VORtech Computing

Industrial and Scientific Computing

P.O. Box 260
2600 AG Delft
The Netherlands

tel. +31(0)15-285 0125
fax. +31(0)15-285 0126
vortech@vortech.nl

MEMO	EV/M08.058
Date	September 2, 2008
Author(s)	dr.ir. E.A.H. Vollebregt
Subject	Introduction to the Costa-system

Document information

Version	Author	Date	Description	Review
0.1	EV	01-09-2008	First version	
File location:	/v3/E05q_bo_simona/c84233-costa-docu/intro-arch-costa			

Motivation for the current document In memo EV/M08.053 an inventory is made of the existing documentation on Costa. Our conclusion is that a better introduction is wanted:

"Een belangrijke moeilijkheid bij het leren werken met Costa is dat de aanwezige technische memo's en design documents van veel voorkennis uitgaan. Het overzicht over Costa is moeilijk te verkrijgen vanuit de documentatie over de programmatuur. Dit zou via het getting started document moeten worden opgelost. Dat document zou weer moeten worden meegeleverd met de programmatuur, en gemakkelijk te lezen en onderhouden moeten zijn. Verder zou het document veel meer aandacht moeten besteden aan het gedegen introduceren van de Costa-filosofie en -architectuur.

Het huidige getting-started document bevat behoorlijk wat informatie voor nieuwe programmeurs, maar leunt wel sterk op bekend verondersteld terminologie. Het zou goed zijn als er een korte introductie van data-assimilatie wordt toegevoegd, als de problematiek van het uitbreiden van modelprogrammatuur met data-assimilatie wat verder wordt uitgewerkt, en als de gebruikte concepten met betrekking tot componenten en object oriëntatie verder worden toegelicht. Pas daarna kunnen de "basic concepts" goed worden geïntroduceerd, kan een overzicht van het werken met Costa worden geschetst, en kan tenslotte het concrete gebruik worden geïntroduceerd. De installatie van Costa zou via een aparte installatiehandleiding of alleen via het INSTALL-bestand moeten worden gedocumenteerd.

In het getting-started document zouden ook belangrijke onderdelen als de Costa workbench en de modelbuilders moeten worden geïntroduceerd."

In the remainder of this memo we propose a new set-up for the getting started document.

1 Introduction

COSTA is a modular framework for data assimilation, containing methods and tools that can be easily applied to a wide range of applications. It allows reuse of data assimilation software, thus reducing the costs of applying data assimilation methods. At the same time, it allows new developments in the field of data assimilation to quickly spread to all applications that might benefit from it.

This Getting Started document introduces COSTA to a new user. It is not intended to be a reference work. For more information about COSTA, the reader is referred to the COSTA website `www.costapse.org`.

This chapter provides a brief glimpse of what COSTA is all about and gives an overview of the contents of this document.

1.1 Contents of this report

To be filled in.

2 A brief introduction of data assimilation

Data assimilation is about the combination of two sources of information, computational models and observations, in order to exploit both of their strengths and compensate for their weaknesses.

Computational models are available nowadays for a wide range of applications: weather prediction, environmental management, oil exploration, traffic management and so on. They use knowledge of different aspects of reality, e.g. physical laws, empirical relations, human behaviour, etc., in order to construct a sequence of computational steps, by which simulations of different aspects of reality can be made.

The strengths of computational models are the ability to describe/forecast future situations (also to explore what-if scenarios), and a large amount of spatial and temporal detail. For instance weather forecasts are run at ECMWF using a horizontal resolution of about $50\,km$ for the entire earth and a time step of 12 minutes. This is achieved with the tremendous computing power of modern day computers, and with carefully designed numerical algorithms.

However, computations are worthless if the system is not initialized properly. “Garbage in, garbage out”. Further the “state” of a computational model may deviate from reality more and more because of inaccuracies in the model, aspects that are not considered or not modelled well, inappropriate parameter settings and so on. Observations or measurements are generally considered to be more accurate than model results. They always concern the true state of the physical system under consideration. On the other hand, the number of observations is often limited in space and in time.

The idea of data assimilation is to combine model and observations, and exploit as much of the information contained in them.

- off-line versus on-line;
- combine values: weights needed;
- statistical framework, std.error;
- deterministic versus stochastic models;
- noise model;
- data-assimilation on top of model.

3 Convincing users to adopt Costa

3.1 Current practice in data assimilation software

Data assimilation techniques are widely used in various modeling areas like meteorology, oceanography and chemistry. Most implementations of data assimilation methods however are custom implementations specially designed for a particular model. This is probably a consequence of the lack of generic data assimilation software packages and tools. An advantage of these custom implementations is that they are in general very computationally efficient. But the use of custom implementations has a number of significant disadvantages:

- Costs - The development and implementation of these methods is very time consuming and therefore expensive.
- Incompatibility - It is very hard to reuse these data assimilation methods and tools for other models than they were originally developed for.

3.2 How COSTA improves the situation

The COSTA project tries to enhance the reuse of data assimilation software by offering a modular framework for data assimilation, containing methods and tools that can be easily applied for general applications. COSTA is set up in order to be as computationally efficient as possible, without losing its generic properties. The aim is that applications developed with COSTA have a comparable computational performance as custom implementations.

COSTA offers support for both users and developers of data assimilation methods. For users it allows models to be quickly connected to the COSTA framework and hence to all the methods that are available in COSTA. For developers, COSTA offers efficient basic building blocks that save a lot of programming work and at the same time makes the new data assimilation software directly connectable to all COSTA compliant models.

3.3 How COSTA works

COSTA provides a generic framework for data assimilation. It is aimed both at model-programmers that want to use data assimilation methods and at developers of data assimilation methods.

COSTA for model programmers For model programmers, COSTA provides a rich set of data assimilation methods. To use them, you have to make your model COSTA compliant. Once your model can interact with COSTA, all the COSTA methods are at your disposal.

What is COSTA compliant? Making your model COSTA compliant involves implementing a number of routines that are going to be called from the data assimilation methods. The set of routines that you must implement is called the stochastic model interface. There are other interfaces as well, each one defining a specific entity called a component. You will read more about components in the next section.

How COSTA calls your implementation of the interface routines. COSTA connects your implementations of these methods to their standard names. These standard names are used in the implementation of the data assimilation methods. There are provisions for working with black-box models(i.e. models for which you do not have the source code), but this will not be discussed in this document.

Providing your observations. Likewise, your observations must be provided in a COSTA compliant way. For the observations there is also a set of routines, called the stochastic observer interface. Usually, you will not implement the interface but convert your observations to a format that can be handled by the standard COSTA implementation of the stochastic observer component.

Further reading. The next chapter (Connecting your model to COSTA) will give a detailed example of how to connect your model to COSTA.

3.4 COSTA for developers of data assimilation methods

For developers of data assimilation methods, COSTA offers a platform for quickly building data assimilation methods that can be used from a wide range of models. COSTA provides various sets of routines that can be used as building blocks. Such a set is called a component. You will read more about components in the section Components.

Routines to interact with models. First of all, COSTA specifies a set of routines to interact with the model to which the data assimilation must be applied. Each COSTA compliant model implements these routines (otherwise it will not be COSTA compliant). The specification of this set of routines is called the stochastic model interface.

Routines to interact with observations. A second set of routines that is essential for data assimilation methods comprises routines to interact with the observations. These include routines to retrieve values and routines to retrieve meta-information.

Figure 1: *Basic design of the Costa system. Model- and observation components (red/white bricks) are plugged into the core of the Costa environment (yellow bricks), and can then exploit the available data-assimilation methods (grey bricks).*

Basic building blocks. Finally, there are various sets of basic building blocks (e.g. for handling vectors, matrices and time). These interact seamlessly with the other COSTA components to let you construct data assimilation methods with a minimum of coding.

Further reading. The chapter called Building COSTA data assimilation methods) will give a detailed example of how to build a COSTA data assimilation method.

4 Basic Concepts of Costa

{*Pre: the reader knows the basic concepts of data assimilation and of the goals of Costa. He now wants to learn more of Costa.*}

4.1 The overall philosophy

Adding data assimilation techniques to existing model software may be a challenging task because the techniques themselves may be complex, especially concerning the description and time evolution of the model noise, and because data assimilation techniques require precise control over the physical/mathematical model that is implemented, for instance repeating a time step under slightly different circumstances. An important step is therefore to identify the precise model equations that are used, and separating these from other computations such as preparation of input values or presentation/storage of model results.

This process of adding data assimilation to an existing model is supported by Costa through a strong separation of concerns. The basic design philosophy is illustrated in Figure 1. The key elements in this picture are:

- a deterministic or stochastic model (red/white bricks);
- a collection of observations (red/white bricks);
- several data assimilation procedures (the grey bricks);
- the core of the Costa-system that connects the different building blocks (the yellow bricks).

This scheme is implemented using object orientation techniques (although non-object oriented programming languages are used). For this the model- and observation-*components* are defined. Components are similar to classes in object-oriented languages. They consist of data and methods (routines) to manipulate that data. There can be multiple *instances* of a component, each with its own set of data. The set of routines to manipulate the data is called the *interface* of the component.

Costa defines the interface of the Costa model component. This consists of the list of methods/routines that a model must provide, such as perform a time-step, deliver its state, or accept a modified state. This interface of the model component is the shape of the empty space in the yellow bricks in Figure 1.

Usually, existing model code does not yet provide the required routines, and certainly not in the prescribed form. Therefore some additional code has to be written to convert between the existing code and the Costa model components interface. This is illustrated in Figure 1 using red and white bricks: the white bricks stand for the original model code, whereas the red 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. This is visualized using a second empty space in the yellow bricks in Figure 1. For using Costa for your model you must fill in this part, by wrapping the existing code for manipulation of your observations and providing it in the required form.

The data assimilation algorithms in Figure 1 cannot yet be seen as instances of another component. For now they are merely procedures that implement different data assimilation techniques with the elements provided by Costa (models, observations, state vectors etc.) as the building blocks. There is however a convergence to a fixed form, a more or less standardised argument list for data assimilation algorithms, such that eventually these may be turned into a well-defined component.

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, and so on, which is often not true for computational models that are not implemented with data assimilation in mind.

The basic design does have a huge advantage over an ad-hoc approach for adding data assimilation to an existing program. It is that the different aspects of a data assimilation algorithm are clearly separated. The algorithmics of the precise Kalman filtering algorithm used are separated from the noise 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 components: adjusting the noise model, adding observations, testing another data assimilation algorithm and so on. This is the major benefit of using the Costa approach.

4.2 Contents of the Costa toolbox

The yellow part in Figure 1 provides the infrastructure needed for connecting generic model- and observation components to generic data assimilation methods. We go into the contents of this part in order to gain a better insight in the structure and working of the Costa system.

Possible intermezzo on use of object orientation in Costa: comparison of a class String in an object oriented language with the CTA_String object.

A basic Costa component is the Costa string component. It is a simple container for character strings, which is primarily introduced to shield the difficulties of sharing text strings between Fortran and C. The operations provided for Costa strings are to create a new instance, set its value, concatenate strings, retrieve its value, and cleanup a string when it is not needed anymore.

Whereas the current Costa software and documentation often uses the word "component", Wikipedia ("Component based software engineering", "Class (computer science)") suggests that "class" would be more appropriate in most cases, with exceptions for the model and observer components.

Another basic component is the Costa time component, which provides a uniform way of handling time. The component registers a time span (interval) and optionally contains a time step attribute. It will be extended in a future version with various time scales and representations, time zones, etc.

A third basic building block is the Costa vector component, which contains a dimension (length), the data type of the vector elements (equal for all elements), and the values of the vector elements. An advantage of incorporating a vector component in Costa is that it provides a generic entity on which data assimilation algorithms can be based, without unnecessarily restricting oneself to the actual data type being used. Further one can choose to provide different implementations (subclasses) of the vector component (class), for instance a distributed vector (for parallel computing), sparse vector (if a significant amount of zeros may occur), or using an optimized BLAS version.

An important component in the Costa toolbox is the Costa tree. It is a generic container for grouping and structuring of data. It may be compared to a "struct" in C or derived type in Fortran. One notable difference is that a Costa tree is a dynamic entity: additional items may be added at run-time, with the names of the items provided as string arguments to the creation routines. Therefore a Costa tree may also be compared to a file system; nodes contain Costa trees (like directories), and leafs contain instances of Costa components (like files).

A special kind (sub-class) of Costa tree is the Costa tree-vector. It is a Costa tree that contains only Costa vectors as leaf elements. Of course it provides all the operations that a Costa tree component provides. Further it supports the operations that a Costa vector is able to perform. The Costa tree-vector is important for instance for describing a model state. In data assimilation one must be able to combine different model states much like vectors can be combined. However, representing the model state by a single vector is a severe restriction for many computational models. It is preferable to be able to distinguish different components of the state, and sometimes needed to distinguish elements with different data types. Further the hierarchical nature of the Costa tree-vector supports composition of larger models out of smaller ones as described later on.

Add an example of a Costa tree-vector, show how it is useful in gathering data for instance to describe a model state.

The Costa toolbox further provides a number of other components: Costa matrix, Costa file, Costa function. More information on these can be found in *the reference manual.*

Costa primarily uses the XML-format for input/configuration-files. There are several facilities for quickly reading such input-files. Costa trees (and tree-vectors) may be written to and read from XML-files as well.

Possibly give an overview of programming conventions used in Costa: the use of handles, build-up of component- and subroutine-names, scalars are doubles, passing by value or by reference, consistent use of return-values, error handling, etc.

Describe the Costa workbench, that is, a generic main program that reads from an XML-file the model(s), observations and data assimilation algorithm to be used and then executes the computations required.

4.3 The Costa stochastic model component

Costa uses a fixed form for a model component, in order to provide consistent terminology to both data assimilation method and model implementers. The general form of a Costa model is

$$x(t + \delta t) = M(x(t), p, u(t), w(t)) \tag{1}$$

Here x stands for the model state, t represents time, p is a vector of parameters, u concerns the forcings of the model, w is the noise/uncertainty, and M is the model operator. Simpler forms of models are possible, for instance a deterministic model without noise/uncertainty, a model without parameters or without access to the forcings, and so on. Although this does not turn a model into an invalid Costa model, it may limit the data assimilation techniques that can be applied.

Data assimilation techniques will have to access the model state. This cannot be done directly. A model may use its own representation of the state. The interface of the Costa model component however requires that a model be able to provide a copy of the state in a Costa tree-vector, and that linear operations on a state-vector can be performed.

Linear operations are an important aspect of the interface of the model component because these operations are used frequently in data assimilation algorithms, and because the implementation may be dependent on the actual model that is implemented. For instance a model may require positivity of specific quantities that it computes (e.g. waterdepth in a flow model), such that blindly combining two state-vectors may give inappropriate results. Therefore a more careful combination recipe may be implemented by the model itself.

Maybe present an overview of the methods of a model here, and/or an example of the construction of a Costa model? Probably not.

4.4 The use of model-builders

The idea of Costa is to make it simple to get started, but to provide full flexibility as well. This is implemented by providing default implementations where possible, but to also allow redefinition of the Costa components.

One place where this idea is given concrete form is in the concept of model builders. A model builder is more or less a template for creating new model components. It fills in as many of the routines that must be provided, such that setting up a new model component is greatly simplified. Once one is up and running one can then tune the implementation to ones own ideas.

The SP model builder ("single processor") can be used to quickly create sequential (non-parallel) model components. This model builder defines choices for the storage and administration of the state vector, model parameters, changes to the forcings, and the noise parameters. With these choices made, a large portion of the methods that must be provided by a model component are provided by the model builder already, and only a few model-specific methods have to be filled in.

Further information on the SP model builder may be taken from the course material (06_-costa_sp-mod.pdf).

It may be appropriate to discuss dynamic linking here as well.

The ideas of model builders are useful for combining separate models as well. For instance most existing simulation models are deterministic, i.e. do not have a stochastic uncertainty component.

$$x(t+\delta t) = M^{det}(x(t), p, u(t)) \tag{2}$$

A stochastic model may be created out of a deterministic model by describing the uncertainty. For this a number of simple ways are available such as just adding noise on the state:

$$x(t+\delta t) = M^{stoch}(x(t), p, u(t), w(t)) = M^{det}(x(t), p, u(t)) + w(t), \tag{3}$$

or using for instance the AR(1) noise model:

$$x(t+\delta t) = M(x(t), p, u(t), n(t)) \tag{4}$$
$$n(t+\delta t) = Dn(t) + w(t) \tag{5}$$

In both cases the resulting equations can be cast into the required form for the Costa model component. In the latter case the state of the extended model contains both x and n, and the combined model operator contains both M and D. And note that equation (5) provides a valid Costa model as well.

This shows that existing Costa model components may be combined to define aggregate Costa model components. This is supported by the model builder called "model combiner". It is often used to separate a deterministic model and its accompanying noise model. This results in a clear separation of the two, reducing the complexity and adding flexibility in playing with alternative noise models.

Other model builders that are provided are the "black box model builder", and the "parallel model builder". *Extend this section with possibilities and restrictions of each.*

4.5 The Costa stochastic observer component

In Costa an observation is not just a value, but contains all the information available for use in a data assimilation method. This involves for instance information on the measurement error, which may be described by a probability density function. Other aspects of observations are the type of quantity that is observed, the unit, time, grid location, and so on.

Optionally describe the measurement model that is used, i.e. the format to which all measurements should comply.

The Stochastic Observer is the Costa component that holds an arbitrary number of observations. It may be instantiated multiple times. A stochastic observer may be queried for a selection of the observations that it holds. For instance the observations within a given timespan may be requested, for a selected set of "stations", or that measure a specific quantity. Such a selection may be used to create a new stochastic observer object.

The stochastic observer further takes care of computing predicted values at observation locations. This concerns the observation relation that is used in data assimilation algorithms: the model state is interpolated and/or converted to the observation location and observed quantity. For this the implementation of a stochastic observer must know the structure of a model state vector, the meaning of its components, and the procedures available for spatial and/or temporal interpolation. For this a stochastic observer contains a substantial application dependent part.

Costa provides default implementations of the stochastic observer and observation descriptions. In this default implementation observations are stored in an SQLite3 database. The database contains two tables, "stations" and "data" for time-independent and time-dependent information respectively. This default implementation provides a basis for setting up ones own observation component. It should grow over time with features that are relevant to different computational models.