COSTA
common set of tools for the assimilation of data

MEMO	CTA memo200701
Date	October 6, 2008
Author(s)	Erwin Loots and Nils van Velzen
Subject	The COSTA ModelCombiner

Document information

Version	Author	Date	Description	Review
1.0	EL	2007-07-11	Initial version	CvV
File location:		<COSTA_DIR>/doc/modelcombiner		

Table of contents

1 Introduction **1**

2 COSTA-models **2**

2.1 Mathematical description of a COSTA-model 2

2.2 Interface functions of the COSTA model component 4

2.3 Extension of the interface, necessary for the ModelCombiner 5

3 Combining COSTA models **6**

3.1 Combining noise models and a deterministic model 6

3.2 Configuration file for ModelCombiner 8

3.3 The AR(1) noise model 9

4 examples **10**

4.1 oscillation model 10

4.2 heat model 12

1 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.

2 COSTA-models

2.1 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.

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

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:

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

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.

2.2 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

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

  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 provides all the functions of the interface.

2.3 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 makes 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 is be constructed in an analogous way.

More specific, a metainfo structure is attached to each state. In general, it consists of some detailed information, description and unit of the quantity. An important part is the grid information. The vector in a state leaf lives on a grid. Up to now only linear grids are supported.

The advantage is that in the case of an axpy-operation, the grid can be used to perform necessary operations. In this way a coarse noise vector can be added to a dense state vector.

Metainfo is not obligatory. States can be used without metainfo, as always was the case, but operations are restricted somewhat. For example, an axpy operation between two states is only possible if the states and their substates have the same length.

Description of the current metainfo structure:

- tag: a short unique description for identification purposes like 'sep' or 'u'. This is used for binary operations like axpy: the function, when called with two large states as input, knowns that it performs the operation on substates with the same tag. Substates which do not have a matching metainfo-tag in any of the substates of the other state, are ignored. it can
- belongs_to: a second tag, also for binary operations. The user can in this way specify that 'sep' and 'noise_of_sep' belong to each other.
- unit: the unit of the quantity described.
- grid This is a separate structure attached on the metainfo.
  - type: 2D or 3D
  - name
  - x_origin, y_origin
  - nx,ny,nz: dimension in each direction
  - dx,dy,dz: distance between two grid points in each direction

description of metainfo operations:

CTAMetainfo_Create(CTA_Metainfo *minfo);

CTAMetainfo_Copy(CTA_Metainfo minfo1, CTA_Metainfo minfo2);

CTAMetainfo_SetUnit(CTA_Metainfo minfo, char* unit);

CTAMetainfo_GetUnit(CTA_Metainfo minfo, char* unit);

CTAMetainfo_SetTag(CTA_Metainfo minfo, char* tag);

CTAMetainfo_GetTag(CTA_Metainfo minfo, char* tag);

CTAMetainfo_SetBelongsTo(CTA_Metainfo minfo, char* tag);

CTAMetainfo_GetBelongsTo(CTA_Metainfo minfo, char* tag);

CTA_Metainfo_GetGrid(CTA_Metainfo minfo, CTAI_Gridm *hgrid);

CTA_Metainfo_SetGrid(CTA_Metainfo minfo, CTAI_Gridm *hgrid);

3 Combining COSTA models

3.1 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 2.1.

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.

3.2 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.

```
  <modelbuilder model="stochastic model" tagstate="a"
                    tagforc="b" tagparam="c">


   <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

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 3.1. 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.

- `propagate`:

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

3.3 The AR(1) noise model

Most users start with their own deterministic model. To make things easier, this model can be combined with a standard AR(1) noise model provided by COSTA. The AR(1) model can be created using a special variant of the Model Builder. That means that the functions that normally have to be provided by the user, are now already given by COSTA. The only aspects that the user has to specify are the characteristic parameters and the grid. An example of the specification in XML is shown in Table 4. The XML-code represents the `<create_input>}`entry in the submodel definition.

The necessary parameters are the standard deviation `std_dev`, the characteristic time `kar_t` and the characteristic length `kar_l`.

$$x_{n+1} = \alpha \cdot x_n + \sqrt{1 - \alpha^2} \cdot \eta$$

```
<action type="x=axpy">
      <var_y  var="state">    boundary noise model </var_y>
      <const_a>              1.0                 </const_a>
      <var_x var="forcings"> deterministic model  </var_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="y=axpy">
      <var_x  var="state"> velocity noise model    </var_x>
      <const_a>           1.0                      </const_a>
      <var_y var="state"> deterministic model     </var_y>
</action>
```

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

where $\alpha = \exp\left(-\delta t kar_t\right)$ and η the covariance matrix computed from the grid and the characteristic length `kar_l`.

4 examples

4.1 oscillation model

As a first example, the oscillation model (without noise) has been combined with an AR(1) process. Note that the grid specification is in this case not entirely logical since the first element of the state describes the (non-constant) position. However, this example will provide some insight in the working of the ModelCombiner.

Files:

`models/oscill/oscill.f90`

`models/oscill/c_oscill.c`

`applications/da_tools/ens_proto/ens_combine_oscill_ar1.f90`

`applications/da_tools/ens_proto/ens_combine_oscill_ar1.xml`

`applications/da_tools/ens_proto/sm_oscill_model.xml`

```
<modelbuild_sp>
   <special_ar1>
     <tag>noise-bc-n</tag>
     <parameters std_dev="1.0" kar_t="105.90" kar_l="0.5">
     </parameters>
     <grid>
        <type_id>2D</type_id>
        <gridsize  nx="4"  ny="1">  </gridsize>
        <gridparams x_origin="0.0" y_origin="2.5"
                   dx="0.5" dy="0.0">
        </gridparams>
     </grid>
   </special_ar1>

</modelbuild_sp>
```

submodel definition: an AR(1) noise model

Table 4: Example of the AR(1) submodel in the input file

`applications/da_tools/ens_proto/sm_ar1_model.xml`

4.2 heat model

As a second example, a second instance of the heat model has been made. The model consists now of the single temperature state, and five forcings: the heat and four boundary condition states. The idea is that the forcings are adjusted using the axpy operation with the corresponding five ar(1) models. In this case, the grids of the ar(1) models are twice as coarse as the grids where the forcings live.

Files:

`models/heat_modelcombiner/heat_model.f`

`models/heat_modelcombiner/c_heatmodel.c`

`applications/da_tools/ens_proto/ens_combine_heat_ar1.f90`

`applications/da_tools/ens_proto/ens_combine_heat_ar1.xml`

`applications/da_tools/ens_proto/sm_ar1_model_nheat_n.xml`

`applications/da_tools/ens_proto/sm_ar1_model_nbc_n.xml`

`applications/da_tools/ens_proto/sm_ar1_model_nbc_e.xml`

`applications/da_tools/ens_proto/sm_ar1_model_nbc_w.xml`

`applications/da_tools/ens_proto/sm_ar1_model_nbc_s.xml`