COSTA
common set of tools for the assimilation of data

MEMO	CTA memo200604
Date	October 6, 2008
Author(s)	Nils van Velzen and Bas van 't Hof
Subject	Combining models and creating stochastic models

Document information

Version	Author	Date	Description	Review
1.0	CvV	2006-12-11	Initial version	BvtH
File location:		<COSTA_DIR>/doc/modelcombiner		

Table of contents

I COSTA components **1**

1 COSTA components and C++ classes **2**

II COSTA ModelCombiner **2**

2 Introduction **2**

3 COSTA-models **3**

3.1 Mathematical description of a COSTA-model 3

3.2 Interface functions of the COSTA model component 5

3.3 Extension of the interface, necessary for the ModelCombiner 6

4 Combining COSTA models **7**

4.1 Combining noise models and a deterministic model 7

4.2 Correctness of the overall propagation equation 8

4.3 Configuration file for ModelCombiner 9

5 Planned work and time estimates **12**

Part I
COSTA components

1 COSTA components and C++ classes

COSTA is problem solving environment intended to facilitate data assimilation for large scale simulation software. It provides a number of building blocks for data assimilation and calibration systems. Combining and building new building blocks should be possible with a minimum of effort.

COSTA provides building blocks in the form of *COSTA components*. COSTA components are very similar to *classes* in C++, which are elaborate *variable types*. Variables of a C++ class or a COSTA component are called *objects*, or sometimes *instantiations* of such a class/component.

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 subroutine/function that can be called with an object as its argument.

COSTA does not prescribe the implementation of the interface and the state. This means that two objects of the same component may store their value in different ways, and that they may perform the same tasks in different ways.

COSTA provides this possibility of multiple implementations by means of *COSTA component classes*. Every object is not only an object of a certain component, but also has a certain component class, which allows COSTA to find the correct implementation of its interface.

Part II
COSTA ModelCombiner

2 Introduction

One of the building blocks that COSTA offers is the *COSTA-model* component. *COSTA-model* objects have an internal state, and a number of *methods* to set, change or get this state. In the COSTA project, a number of such methods has been designed. The implementation is left to the user: existing code may be used to create the methods of the COSTA-model component. COSTA also offers a set of building blocks to help setting up a COSTA-model component. It is not necessary that all methods are available for a

COSTA-model component: a component lacking certain methods simply cannot be used for certain tasks.

The *COSTA ModelCombiner* is a tool which can be used to create a new COSTA-model component. It combines two or more COSTA-model componentes into one COSTA-model component. The combined method, including its methods, of the combined model can be configured using an XML input file.

The COSTA ModelCombiner is a generic tool for the construction of larger COSTA-model componentes, and may be used to couple all kinds of COSTA models. A specific type of combination for which the ModelCombiner is especially intended, is the coupling of a deterministic (simulation) model and a noise model into a stochastic model.

Combining a deterministic model and a noise model is an important buidling block in data assimilation, because existing simulation models are in general deterministic, and many data assimilation methods need a stochastic model. This means that existing models have to be extended into a stochastic model before assimilation techniques can be applied.

This memo gives a description of this generic tool called COSTA ModelCombiner. This description consists of two parts. Section 4.3 describes how the ModelCombiner can be used by a user. Section 4.3 describes the way in which the ModelCombiner carries out all of its tasks. The description of the ModelCombiner requires detailed information about the COSTA model component. This information is given in Sections 3.

3 COSTA-models

3.1 Mathematical description of a COSTA-model

The COSTA ModelCombiner is a tool for the construction of COSTA model components. Its explanation requires a clear understanding of what a COSTA-model is. This section is intended to explain the COSTA-model component in sufficient detail.

COSTA-models are COSTA components and therefore have an 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 'value' of a COSTA-model $s = (x, u, g, G^u, W^u, G^A, W^A, t)$ consists of the following three parts:

- $x = (\phi, p^u, p^A)$: the 'extended state' (extended solution), consisting of
  - ϕ: 'model state' or (model solution)
  - p^u: forcing-noise parameters

  - p^A: operator-noise parameters

- u: the *forcings*
- g: the *parameters* or *schematization* of the model.
- W^u, W^A, G^u, G^A: interpolation and covariance matrices used in noise models.
- t internal 'time' of the model (not very important in this memo).

In the case of a deterministic simulation, there are no noise parameters. The model state is *propagated* in the following way.

$$\phi\left(t_{i+1}\right) \quad = \quad A\left[\phi\left(t_i\right), u\left(t_i\right), g\right]: \tag{1}$$

The new values are calculated from the old values using some (often very complicated) function A, under the influence of the current values for the forcings u and the values of the parameters g, which are time-independent.

In the general case of a stochastic model, uncertainties are included in the propagation equation. The propagation of the 'true' state ϕ^t is assumed to follow the propagation operator A, except for an error δA:

$$\phi^t\left(t_{i+1}\right) \quad = \quad A\left[\phi^t\left(t_i\right), u^t\left(t_i\right), g\right] + \delta A\left(t_i\right). \tag{2}$$

The 'true' state ϕ^t is of only found when the 'true' previous state, forcings and parameters are entered into the propagation operator. These 'true' values are thought to be given by the forecast values (indicated with the superscript f) and an error term:

$$\begin{aligned} \phi^t(t_i) &= \phi^f\left(t_i\right) + \delta\phi\left(t_i\right), \\ u^t\left(t_i\right) &= u^f\left(t_i\right) + \delta u\left(t_i\right), \\ g^t &= g^f + \delta g \end{aligned} \tag{3}$$

The error terms δA and δu can be obtained (through interplation) from the *noise parameter* vectors p^A and p^u with a much smaller dimension:

$$\delta u(t_i) = W^u p^u(t_i) \quad , \quad \delta A(t_i) = W^A p^A(t_i). \tag{4}$$

The noise parameters may be described by AR(1) processes:

$$\begin{aligned} \delta u^f(t_{i+1}) &= \mathrm{diag}(\alpha^u)\delta u(t_i) + \eta^u(t_i), \\ \delta A^f(t_{i+1}) &= \mathrm{diag}(\alpha^A)\delta A(t_i) + \eta^A(t_i), \end{aligned} \tag{5}$$

Where η^A and η^u are normally distributed stochastic variables, with their covariance matrices G^A and G^u given by

$$G^u = E(\eta^u(t_i)\eta^u(t_i)^T) \quad , \quad G^A = E(\eta^A(t_i)\eta^A(t_i)^T). \tag{6}$$

The propagation of the complete system is described in the following equation:

$$x^t(t_{i+1}) = A_x[x^t(t_i), u(t_i), g] + \begin{pmatrix} 0 \\ \eta \end{pmatrix}, \tag{7}$$

where

- The propagation operator A_x is given by

$$A_x[(\phi, p^u, p^A), u, g] = \begin{pmatrix} A[\phi, u + W^u p^u, g] + W^A p^A \\ \mathrm{diag}(\alpha^u) p^u \\ \mathrm{diag}(\alpha^A) p^A \end{pmatrix} \tag{8}$$

- The covariance matrix G of the normally distributed vector η is given by

$$G = \begin{pmatrix} G^u & 0 \\ 0 & G^A \end{pmatrix}. \tag{9}$$

The internal structure of the model, given in equations (5), is not essential for most data assimilation methods. The extended state vector x may also be composed of different parts, and the extended propagation operator A_x may have a different structure. The overall propagation equation (7), however, is crucial, because it is the starting point of most data-assimilation methods.

The interface of the COSTA model component was designed to perform all the necessary manipulations of stochastic models of the kind described in this section. Models with a simpler structure are obtained by leaving certain parts of the model empty.

3.2 Interface functions of the COSTA model component

The previous section discussed the structure of COSTA models. The section concluded by stating that the interface of the COSTA model component contains all the functions necessary to support data assimilation methods.

The COSTA ModelCombiner is a *COSTA component class*. This means that it is one of the possible *implementations* of the COSTA model interface. Since this memo intends to describe the usage of the COSTA ModelBuilder and the way it works, it is important to know all the functions in the interface.

A COSTA model provides the following functions:

- `DefineCLass`, `Create`, `Free`: functions necessary for the construction and descruction of COSTA models.
- `Compute`: carry out the time steps necessary to step through a given time span.

– FOR $i = i_{start}, \cdots, i_{end}$, DO

$$\begin{aligned} \phi &:= A[\phi, u(t_i) + W^u p^u, g + \delta g] + W^A p^A, \\ p^u &:= \mathrm{diag}(\alpha^u) p^u + \eta^u(t_i), \\ p^A &:= \mathrm{diag}(\alpha^A) p^A + \eta^A(t_i). \end{aligned} \tag{10}$$

END

- `AddNoise`: specify the noise which is to be added to the forcings, state and schematization;

  – FOR $i = i_{start}, \cdots, i_{end}$, DO

$$\begin{aligned} \eta^u(t_i) &:= G^u \texttt{randn}(\texttt{size}(p^u)) \\ \eta^A(t_i) &:= G^A \texttt{randn}(\texttt{size}(p^A)) \end{aligned} \tag{11}$$

  END

- `SetState`, `GetState`, `Axpy`: set, return or modify the state values ϕ in the form of a *COSTA state vector* object;
- `SetForc`, `GetForc`, `AxpyForc`: set, return or modify the forcing values u in the form of a *COSTA state vector* object;
- `SetParam`, `GetParam`, `AxpyParam`: set, return or modify the model schemetization g in the form of a *COSTA state vector* object;
- `GetNoiseCount`, `GetNoiseCovar`: return (dimension of) covariance matrix G of the noise model;
- `GetObsValues`: interpolate the model state to the observations.
- `GetObsSelect`: return information which can be used to read only the observations from a *COSTA stochastic observer* object for which predictions can be generated by the model.

Using the existing COSTA models, the ModelCombiner has to provide all the functions of the interface.

3.3 Extension of the interface, necessary for the ModelCombiner

In the current project, the interface of the COSTA model will be extended: when getting, setting or updating the state, forcings or schematization, so-called *meta-information* will be supplied to describe the information given or asked. This will make it possible for the model to interpret and meaningfully process the information given, or to supply the

correct information. It will also make it possible to get, set or change only *part* of the state, forcings or schematization, because only the information described by the meta-information is returned, set or changed. The new interface will make it (much) easier to combine COSTA models, beacuse it makes it possible to pass the information from one model to another.

The meta-information will have to be obtained from the COSTA models. This is similar to the meta-information which is given about the COSTA Stochastic Observer component by the COSTA Observer Description component. The meta-information object will be constructed in an analogous way.

4 Combining COSTA models

4.1 Combining noise models and a deterministic model

A very important example of a combined model is the combination of a deterministic model and a noise model.

- The **deterministic model** has the state $s_d = (\phi, u, g)$. There are no noise parameters. The propagation rule is:

$$\phi(t_{i+1}) := A[\phi(t_i), u(t_i), g] \tag{12}$$

- The **noise model for the forcings** has the state $s_u = (p^u)$. The propagation rule is:

$$\begin{aligned} p^u &:= \mathrm{diag}(\alpha^u)\ p^u + \eta^u(t_i), \\ \eta^u(t_i) &\equiv N(0,1). \end{aligned} \tag{13}$$

- The **noise model for the solution** has the state $s_A = (p^A)$. The propagation rule is:

$$\begin{aligned} p^A &:= \mathrm{diag}(\alpha^A)\ p^u + \eta^A(t_i). \\ \eta^A(t_i) &\equiv N(0,1). \end{aligned} \tag{14}$$

These three submodels are each considerably simpler than the stochastic model described in Section 3.1.

The state ϕ of the combined model is the concatenation of the states of the three submodels; so are the other . The COSTA state vector component has the functionality to handle concatenated vectors.

The same thing can be said for the forcings, schematization and noise parameters. The combined covariance matrix has a block diagonal structure.

The propagation of the combined model consists of the following steps:

1. Interpolate the forcings-parameters and add the forcings-noise to the forcings

$$u(t_i) := u(t_i) + W^u p^u \tag{15}$$

2. Propagate the deterministic model

$$\phi := A[\phi, u(t_i), g] \tag{16}$$

3. Interpolate the model-noise and add the model-noise to the solution

$$\phi := \phi + W^A p^A. \tag{17}$$

4. Propagate forcings-noise

$$p^u := \mathrm{diag}(\alpha^u)\ p^u + \eta_u(t_i) \tag{18}$$

5. Propagate model-noise

$$p^A := \mathrm{diag}(\alpha^A)\ p^A + \eta_A(t_i) \tag{19}$$

In this example, it is clear how the submodels should be combined into a combined model, using the functions in the interface of the COSTA model component, and interpolations needed for the commnunication between the submodels.

4.2 Correctness of the overall propagation equation

The previous section was a simple but important example of a combined model. A wide variety of coupled models may be imagined, in which not only one deterministic model and a (possibly large) number of noise models, but also multiple deterministic models as well as complete stochastic models.

There are certain restrictions to the things which may be coupled. The reason is the equation (7), in which the propagation of the complete state is described. This equation is crucial because it is the starting point of almost every data assimilation method. Complex coupled models may have a propagation equation which does not fit this form. The reason is that the random vector η is not only added to the 'combined extended state', but also used in non-linear calculations. This section gives an example of such an illegal coupled model.

The illegal model consists of a deterministic models and a noise model. The noise model (p^u) for the flow calculations is coupled to the deterministic model (ϕ, u, g) in the following way:

$$\begin{aligned} p^u &:= \mathrm{diag}(\alpha^u) p^u + \eta_u, \\ \phi &:= A_\phi[\phi, u(t_i) + W^u\ p^u, g]. \end{aligned} \tag{20}$$

The random vector η_u is used in the propagation operator A_ϕ. A very simple remedy to this problem is to change the order of the calculations, and obtain

$$\begin{aligned} \phi &:= A_\phi[\phi, u(t_i) + W^u\ \tilde{p}^u, g]. \\ \tilde{p}^u &:= \mathrm{diag}(\alpha^u)\tilde{p}^u + \tilde{\eta}_u, \end{aligned} \tag{21}$$

This reordered system is equivalent to the previous system, with

$$\tilde{p}^u(t_i) = p_u(t_{i-1}) \quad , \quad \tilde{\eta}_u(t_i) = \eta_u(t_{i-1}). \tag{22}$$

The ModelCombiner will have to be able to check that the propagation equation corresponds to equation (7). A sufficient (but perhaps not necessary) condition is that the propagation steps for models whose covariance matrix is nonempty must be the very last actions: after the first such model is propagated, the only calculations allowed are propagations of other models, and no more values may be exchanged.

If the combined model does not conform to this condition, the user may be able to make it that way by reordering the calculations.

4.3 Configuration file for ModelCombiner

The previous sections provide some insight into the way a combined model will work and the things that must be specified to the ModelCombiner. In this Section, it will be explained how the COSTA user (i.e. the model programmer) can describe the coupled model to the ModelCombiner.

The information is presented to the ModelCombiner in the form of a configuration file, written in XML.

The overall structure of an input file for the ModelCombiner is given in Table 1. It consists of two parts: the definitions of the submodels and the specification of the propagation step of the combined model.

An example of the definitions of the submodels is given in Table 2. Every submodel is given a name. Some additional information is necessary since the combined model creates its own submodels.

Table 3 gives an example of the 'actions' which constitute the propagation of the extended solution in the combined model. The example is very similar to (but a little more extended than) the steps given in Section 4.1. Every step in the propagation is called an 'action'. Several kinds of actions are distiguished:

- `set`, `get`, `axpy`:

  Set , get or adjust a part of the submodel state, forcings or parameters, using the values and meta-information of the state, forcings or parameters of an other submodel.

```
<modelbuilder model="stochastic model">

   <submodels>

        submodel definitions, between <submodel>

   </submodels>

   <propagation>
       <deterministic>

            things to be done for the propagation of the combined, extended
            solution, between <action type=*>, except the propagation of
            stochastic models

       </deterministic>
       <stochastic>
          <action type=propagate> boundary noise model  </action>
          <action type=propagate> wind noise model      </action>
          <action type=propagate> viscosity noise model </action>
          <action type=propagate> velocity noise model  </action>
       </stochastic>
   </propagation>

</modelbuilder>
```

Table 1: *Overall structure of the input file for the ModelCombiner*

```
<submodel>
       <name>         deterministic model      </name>
       <model_class>  CTA_WAQUA_MODEL          </model_class>
       <create_input> control_simona.txt       </create_input>
</submodel>

<submodel>
       <name>         boundary noise model     </name>
       <model_class>  CTA_MODEL_BUILDER        </model_class>
       <create_input> boundary_noise_model.xml </create_input>
</submodel>

submodel definitions for wind noise model, viscosity noise model and velocity
noise model, similar to that of boundary noise model
```

Table 2: Example of the submodel definitions in the input file

```
<action type=axpy>
      <input_y  var=state>    boundary noise model </input_y>
      <const_a>               1.0                  </const_a>
      <output_x var=forcings> deterministic model  </output_x>
</action>

forcing adaptations for wind noise model and viscosity noise model, similar to that
of boundary noise model


<action type=propagate>    deterministic model     </action>

<action type=axpy>
      <input_x  var=state> velocity noise model    </input_x>
      <const_a>            1.0                     </const_a>
      <output_y var=state> deterministic model     </output_y>
</action>
```

Table 3: *Example of the specification of the all the steps in the propagation of the combined model, except propagation of stochastic models.*

- `propagate`:

  Carry out time step calculation(s) until the desired simulated time level.

5 Planned work and time estimates

Getting the ModelCombiner up and running requires the following extensions to the COSTA environment:

1 Extensions to the COSTA model component.

  1a State, forcings and schematization description component

  1b Set, get and axpy-operations using description component

2 Extending the ModelBuilder so that AR(x) noise models may be set up quickly and easily.

3 Creating a new COSTA model-class, called CTA_MODEL_COMBINER.

4 Making the interface functions:

  4a A 'create' function, using an XML tree as input, which creates the substates and administrates the structure of the combined model.

  The COSTA ModelBuilder will be extended so that generic noise models can be supported, consisting of a parameter vector, a propagation matrix (which may be full, diagonal, or a scalar times unity), and a covariance matrix G.

  4b A 'free' function, which frees the submodels and the internal administration of the combined model.

  4c The get/set/axpy functions for state, forcings, schematization (parameters) and covariance matrices, which work on the concatenated states, forcings, schematizations and covariance matrices of the submodels.

  4d The observation functions.

  4e The propagation function.

6 Using the ModelCombiner, for validation:

  6a For the heat-model example

  6b For WAQUA.

7 Unit tests for the ModelCombiner, stochastic model builder facilities and interpolation function-facilities in COSTA

8 Documentation (Programmers'/users' guide)

		Description	Hours	
1		**COSTA model component**		
	a	Description component	6	
	b	AXPY using description component	6	
	c	Extension to COSTA state componenent for partly matching states	8	
		TOTAL		20
2		**Extension of COSTA ModelBuilder for generic noise models**		12
3		**Setting up new COSTA model class**		4
4		**Creating interface for ModelBuilder COSTA models**		
	a	Create-function (calling Create for submodels)	12	
	b	Free-function (calling Free for submodels)	4	
	c	Get/set/axpy for state	4	
		Get/set/axpy for forcings	4	
		Get/set/axpy for parameters	4	
	d	function `GetObsValues`	12	
		function `GetObsSelect`	8	
	e	Propagation	32	
		TOTAL		80
5		**Interpolation function facilities**		
		Generic identity function	4	
		Substate selection function	4	
		(m,n) interpolation	8	
		TOTAL		16
6		**Using the ModelCombiner**		
	a	In heat model	12	
	b	In WAQUA (without smoothing functionality)	24	
		TOTAL		36
7		**Unit tests**		
		State functions	4	
		ModelCombiner	8	
		TOTAL		14
8		**Documentation**		8
		Unexpected (20%)		37
TOTAL				223