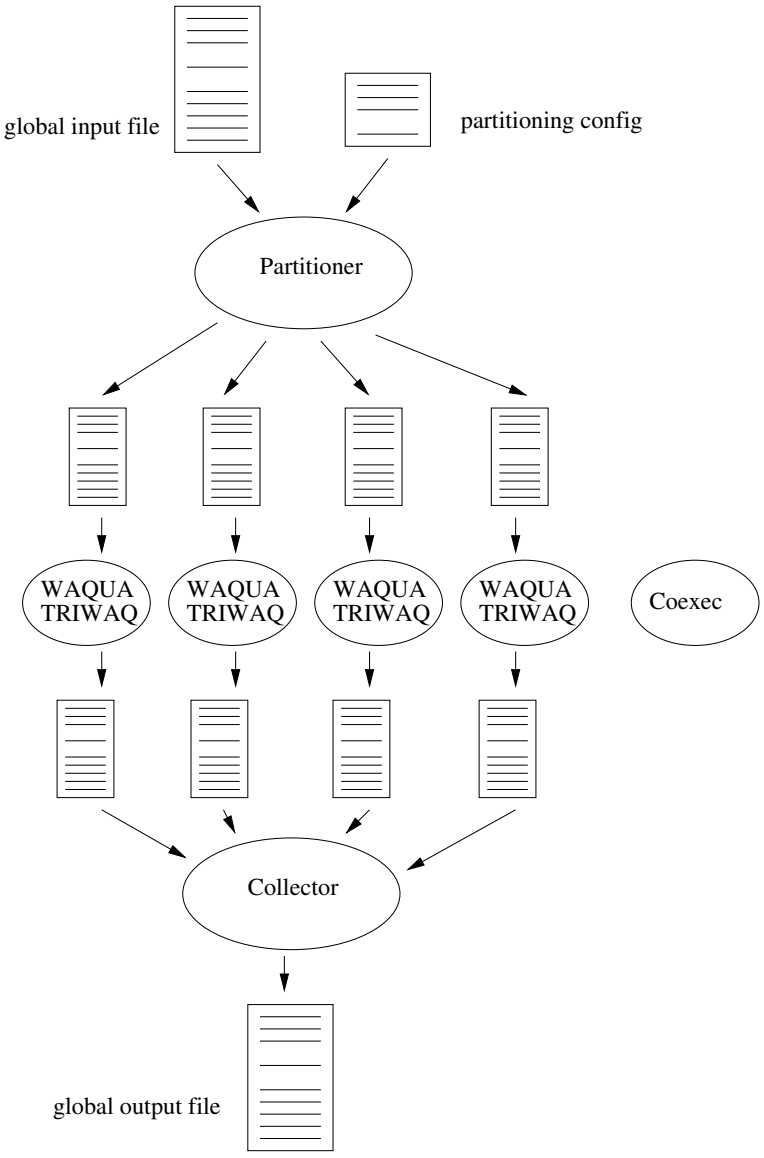

Figure 6.8: Steps of a parallel run with WAQUA/TRIWAQ. The original input file of the global domain is split up into input files for smaller sub-domains in the preprocessing stage. Then a WAQUA/TRIWAQ process is started for each sub-domain. These processes communicate with each other in order to solve the global problem. At the end of the simulation, the output of all sub-domains are collected into a single output file of the global domain.

can be combined with parallel computing. In that case the sub-domains are split up into smaller sub-domains by the preprocessor. The simulation is exactly the same as a parallel simulation, although the communication is more difficult since it involves interpolation of values between the various grids. After the simulation the results are again collected.

Creation of a steady state filter for the CZUNO model

The operational model that is currently used in the Netherlands for day-to-day sea level forecast is the Dutch Continental Shelf Model (DCSM), Figure 6.9. The model has a spherical regular grid of 201 times 173 grid cells with a longitudinal grid size of $\frac{1}{12}$ deg. and latitudinal grid size of $\frac{1}{8}$ deg. This corresponds to approximately 8×8 km grid cells. The model is driven by tidal boundaries and wind. Measurements of the water level stations near the British and Dutch coast are assimilated by use of a steady state Kalman filter. The names and locations of the stations are plotted in Figure 6.9.

The mesh sizes near the Dutch coast are too coarse for accurate predictions. In order to increase the resolution near the Dutch coast a new model called CZUNO is currently developed by the Dutch Rijkswaterstaat. The CZUNO model is a domain decomposition model that combines the meshes of the new DCSM model (version 6) with a spherical curvilinear grid of 1121 times 1261 grid cells and the finer ZUNO (Zuidelijke Noordzee) model with 1458 times 654 grid cells that models the area near the Dutch coast. The grid of the combined model is plotted in Figure 6.10.

At this time, the CZUNO model is still under development and a lot of tuning and calibration work needs to be performed before the model can be used operational. One of the operational applications will be the day-to-day sea level forecast. In order to show that large and complex domain decomposition models can be used in the COSTA framework we have created a steady state filter for the CZUNO model.

The main sources of uncertainty are the forcings of the CZUNO model. We have added noise on the wind and open boundary forcings modeled by an AR(1) noise model. The stochastic model is given by

$$\{\mathbf{x}, \mathbf{n}_w, \mathbf{n}_o\} = \{M\left(\mathbf{x}, \mathbf{u}(t) + \{\mathbf{n}_w, \mathbf{n}_o\}\right), \alpha_o \mathbf{n}_o, \alpha_w \mathbf{n}_w, \} \tag{6.16}$$

where $\mathbf{x}$ denotes the state-vector of the deterministic WAQUA/TRIWAQ model, $\mathbf{n}_w$ the noise parameters on the wind field and $\mathbf{n}_o$ the noise on the open boundary forces and α_w and α_o the noise time correlation parameters of the AR(1) noise model.

The wind noise is assumed to be spatially correlated and is specified on a regular grid with grid cells of 200×200km. When added to the model, the noise is interpolated onto the finer wind grid that drives the model. Similarly only a limited number of noise parameters are specified on the openings where the noise is interpolated as well before it is added to the model.

In total we have computed four steady state filters. The steady state filters have been computed by performing a run of 12 hours using the RRSQRT-filter [Verlaan and Heemink, 1997b, Verlaan, 1998] with 20 and 40 modes and with the Ensemble Kalman [Evensen, 1994] filter with 20 and 40 members. The waterlevel gain that is computed in these four runs for the station at location North Shields is plotted in Figure 6.11.

There are some remarkable differences between the gains that are computed with RRSQRT Kalman and Ensemble Kalman. The Ensemble Kalman gain that is computed with 20 modes suffers from a large number of spurious correlations that disappear when a higher number of members is used. The gains that have been computed with RRSQRT have a remarkable spurious correlation near Denmark. We have investigated the source of this correlation and it is the result of a problem in the current version of the CZUNO model. A significant difference between the RRSQRT gains and Ensemble gains is the order. The gain that is computed by the RRSQRT filter is an order 10 smaller than the Ensemble gain. This needs to be further investigated in the future. The differences might be caused by the non-linear behavior of the model, that cannot be well captured by the RRSQRT filter that linearizes around the central model at every timestep. Underestimation of the model error covariance can be another source for the smaller values in the gain matrix. The model error covariance matrix is truncated in the rank reduction step of the RRSQRT filter. As a result the filter will always underestimate the model error covariance. Possibly in combination with a scaling strategy, that is not optimal, this can result in an significant underestimation of the error in the waterlevels by the RRSQRT filter.

With this experiment we have shown that COSTA can be used with large complex WAQUA/TRIWAQ models. The tools are now available and they can be used for further calibration and development of the new CZUNO model.

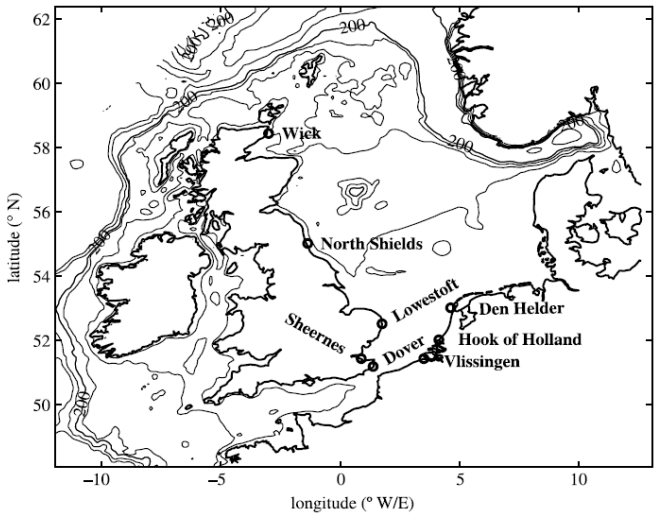

Figure 6.9: The modeled area of the Dutch Continental Shelf Model and the locations of the water level observation stations that are assimilated for the operational forecasts.

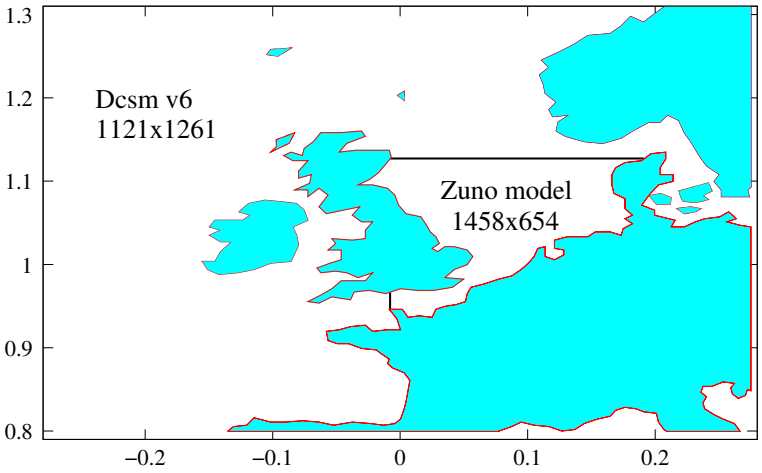

Figure 6.10: The grid of the CZUNO model. The grid is composed from the ZUNO model and a part of the DCSM model

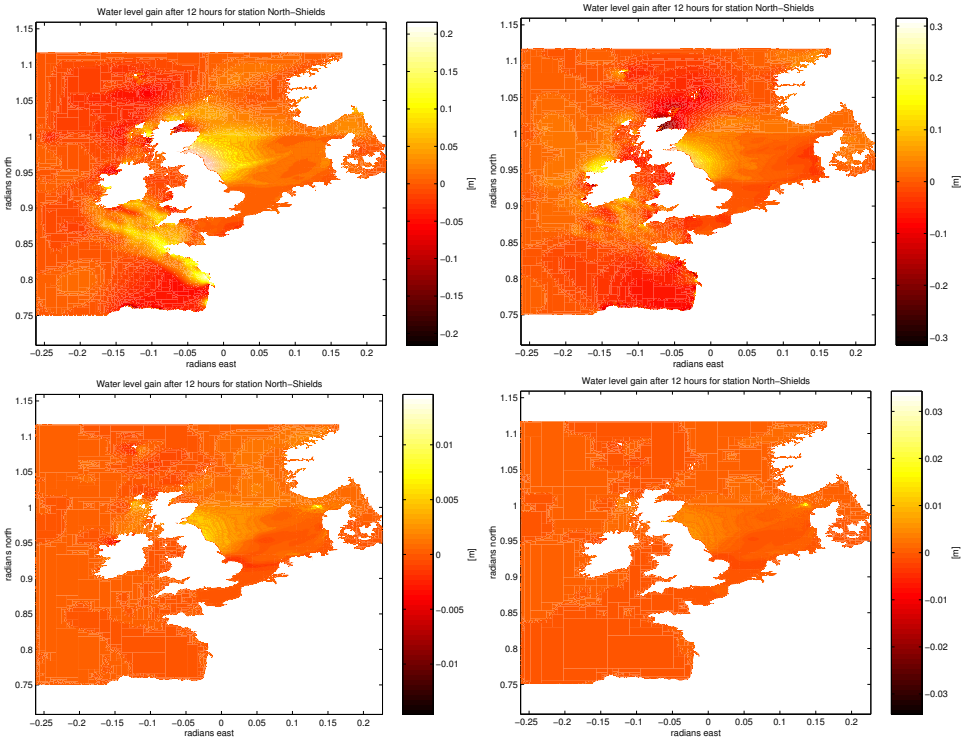

Figure 6.11: Computed water level gain after a 12 hours run for station North Shields for the CZUNO model computed with the COSTA Ensemble-Kalman filter and RRSQRT-Kalman filter. The first row present the gain computed with Ensemble Kalman with 20 and 40 members and the second row the results that are computed with the RRSQRT filter with 20 and 40 modes

6.9.3 Combined mode- and space-parallelization for Chimere

Background

Emissions of ships have a large influence upon the local air quality. Nearly 70 % of the ship emissions occur within 400 kilometers of land [Lowles and Apsimon, 1996]. It was estimated [Lowles and Apsimon, 1996] that the air around the busiest ports bordering the English Channel and the North Sea had SO_2 concentrations exceeding the WHO air quality guidelines. Given the forecasted growth of cargo over seas in the next five years, the importance of shipping emissions can not be overestimated.

In the IMPOSE-project (IMpact Of Shipping Emissions), funded by the Dutch agency for aerospace programmes (NIVR), a monitoring system for shipping-related emissions and air pollution will be developed. An extensive database of satellite observations is currently being created, which can be used for air quality and emission analyses. Data assimilation with COSTA will be used to incorporate the relevant observations in the air quality model Chimere.

Chimere [Bessagnet et al., 2008] is a chemistry-transport model primarily designed to produce daily forecasts of ozone, aerosols and other pollutants. Also, long-term simulations for emission control scenarios can be performed. The model runs over a range of spatial scales from the urban scale (100 Km, resolutions of 1 Km) to the regional scale (several thousands of kilometers with resolutions of 100 Km). The model requires the following input data: meteorological data, boundary conditions, land-use information and emissions. For the former three types of input data, adequate default data is provided. The emissions, to be provided by the user, are the most uncertain and the noise model of the data assimilation therefore focuses upon these.

Parallelization in Chimere

The main spatial domain of Chimere is horizontally divided in rectangular subdomains. The master-worker programming model is used: the master performs all initializations and send to each worker its share of data, corresponding to its subdomain. The master gathers results from the workers.

The Chimere model software has been adjusted to allow integration into COSTA. This involved some reshuffling of the outer layer code while the numerical and chemical kernel of Chimere was left intact.

The state vector of the Chimere model mainly consists of the arrays of species concentrations. For the stochastic model, noise was added to the antropic emissions. These emissions can be divided into ship-related

and non-ship-related emissions to assess the specific impact of the ships emissions.

The Chimere model is computationally rather demanding. While the efficiency of the model (domain) parallelization has yet to be determined more accurately for our schematization, parallelization of the model state propagations seems to be promising given the relatively small state.

The Impose-model is still under development and a lot of tuning will be needed before the operational use of the model. This tuning also involves the use of averaging kernels to transform the vertical columns of NO_2 as computed in the model to the one-column value as acquired by the OMI satellite data.

A first assessment consisted of a small computational domain of 15x10 cells and 8 vertical layers (which is the default value in Chimere). The domain was not subdivided (this amounts to 1 master and 1 worker per model instance). It turned out that the speedup for the automatic parallelized computation was almost linear in the number of processors used. That means that, indeed, the assimilation time (which involves getting, putting and manipulating the tree-vector) is relatively short compared to the computational work of Chimere itself.

Currently the Impose-model is further being developed. The assessment of the combination of parallel propagation of model states and model-parallel will be made again for the full-blown model.

6.10 Conclusions

In this paper we have presented the capabilities of the COSTA framework for data assimilation with respect to parallel computing and model coupling. For this we introduced the object oriented approach in which existing dynamical models are wrapped into "COSTA model components". We described the COSTA tree-vector object that is used to provide an abstract interface for moving information around, and COSTA's model combiner ideas.

The chosen object oriented approach makes it possible to solve some complex issues in data assimilation and model calibration in a generic way. Hiding the model state and using multiple instances of a model makes it possible to automatically parallelize data assimilation and calibration methods without the need to change the code of the algorithms. The uniform way to address model instances using the COSTA model interface makes it possible to combine models into larger composite models. Finally we have

shown that models that are parallelized can be used in COSTA as well.

We have shown experiments with three large scale, complex models: the air quality models LOTOS-EUROS and Chimere and the shallow water model WAQUA/TRIWAQ. These experiments demonstrate the feasibility of the COSTA approach. The computational time for a data assimilation run may be reduced significantly by using COSTA's automatic parallelization of the model runs. The experiments with WAQUA/TRIWAQ and Chimere further illustrate that the models used may employ parallelization and domain decomposition as well.

At this time the computations in the data assimilation algorithms have not yet been parallelized. Therefore future research is needed to include the automatic parallelization of not only the state-vector propagation but the other computations in the data assimilation methods as well.

6.A Interface of the COSTA model component

In this section we give an overview of all the methods that are part of the interface of the COSTA model component. A COSTA model does not need to implement all methods. There are two situations possible when a method is not implemented by the model programmer:

1. COSTA will provide a generic implementation of the missing method. This implementation will probably not be the most optimal or efficient but at least it is available.

2. The method is not available for the model component. In that case not all data assimilation methods or model calibration methods can be used in combination with the model.

We have split up the methods in a number of groups. The first group consists of methods that must be implemented or are available for all COSTA models. A generic implementation is available for methods that are denoted with an asterisk.

- **Create**: Create a new model instance
- **Free**: Free a model instance
- **GetTimeHorizon**: Get the time horizon of the model.
- **GetCurrentTime**: Get the current time of the model.

- **Compute**: Propagate the model state-vector forward in time to a given end-time.
- **GetObsValues**: Get (interpolate) the internal state of the model to the observations described as specified in the observation description component.
- **AnnounceObsValues***:Announce in advance to the model which observations will be requested
- **GetObsSelect***: Get a query for the stochastic observer in order to filter out the observations that can actually be provided by the model

Stochastic models include the following additional methods:

- **AddNoise**: Add noise during during the given timespan at the Compute
- **GetNoiseCount**: Get the number of noise parameters of the model
- **GetNoiseCovar**: Get the root covariance matrix of the noise parameters of the model

Calibration models change the model parameters $\mathbf{p}$. The following methods are available for changing the model parameters:

- **SetParam**: Set the parameters of the model
- **GetParam**: Get a copy of the parameters of the model
- **AxpyParam***: Add change to model parameters $\Delta\mathbf{p}$ to model state-vector p according to $\mathbf{p} := \alpha\Delta\mathbf{p} + \mathbf{p}$

Data assimilation methods change the model state $\mathbf{x}$. The following methods are available for changing the state:

- **SetState**: Set the state-vector of the model
- **GetState**: Get a copy of the state-vector of the model
- **AxpyState**: Add state-vector $\mathbf{y}$ to model state-vector $\mathbf{x}$ according to $\mathbf{y} := \alpha\mathbf{x} + \mathbf{y}$
- **GetStateScaling***: Get an element-wise scaling for the state-vector of the model

- **Export**: Export the whole internal state of a model
- **Import**: Import the whole internal state of a model

The model forcings are in general not adapted by data assimilation or calibration algorithms. The methods that are provided can be used for creating a stochastic model where noise is added to the forcings of the model. The following methods are available for changing the forcings of the model:

- **GetForcings**: Get a copy of the value of the forcings of the model
- **AxpyForcings**: Add a constant change Δu to the model forcings $u(t)$ according to $u(t) := \alpha \Delta u + u(t)$ for a given timespan $t_s ... t_e$

For models that provide an adjoint implementation the following methods need to be implemented:

- **AdjSetointForc**: Set the adjoint forcings of the model (corresponds to the observations in a forward run).
- **AdjPrepare**: Announce to model that an adjoint run will follow the normal forward run.
- **AdjCompute**: Run the adjoint model for the given timespan.

As already mentioned not all methods need to be implemented in order to use the model in COSTA. A practical approach is to implement those methods that are needed for the preferred (group of) data assimilation methods first. Optionally the interface can be extended to increase the performance or usability of the model.

Chapter 7

Conclusions and future work

The work that is presented in this thesis forms the basis for a new approach towards the development and programming of data assimilation and model calibration software. The main scientific impact is that a generic framework like COSTA allows data assimilationists easily to experiment with different reliable and well tested implementations of data assimilation or model calibration algorithms. These data assimilation and model calibration algorithms can be developed independent from a particular simulation model or application area.

The three main building blocks of any application of data assimilation or model calibration method are the dynamic model, observations and algorithm. In this work we present a software development strategy that allows us to create software in which these three building blocks are strictly separated from each other. The presented approach is based on concepts from object oriented programming where all building blocks are defined as classes.

The use of object oriented programming in parallel and high performance computing applications is still tentative. One of the main reasons for not using object oriented programming is that it is expected to have a negative impact on the performance. This thesis shows that object oriented programming can be used without giving in to the performance of the application. Object oriented programming can even be used to improve the performance since it allows for automatic parallelization of computations and easy connection with highly optimized third party software components. The most important classes will be discussed in the next sections

The model class
The model class specifies the interface to the dynamic model to which the

data-assimilation is applied. The implementation of the model class for an existing model is not difficult and does not take a lot of programming. In fact only the model class interface needs to be implemented. This will not take more effort than preparing an existing model code in order to use it in combination with a custom implementation of a data assimilation or model calibration algorithm.

The interface can be implemented either by wrapping the model or inside the model code. Wrapping the model provides a way to use models without the need to make any changes to the model code. Implementing the interface directly in the model code is usually more efficient but not always possible or desirable.

Using a class representation for the models has a number of advantages. It allows algorithms to be developed and implemented independent from a particular model. Also, it makes it possible to automatically parallelize the computations of model timesteps, therefore significantly improving the computational performance without any additional programming effort or knowledge on parallel computing. Finally, the object oriented approach offers an uniform way for combining models into larger composite models and finally it makes it possible to seamlessly use models that use parallel computing themselves independent from the chosen parallelism methodology.

The interface of the model class contains an at first sight redundant but important method called the axpy method on the model state. This method is of vital importance to be able to handle model specific issues in comparing and updating model states and to reduce copying overhead.

Tree-vectors

We have designed the tree-vector class, a novel approach for representing model state vectors, parameters and forcings offsets. The tree-vectors main purpose is the representation of vectors and performing optimized liner algebra operations on vectors. Additionally, tree-vectors are extremely flexible. Multiple tree-vectors can be combined in self describing composite tree-vectors. This new approach provides an excellent bridge between the preferred single vector presentation of variables in algorithms and the scattering of these variables over multiple vectors in the model, additionally it makes combining variables of a composite or parallel distributed model trivial, whereas it is a difficult task in conventional programming.

Another important aspect of tree-vectors is the possibility to optionally add meta information about the content of the tree-vector. The meta information makes it possible to automatically interpolate between values and

eases the post processing of data.

The Observations

Observations should not to be presented by a single vector per timestep. Instead, a self describing class called the stochastic observer is presented in this work. A stochastic observer not only contains the measurements but information on among others the error statistics, location, units and observation kernels. Because the observations are self describing it becomes possible to combine observations from various sources and to make selections of observations. The meta information of the observations is used as well for interpolating the predicted model values to the observed values.

The COSTA framework

The concepts introduced in this thesis have been implemented and validated in the COSTA data-assimilation framework. The various experiments using a range of models and algorithms have shown the effectiveness of the proposed development strategy. Using COSTA we have been able to demonstrate that the research objectives that are formulated in Section 1.2 can be met.

COSTA has been applied to a range of operational dynamic models including WAQUA/TRIWAQ and LOTOS-EUROS Chimere and Sobek-Re. This illustrates that a single interface definition for the model can be used for a range of models. Similarly we have implemented a range of data assimilation and model calibration methods. We have performed a large number of experiments to show that the presented concepts allow to make arbitrary combinations between these methods and the models.

Algorithms

Using an uniform framework, the implementations of the data-assimilation algorithms can be applied to and tested on various dynamic models. Hence, the code is tested much more thoroughly than dedicated code that is developed for a single dynamic model. As a result these generic implementations will be more robust and will contain less errors. The correctness of the implementations is very important, not only in an operational context but also in the scientific context. The performance of different algorithms can only be investigated and compared meaningfully when the algorithms do not contain significant errors.

The COSTA framework is very useful for the development of new algorithms. It makes it possible to compare the performance (both in terms of computational efficiency and in terms of effectiveness) of new algorithms

with respect to algorithms that are already available in COSTA. The comparison can be made for all the models that are coupled to COSTA, from small academic 'toy' models to large and complex operational models.

New algorithms, once implemented in COSTA, can be directly made available to other researchers and users of COSTA as well. Hence, new developments can quickly be taken up in practice.

From the point of view of the users of data-assimilation methods, the COSTA approach has the advantage that they can easily seek out the most effective method for their model. These capabilities are shown using a case study with the LOTOS-EUROS model. A wide range of experiments with the LOTOS-EUROS model in combination with the EnKF, EnSRF, RRSQRT, and COFFEE filters could be carried out without the need for any programming.

The set of data assimilation methods in COSTA framework is constantly being extended. The current ongoing research with COSTA includes the calibration of the new WAQUA/TRIWAQ DCSM version 6 model. Model reduction using Proper Orthogonal Decomposition is used to approximate the covariance of the model variability [Altaf et al., 2008]. The adjoint of the reduced model is used instead of the adjoint of the Tangent Linear model in the variational method. This shows that variational methods can also be accommodated in COSTA.

Performance

Existing operational data assimilation systems can be replaced by a COSTA based system. It has been proven that an efficient equivalent COSTA based alternative can be realized for the RRSQRT-Kalman filter implementation of the WAQUA/TRIWAQ model. Model specific issues like in drying and flooding of parts of the model can be handled in a generic way as well. We have shown that the performance of the generic COSTA based code is comparable to the original code.

The ability of COSTA to automatically parallelize computations is another consequence of using object oriented concepts in high performance computing. The automatic parallelization of data assimilation applications is investigated and illustrated using the LOTOS-EUROS model. From this experiment we see that speedup of the automatic parallelization was found to be close to the theoretically optimal speedup. It has been shown using the Chimere model and WAQUA/TRIWAQ that large scale parallel models can be used without problems in the COSTA framework similar to the way sequential models are used.

It has been shown that a significant improvement of the computational

time can be accomplished using the automatic parallelization that is already available in COSTA. However to get a good performance on a large number of processors it is necessary as well to parallelize additional computations in the algorithms. The possibilities to extend the automatic parallelization in COSTA is one of the research items that will be carried out in the near future. One of the options is to automatically parallelize the linear algebraic calculations e.g. by using an third party software package like PETSc [Balay et al., 2001, 1997] in combination with COSTA.

Towards the future

Currently the DATools package from Deltares and the COSTA framework are combined into a new open source data assimilation framework called OpenDA. Within OpenDA all high level programming and configuration will use the Java programming language. Java provides an excellent object oriented language for combining the building blocks and configuring the system. The computations themselves will be performed by the native, C and Fortran based, part of this new framework. The native part will be mainly based on the existing COSTA code. The high performance native part guarantees the performance and will implement the support for parallel computing.

Bibliography

M.U. Altaf, A.W. Heemink, and M. Verlaan. Inverse shallow water flow modeling using model reduction. Submitted to International Journal for Multiscale Computational Engineering, 2008.

D.B.O. Anderson and J.B. Moore. Optimal Filtering. Prentice-Hall, 1979.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenny, and D. Sorensen. LAPACK User's Guide, Third Edition. SIAM, 1999.

J. Anderson, N. Collins, D. Dowell, S. Gentile, T. Hoar, H. Liu, and K. Raeder. DART web page, 2009. `http://www.image.ucar.edu/DAReS/DART`.

S. Balay, V. Eijkhout, W.D. Gropp, L.C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, Modern Software Tools in Scientific Computing, pages 163–202. Birkhäuser Press, 1997.

S. Balay, K. Buschelman, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang. PETSc Web page, 2001. `http://www.mcs.anl.gov/petsc`.

S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

B. Bessagnet, L. Menut, G. Aymoz, H. Chepfer, and R. Vautard. Modeling dust emissions and transport within europe: The ukraine march 2007 event. J. Geophys. Res., 113(D152002), 2008. doi:10.1029/2007JD009541.

N. Booij, R.C. Ris, and L.H. Holthuijsen. A third-generation wave model for coastal regions: 1. model description and validation. J. Geophys. Res., 104:7649–7666, 1999.

F. Bouttier and P. Courtier. Data assimilation concepts and methods. In ECMWF website, 1999.

R.H. Byrd, J. Nocedal, and R.B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. Mathematical Programming, 63:129–156, 1994.

CCA. Common component architecture forum, 2009. `http://www.cca-forum.org/`.

B. Chapman, G. Jost, and R. van der Pas. Using OpenMP : portable shared memory parallel programming. MIT Press, 2007.

W.A. Cooper, T.W. Schlatter, T. Gal-Chen, and D.B. Parsons. The analysis of observations, with applications in atmospheric science. In The Advanced Study Program Colloquium Series, National Center for Atmospheriv Research, 1999. `http://www.asp.ucar.edu/colloquium/1992`.

G.P. Cressman. An operational objective analysis system. Mon. Weather. Rev., 87:367–374, 1959.

B. Denby, M. Schaap, A. Segers, P. Builtjes, and J. Horálek. Comparison of two data assimilation methods for assessing PM_{10} exceedances on the european scale. Atmospheric Environment, 42(30):7122–7134, sep 2008. doi:10.1016/j.atmosenv.2008.05.058.

R.van der Merwe and E.A. Wan. ReBEL web page, 2006. `http://choosh.ece.ogi.edu/rebel`.

J. Dongarra. BLAS web page, 1989. `http://www.netlib.org/blas`.

J.-Ph. Drécourt. DAIHM Toolbox Version 1.02. DHI Water & Environment, Agern Alle 11, 2970 Hørsholm, Denmark, 2001. URL \url{`http://projects.dhi.dk/daihm/Files/daihm_manual.pdf`}.

J.-Ph. Drécourt. Data assimilation in hydrological modelling. PhD thesis, Technical University of Denmark, Kgs. Lyngby., 2004.

J.-Ph. Drécourt, J.N. Hartnack, H. Madsen, and J.T. Sørensen. DAIHM web page, 2003. `http://projects.dhi.dk/daihm`.

T.F. Edgar and D.M. Himmelblau. Optimization of chemical processes. McGraw-Hill International editions, 1989. ISBN ISBN 0-07-100415-7.

H. Elbern, H. Schmidt, O. Talagrand, and A. Ebel. 4d-variational data assimilation with an adjoint air quality model for emission analysis. Environmental Modelling & Software, 15:539–548, 2000.

G. Evensen. The ensemble kalman filter: theoretical formulation and practical implementation. Ocean Dynamics, 53:343–367, 2003.

G. Evensen. Sampling strategies and square root analysis schemes for the enkf. Ocean Dynamics, 54:539–560, 2004.

G. Evensen. Data assimilation, The Ensemble Kalman Filter. Springer, 2006.

G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. J. Geophys. Res., 99:10,143–10,162, 1994.

G. Evensen. NERSC web page, 2007. `http://enkf.nersc.no`.

G. Evensen and P. van Leeuwen. Assimilation of geosat altimeter data for the agulhas current using ensemble kalman filter with quasi-geostropic model. Mon. Wea. Rev., 124:85–96, 1996.

G. Evensen and P.J. van Leeuwen. An ensemble kalman smoother for nonlinear dynamics. MWR, 128:1852–1867, 2000.

R. Fletcher and C.M. Reeves. Function minimization by conjugate gradients. Computer Journal, 7:149–154, 1964.

A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, 1994.

R. Giering and Th. Kaminski. TAF web page, 2000. `http://www.fastopt.com`.

P. Gijsbers and R.V. Moore. OMI architecture working document. Technical report, European Union research project HarmonIT, 2003.

P.J.A. Gijsbers. The OpenMI architecture - details. In Proceedings of the 6th International Conference on Hydroinformatics, 2004.

S.P. Han. A globally convergent method for nonlinear programming. Journal of Optimization Theory and Applications, 22(3), 1977.

R.G. Hanea, G.J.M. Velders, and A. Heemink. Data assimilation of ground-level ozone in europe with a kalman filter and chemistry transport model. J. Geophys. Res., 109, 2004. doi:10.1029/2003JD004283.

R.G. Hanea, G.J.M. Velders, A.J. Segers, M. Verlaan, and A.W. Heemink. A hybrid kalman filter algorithm for large-scale atmospheric chemistry data assimilation. Monthly Weather Review, 135:140–151, 2007.

HarmonIT. HarmonIT web page, 2009. `http://www.harmonit.org`.

A.W. Heemink and H. Kloosterhuis. Data assimilation for non-linear tidal models. International Journal for Numerical Methods in Fluids, 11:1097–1112, 1990.

A.W. Heemink, M. Verlaan, and A.J. Segers. Variance reduced ensemble kalman filtering. Mon. Weather Rev., 129:1718–1728, 2001.

C. Hill, C. DeLuca, V. Balaji, M. Suarez, and A. da Silva. Architecture of the earth system modeling framework. Computing in Science and Engineering, 6:18–27, 2004.

A. Hjellbrekke and S. Solberg. Ozone measurements 2003. EMEP/CCC-Report 4/2005, NILU, Norway, august 2005.

P. Houtekamer and H.L. Mitchell. Data assimilation using an ensemble kalman filter technique. Mon. Weather. Rev., 126:796–811, 1998.

S. Hummel, M.R.T. Roest, and H.H. ten Cate. The oms backbone: System software for an open modelling system. volume 1, pages 624 – 629, 2002.

E. Kalnay. Atmospheric Modeling, Data Assimilation, and Predictability. Cambridge University Press, 2002.

P. Kruchten. Modeling component systems with the unified modeling language. In International Workshop on Component-Based Software Engineering, 1998.

Th. Lagarde. Assimilation variationnelle et variabilité spatio-temporelle. PhD thesis, Université Paul Sabatier (UPS), 2000.

H.P. Langtangen. Building programmable problem solving environments for porous media flow. In CMWR XVI -Computational Methods in Water Resources the 2003 ACM symposium on Applied computing, june 2006.

C. Larman. Agile & Iterative Devellopment, A manager's Guide. Agile Software Development Series. Alistair Cockburn and Jim Highsmith, 2004.

LIBXML. The xml c parser and toolkit of gnome, 2009. `http://www.xmlsoft.org`.

G. Lindström, B. Johansson, M. Persson, M. Gardelin, and S. Bergström. Development and test of the distributed hbv-96 hydrological model. Journal of Hydrology, 201:272–288, 1997.

D.M. Livings, S.L. Dance, and N.K. Nichols. Unbiased ensemble square root filters. Technical Report Numerical analysis report 6/2006, The University of Reading, 2006.

I. Lowles and H. Apsimon. The contribution of sulphur dioxide emissions from ships to coastal acidification. International journal of environmental studies, 51(1):21–34, 1996.

Th. Morel, A. Thévenin, A. Piacentini, and O. Thual. PALM web page, 2008. `http://www.cerfacs.fr/globc/PALM\_WEB/`.

E.E.A. Mouthaan, A.W. Heemink, and K.B. Robaczewska. Assimilation of ers-1 altimeter data in a tidal model of the continental shelf. Deutsche Hydrographische Zeitschrift, 46:285–319, 1994.

I. Nassi and B. Shneiderman. Flowchart techniques for structured programming. SIGPLAN Notices, 8:12–26, 1973.

U. Naumann, J. Utke, M. Fagan, N. Tallent, P. Hovland, B. Norris, M. Mills-Strout, C. Wunsch, C. Hill, P. Heimbach, and P. Barton. OpenAD web page, 2009. `http://www.mcs.anl.gov/OpenAD`.

J.A. Nelder and R. Mead. A simplex method for function minimization. Computer Journal, 7:308–313, 1965.

J. Nocedal. Updating quasi-newton matrices with limited storage. Mathematics of Computation, 35:773–782, 1990.

W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Numerical recipes, the art of scientific computing. Cambridge University Press, 1989.

P. Reggiani and J. Schellekens. Invited commentary:modelling of hydro-logic responses: The representative elementary watershed (rew) approach as an alternative blueprint for watershed modelling. Hydrological Processes, 17: 3785–3789, 2004. doi:10.1002/hyp.5167.

M.R.T. Roest. Partitioning for Parallel Finite Difference Computations in Coastal Water Simulation. PhD thesis, Delft University of Technology, 1997.

W. Royce. Managing the development of large software systems. In Proceedings of IEEE WESCON, pages 1–9, august 1970.

P. Sakov and P.R. Oke. Implications of the form of the ensemble transformation in the ensemble square root filters. Mon. Weather. Rev., 136(3): 1042–1053, March 2008.

M. Schaap, H.A.C. Van Der Gon, F.J. Dentener, A.J.H. Visschedijk, M. Van Loon, H.M. ten Brink, J.-P. Putaud, B. Guillaume, C. Liousse, and P.J.H. Builtjes. Anthropogenic black carbon and fine aerosol distribution over europe. J. Geophys. Res., 109, 2004. doi:10.1029/2003JD004330.

M. Schaap, R.M.A. Timmermans, M. Roemer, G.A.C. Boersen, P.J.H. Builtjes, F.J. Sauter, G.J.M. Velders, and J.P. Beck. The LOTOS–EUROS model: description, validation and latest developments. International Journal of Environment and Pollution, 32(2):270–290, 2008. doi:10.1504/IJEP.2008.017106.

A.J. Segers, A.W. Heemink, M. Verlaan, and M. van Loon. A modified rrsqrt-filter for assimilating data in atmospheric chemistry models. Environ. Model. Softw, 15(6-7):663–671, September 2000.

G. El Serafy, H. Gerritsen, S. Hummel, A.H. Weerts, A.E. Mynett, and M. Tanaka. Application of data assimilation in portable operational forecasting systems - the datools assimilation environment. Ocean Dynamics, 57:485–499, 2007.

M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. MPI-the complete reference. Vol. 1. The MPI core. MIT Press, 1998.

P. Spellucci. Numerische Verfahren der nichtlinearen Optimierung. ISNN Lehrbuch Birkhäuser Verlag, 1993.

G.S. Stelling. On the construction of computational methods for shallow water flow problems. PhD thesis, Delft University of Technology, 1983. Rijkswaterstaat communications no. 35.

G.S. Stelling and S.P.A. DuinMeijer. A staggered conservative scheme for every froude number in rapidly varied shallow water flows. Int. J. Numer. Meth. Fluids, 43:1329–1354, 2003.

E. Stephen, N.S. Sivakumaran, and R. Todling. A fixed-lag kalman smoother for retrospective data assimilation. MWR, 122:2838–2867, 1994.

C. Szyperzki. Component Software: Beyond Object-Oriented Programming. Addison-Wesley Longman, 1997.

O. Talagrand and P. Courtier. Variational assimilation of meteorological observations with the adjoint vorticity equation. i: Theory. Quart. J. R. Meteorol. Soc., 113:1311–1328, 1987.

M.K. Tippett, J.L. Anderson, C.H. Bishop, T.M. Hamill, and J.S. Whitaker. Ensemble square root filters. Mon. Weather. Rev., 131:1485–1490, 2003.

M. van Loon, P.J.H. Builtjes, and A.J. Segers. Data assimilation of ozone in the atmospheric transport chemistry model lotos. Environ. Model. Softw, 15(6-7):603–609, September 2000.

N. van Velzen. Costa a problem solving environment for data assimilation. In Proceedings of CMWR XVI -Computational Methods in Water Resources the 2003 ACM symposium on Applied computing, june 2006.

N. van Velzen and A.J. Segers. A problem-solving environment for data assimilation in air quality modelling. Environmental Modelling & Software, 2009. doi: 10.1016/j.envsoft.2009.08.008.

N. van Velzen and M. Verlaan. COSTA a problem solving environment for data assimilation applied for hydrodynamical modelling. Meteorologische Zeitschrift, 16(6):777–793, December 2007. doi: 10.1127/0941-2948/2007/0241.

M. Verlaan. Efficient Kalman Filtering Algorithms for Hydrodynamic Models. PhD thesis, Delft University of Technology, April 1998.

M. Verlaan and A.W. Heemink. Tidal flow forecasting using reduced rank square root filters. Stochastic Hydrology Hydraulics, 11:349–368, 1997a.

M. Verlaan and A.W. Heemink. Tidal flow forecasting using reduced rank square root filters. Stochastic Hydrology Hydraulics, 11:349–368, 1997b.

M. Verlaan, E.E.A. Mouthaan, E.V.L. Kuijper, and M.E. Philippart. Parameter estimation tools for shallow water flow models. Hydroinformatics, 96:341–347, 1996.

E.A.H. Vollebregt. Parallel Software Development Techniques for Shallow Water Models. PhD thesis, Delft University of Technology, 1997.

E.A.H. Vollebregt, M.R.T. Roest, and J.W.M. Lander. Large scale computing at Rijkswaterstaat. Parallel Computing, 29:1–20, 2003.

A.H. Weerts and G.Y.H. El Serafy. Particle filtering and ensemble kalman filtering for state updating with hydrological conceptual rainfall-runoff models. Water Resour. Res., 42, 2006. doi:10.1029/2005WR004093.

J.S. Whitaker and T M. Hamill. Ensemble data assimilation without perturbed observations. Mon. Weather. Rev., 130:1913–1924, 2002.

G. Whitten, H. Hogo, and J. Killus. The Carbon Bond Mechanism for photochemical smog. Environ. Sci. Technol, 14:14690–14700, 1980.

Summary

A Generic Software Framework for Data Assimilation and Model Calibration

Nils van Velzen

Dynamic simulation models are used in many application areas. The accuracy of dynamic models can be increased by using observations in conjunction with a data assimilation or model calibration algorithm. However, implementing such algorithms usually increases the complexity of the model software significantly. This leads to high costs for the maintenance and further extension of the software.

To ease this problem, this thesis proposes a software framework for data assimilation and model calibration that is strictly separated from the dynamic model. The interaction between the model and the framework is established through a well-defined interface. Thus, the model can be maintained and developed further without significant influence from the data assimilation and model calibration functionality.

Reversely, the data assimilation and calibration framework can also be developed independently from the model software. It can be used for a large number of models, thus making the investment to implement data assimilation or model calibration for a model much lower. Also, the reuse will quickly reveal any errors in the framework code.

At first experts in the field of data assimilation and calibration are consulted in order to find out what the desired functionality should be of this software framework. Then algorithms are studied to determine which common building blocks are used by these algorithms. These common building blocks are translated into software building blocks using concepts from object oriented programming. By combining these software building blocks it is easy to create a data assimilation or model calibration program.

Based on the proposed design, a software package called COSTA is created. With COSTA, it is possible to test the proposed approach on complex

operational dynamical models. The proposed approach has been evaluated and tested in three studies on large operational models.

The first study investigates whether it is possible to reconstruct the existing data assimilation functionality for the shallow water model WAQUA/TRIWAQ along the lines proposed in this thesis. The data assimilation functionality was reprogrammed in the COSTA framework and then coupled to the original WAQUA/TRIWAQ code. This study demonstrates that the combination of COSTA and WAQUA/TRIWAQ has the same functionality as the original WAQUA/TRIWAQ-with-data-assimilation, but is much better structured. Also, implementing the data assimilation with COSTA proved to take relatively little effort. Moreover, the additional computational overhead in the new more flexible program is very low.

In the second study, the flexibility of the system was studied. The purpose of this study was to show that a dynamic model that complies with the COSTA standard can be trivially combined with the different data assimilation algorithms. To demonstrate this, a case study is set up wherein the operational LOTOS-EUROS air quality model is combined with four data assimilation methods. In this study, the results of these different methods are compared. Due to the software framework, it was very simple to set up the simulations and compare the results.

Finally, in the third study it is examined how parallel computing fits into the proposed approach. A number of aspects of parallel computing are investigated. First, it is shown using the LOTOS-EUROS model how the data assimilation computations can be automatically parallelized without any effort of the (model) programmer. Then it is explained how models that are already parallel by themselves can easily be used. This is demonstrated by performing experiments with the very large parallel domain decomposition model WAQUA/TRIWAQ called CZUNO and parallel air quality model Chimere.

In conclusion it has been possible to design and implement a software framework that fulfills the predefined requirements. A road map is given on how this framework can be further developed and improved in the future such that it can be used by an even larger group of users.

Samenvatting

Een generiek software raamwerk voor data-assimilatie en model kalibratie
Nils van Velzen

Dynamische simulatie modellen worden in vele toepassingsgebieden gebruikt. De nauwkeurigheid van dynamische modellen kan vergroot worden door gebruik te maken van observaties in combinatie met een data assimilatie of model kalibratie algoritme. Echter het implementeren van deze algoritmes leidt in het algemeen tot een significante verhoging van de complexiteit van de programmatuur. Dit leid tot hogere kosten voor het onderhoud en toekomstige uitbreidingen aan de programmatuur.

Om dit probleem te verkleinen wordt in dit proefschrift een software raamwerk ontworpen voor data assimilatie en model kalibratie, waarbij de algoritmes strikt gescheiden zijn van het dynamische model. Alle interactie tussen het model en het raamwerk vindt plaats via een goed gedefinieerde interface. Het model kan daardoor onderhouden en ontwikkeld worden zonder een noemenswaardige invloed van de data assimilatie en model kalibratie functionaliteit.

Omgekeerd kan het data assimilatie en kalibratie raamwerk onafhankelijk ontwikkeld worden van de model programmatuur. Het kan gebruikt worden voor een groot aantal modellen. De investering die noodzakelijk is om data assimilatie en model kalibratie technieken te implementeren voor een model is daarom veel kleiner. Het hergebruiken van programmatuur zal ook tot gevolg hebben dat fouten in het raamwerk sneller gevonden worden.

Allereerst is er met behulp van experts op het vakgebied gedefinieerd wat de gewenste functionaliteit zou moeten zijn van een degelijk software raamwerk. Vervolgens zijn verschillende algoritmes bestudeerd om te achterhalen welke bouwstenen deze algoritmes gemeenschappelijk hebben. Door gebruik te maken van concepten uit het object geörienteerde programmeren zijn deze bouwstenen vertaald in software bouwstenen waarmee het eenvoudig is om

een data-assimilatie of model kalibratie programma samen te stellen.

Het ontwerp is uitgewerkt in een software pakket genaamd COSTA. Op deze manier was het mogelijk om de voorgestelde aanpak ook daadwerkelijk te testen in combinatie met complexe operationele dynamische modellen. In een drietal studies is de voorgestelde aanpak uitgewerkt en is de bruikbaarheid aangetoond voor grote operationele modellen.

In de eerste studie is het onderzocht of het mogelijk is om met een geringe inspanning de bestaande data assimilatie functionaliteit van het ondiep water model WAQUA/TRIWAQ te reconstrueren volgens de in dit proefschrift voorgestelde concepten. De data assimilatie functionaliteit is geherprogrammeerd in het COSTA raamwerk en daarna gekoppeld met de oorspronkelijke WAQUA/TRIWAQ code. Deze studie toont aan dat de combinatie tussen COSTA en WAQUA/TRIWAQ de zelfde functionaliteit heeft maar dan veel beter gestructureerd. Tevens kon het algoritme met een geringe inspanning in COSTA geprogrammeerd worden. Bovendien is gebleken dat de additionele rekentijd van het nieuwe veel flexibelere programma zeer gering is.

In de tweede studie is de flexibiliteit van het systeem onderzocht. Het doel van deze studie was om aan te tonen dat een dynamisch model dat aangepast is om binnen de generieke omgeving te werken op een triviale wijze gecombineerd kan worden met verschillende data assimilatie algoritmes. Om dit aan te tonen is er een studie uitgevoerd waarbij het operationele luchtkwaliteits model LOTOS-EUROS gecombineerd is met een viertal data-assimilatie methoden waarbij de resultaten van deze verschillende methodes met elkaar vergeleken zijn. Dankzij het software raamwerk was het zeer eenvoudig om de noodzakelijke simulaties uit te voeren en de resultaten met elkaar te vergelijken.

Tot slot is in een derde studie onderzocht hoe parallel rekenen past binnen de voorgestelde aanpak. Een aantal aspecten van parallel rekenen zijn onderzocht. Ten eerste is met behulp van het LOTOS-EUROS model uitgewerkt hoe berekeningen automatisch geparallelliseerd kunnen worden zonder dat hiervoor enige inspanning van de (model) programmeur noodzakelijk is. Vervolgens is uitgewerkt hoe modellen die zelf al parallel zijn ook op een eenvoudige wijze gebruikt kunnen worden. Om dit aan te tonen zijn er experimenten uitgevoerd met het zeer grote parallelle domein decompositie model voor WAQUA/TRIWAQ genaamd CZUNO en het parallelle luchtkwaliteits model Chimere.

Concluderend is het mogelijk gebleken om een software raamwerk te ontwerpen en te implementeren dat aan de vooraf gestelde eisen voldoet. Tevens zijn vervolgstappen aangegeven op welke wijze het software raamwerk in de toekomst nog verder ontwikkeld en verbeterd kan worden zodat

deze voor een nog grotere groep gebruikers inzetbaar is.

Acknowledgments

This thesis is not only the result of my work but many people have made a contribution, large or small, direct or indirect. I'd like to thank all these people and some of them in particularly.

The first persons I'd like to mention are Mark Roest and Edwin Vollebregt, my employers at VORtech bv. They were willing to make a significant financial contribution to this work by allowing me to perform my research partially during working hours. Moreover I'd like to thank Mark for reviewing my often badly written drafts and giving me the necessary feedback on my work and Edwin for asking critical questions that helped me to sharpen my ideas and helping to keep my focus wide.

I'd like to extend a deep gratitude to Martin Verlaan. Not only did he make a significant contribution in constructing the financial foundation of my research but moreover I enjoyed the many fruitful hours of discussion we had. In this context I'd also like to thank Herman Gerritsen, Ghada El Serafy and Stef Hummel. With Stef I had many very intense discussions. We often disagreed at the beginning but I think we almost always agreed at the end.

I'd also like to thank Arnold Heemink and Hai Xiang Lin for supervising this project and helping me to reach the finish line in many ways.

I have enjoyed my days working at the TU-Delft and I'd like to thank all colleagues of the 5-th floor for their contribution to a nice working atmosphere. I'd like to mention in particular Kees Lemmens, Alina Barbu, Umer Althaf and of course my officemates Julius Sumihar, Loeky Haryanto, Gosia Kaleta and Remus Hanea. I'd especially like to thank Julius for our discussions, good music taste and his contributions to COSTA and Remus for sharing his wide knowledge on data assimilation algorithms.

Many PhD students at the TU-Delft have lunch, sorted by their nationality. Since Dutch PhD students are hard to find I was happy to meet Kateřina Staňková, who has also a rare nationality, to have lunch with me and chocolate milk moments. I'd like to thank her for our many talks, drag-

ging me into some crazy sport called climbing and teaching me to read and understand Macha a Šebestovou.

I enjoyed working together with Martijn Schaap en Arjo Segers who helped me performing my experiments with the LOTOS-EUROS model and giving me insight in how data assimilation is used within TNO.

I would like to thank Ágnes Mika, Henk Eskes and Hein Zelle for the nice discussions and cooperation and I'd especially like to thank Hein for helping me making the OMI satellite image that I used for the cover of this thesis.

Many of my colleagues at VORtech have also made a contribution to this research. In particular I'd like to thank Bas van 't Hof for sharing his knowledge, Job Verkaik for his suggestions, Johan Meijdam for helping me with C programming and Erwin Loots, who took over most of the development work on COSTA when I was writing my thesis.

I'm also very grateful to my parents who made it possible for me to go to university such that I could learn for a job that is my hobby as well and I would like to thank my parents-in-law for their support.

At the end of my acknowledgment I'd like to thank the most important people in my life: my dear wife Carina and our children Micha, Tirsa, Joach and Simrit for there constant support, love and allowing me to work for many hours on this research. Spending time that I should have spent with them. I'm really grateful that they were willing to make this sacrifice for me such that I could accomplish this work. Thank you!

Curriculum Vitae

Cornelis (Nils) van Velzen was born on October 2, 1975 in Rijswijk (ZH), the Netherlands.

In 1993 he passed his HAVO exam at the Christelijk lyceum dr. W.A. Visser 't Hoofd in Leiden. The following year he passed his VWO exam by taking the staatsexamen in The Hague. In the same year he started to study mathematics at Leiden University. In 1999 he obtained his MSc degree on developing a solution algorithm for the large nonlinear model of the steam pyrolysis process called SPYRO®.

In the period from 1998 to 2001 he has worked at KTI, later Technip, where he worked on developing and implementing numerical solution and optimization algorithms to be used with the SPYRO® model. Since 2004 he is again, part-time, involved in the developement of various simulation software at Technip.

Since 2001 he works as a numerical mathematician and software engineer at VORtech bv in Delft. Here he has worked on problems from various application areas. At VORtech he became in particular specialized in parallel computing, domain decomposition, process coupling, optimization and data assimilation.

In October 2004 he started with a part-time PhD research project at Delft University of Technology under the supervision of prof. dr. ir. Arnold Heemink and dr. Hai Xiang Lin. His research was on developing a flexible software framework for data assimilation and model calibration for various application areas.