An object oriented model framework for the future of COSMO model

Davide Cesari

Servizio Idro Meteo

Regional Hydrometeorological Service of Emilia-Romagna

Bologna, Italy

COSMO general meeting
Αθήνα, Ελλάς, September 2007

Outline

Outline

Advantages

A new object-oriented framework for the COSMO model could allow:

- + easier maintainance of the model code
- + faster introduction of scientific novelties that may emerge from COSMO countries and the from the “formerly called” LM-users (urban and climatological features, chemistry) in the mainstream code without affecting the stability for operational NWP
- + native (re)introduction of features, like 2-way nesting, abandoned because of complexity and bugs
- + more natural introduction of new features that are required by the changes in the scientific scenario, like new Data Assimilation techniques, (4Dvar or whatever?) etc.

Advantages

A new object-oriented framework for the COSMO model could allow:

+ easier maintainance of the model code
+ faster introduction of scientific novelties that may emerge from COSMO countries and the from the “formerly called” LM-users (urban and climatological features, chemistry) in the mainstream code without affecting the stability for operational NWP
+ native (re)introduction of features, like 2-way nesting, abandoned because of complexity and bugs
+ more natural introduction of new features that are required by the changes in the scientific scenario, like new Data Assimilation techniques, (4Dvar or whatever?) etc.

Advantages

A new object-oriented framework for the COSMO model could allow:

+ easier maintainance of the model code
+ faster introduction of scientific novelties that may emerge from COSMO countries and the from the “formerly called” LM-users (urban and climatological features, chemistry) in the mainstream code without affecting the stability for operational NWP
+ native (re)introduction of features, like 2-way nesting, abandoned because of complexity and bugs
+ more natural introduction of new features that are required by the changes in the scientific scenario, like new Data Assimilation techniques, (4Dvar or whatever?) etc.

Advantages

A new object-oriented framework for the COSMO model could allow:

+ easier maintainance of the model code
+ faster introduction of scientific novelties that may emerge from COSMO countries and the from the "formerly called" LM-users (urban and climatological features, chemistry) in the mainstream code without affecting the stability for operational NWP
+ native (re)introduction of features, like 2-way nesting, abandoned because of complexity and bugs
+ more natural introduction of new features that are required by the changes in the scientific scenario, like new Data Assimilation techniques, (4Dvar or whatever?) etc.

Disadvantages

- a lot of work to be done
- a change in the habits by the scientific code developers is required
- a wrong initial planning may require big efforts later for being corrected, with many changes spread thoroughout the code
- a lot of work to be done

Disadvantages

- a lot of work to be done
- a change in the habits by the scientific code developers is required
- a wrong initial planning may require big efforts later for being corrected, with many changes spread thoroughout the code
- a lot of work to be done

Disadvantages

- a lot of work to be done
- a change in the habits by the scientific code developers is required
- a wrong initial planning may require big efforts later for being corrected, with many changes spread thoroughout the code
- a lot of work to be done

Disadvantages

- a lot of work to be done
- a change in the habits by the scientific code developers is required
- a wrong initial planning may require big efforts later for being corrected, with many changes spread thoroughout the code
- a lot of work to be done

Outline

Definition of a class-like module in F90, storage

```
MODULE spacetime_grid_class
USE datetime_class
IMPLICIT NONE

PRIVATE
PUBLIC spacetime_grid, ini_t, act_t, fin_t

TYPE(datetime) :: ini_t, act_t, fin_t ! Class static variables

TYPE spacetime_grid ! Instance variables
  PRIVATE
  TYPE(datetime) :: ini_t, act_t, fin_t
  TYPE(timedelta) :: dt
  INTEGER nx, ny
END TYPE spacetime_grid

END MODULE spacetime_grid_class
```

Definition of a class-like module in F90, storage

```
PRIVATE
PUBLIC spacetime_grid, ini_t, act_t, fin_t

TYPE(datetime) :: ini_t, act_t, fin_t ! Class static variables
```

- `PRIVATE` should be the default whenever possible
- class public static storage should be limited to the minimum necessary (truly global variables needed by other classes)

This will avoid having long lists of `USE ... ONLY` in order to document the external variables and will reduce the cross-dependencies between classes.

Definition of a class-like module in F90, storage

```
TYPE spacetime_grid ! Instance variables
  PRIVATE
  TYPE(datetime) :: ini_t, act_t, fin_t
  TYPE(timedelta) :: dt
  INTEGER nx, ny
END TYPE spacetime_grid
```

- the class instance storage should have the `PRIVATE` attribute when possible, but this is not as strict as the previous guidelines

This will give more freedom to change the internal structure of the class without affecting the procedures that `USE` it.

Definition of a class-like module in F90, storage

Benefits from F2003

Selective `PRIVATE`/`PUBLIC` attributes for single components of a derive type and the `PROTECTED` attribute for read-only components.

Definition of a class-like module in F90, methods

```
MODULE spacetime_grid_class
USE datetime_class
IMPLICIT NONE

PRIVATE
PUBLIC spacetime_grid, init, delete, compute,

INTERFACE init
  MODULE PROCEDURE spacetime_grid_init
END INTERFACE

CONTAINS

SUBROUTINE spacetime_grid_init(this)
TYPE(spacetime_grid), INTENT(inout) :: this
...

END MODULE spacetime_grid_class
```

Outline

Description of the prototype

A model framework prototype has been implemented (initially planned for the VHREM project, it remained in my mind for almost a year, then it has been written down during the last 4 weeks), the current functionality is:

- reads a minimal configuration (grid size, time step, nested grid hierarchy)
- can do time stepping with multiple nested grids
- includes “dummy” dynamics and parallel environment modules
- has an experimental model and variable table configuration system
- includes “hooks” for relaxation, grid interaction, physics, assimilation, I/O

Description of the prototype

A model framework prototype has been implemented (initially planned for the VHREM project, it remained in my mind for almost a year, then it has been written down during the last 4 weeks), the current functionality is:

- reads a minimal configuration (grid size, time step, nested grid hierarchy)
- can do time stepping with multiple nested grids
- includes “dummy” dynamics and parallel environment modules
- has an experimental model and variable table configuration system
- includes “hooks” for relaxation, grid interaction, physics, assimilation, I/O

Description of the prototype

A model framework prototype has been implemented (initially planned for the VHREM project, it remained in my mind for almost a year, then it has been written down during the last 4 weeks), the current functionality is:

- reads a minimal configuration (grid size, time step, nested grid hierarchy)
- can do time stepping with multiple nested grids
- includes “dummy” dynamics and parallel environment modules
- has an experimental model and variable table configuration system
- includes “hooks” for relaxation, grid interaction, physics, assimilation, I/O

Description of the prototype

A model framework prototype has been implemented (initially planned for the VHREM project, it remained in my mind for almost a year, then it has been written down during the last 4 weeks), the current functionality is:

- reads a minimal configuration (grid size, time step, nested grid hierarchy)
- can do time stepping with multiple nested grids
- includes “dummy” dynamics and parallel environment modules
- has an experimental model and variable table configuration system
- includes “hooks” for relaxation, grid interaction, physics, assimilation, I/O

Description of the prototype

A model framework prototype has been implemented (initially planned for the VHREM project, it remained in my mind for almost a year, then it has been written down during the last 4 weeks), the current functionality is:

- reads a minimal configuration (grid size, time step, nested grid hierarchy)
- can do time stepping with multiple nested grids
- includes “dummy” dynamics and parallel environment modules
- has an experimental model and variable table configuration system
- includes “hooks” for relaxation, grid interaction, physics, assimilation, I/O

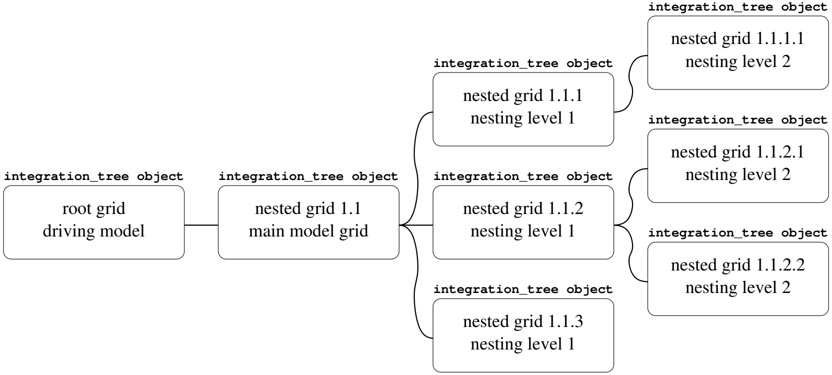
Outline

Integration tree class

The main class which drives the integration process is called `integration_tree`:

- describes a full application of the “model operator” on a single grid
- carries pointers to objects of the same class describing parent grid and child grid(s)
- the “root” of the integration tree could be the driving model data interpolated on the main computational grid (special case of “non prognostic” `integration_tree` object)
- the child of the root grid would then be the main “prognostic” grid
- the child(ren) of the main computational grid may represent nested grids and so on

Scheme of the `integration_tree` grid hierarchy



Integration tree class contents

The `integration_tree` class contains classes representing all the numerical packages of the model: dynamics, physics (each parameterization should reside in a separate subclass), assimilation, chemistry, etc. + Input/Output

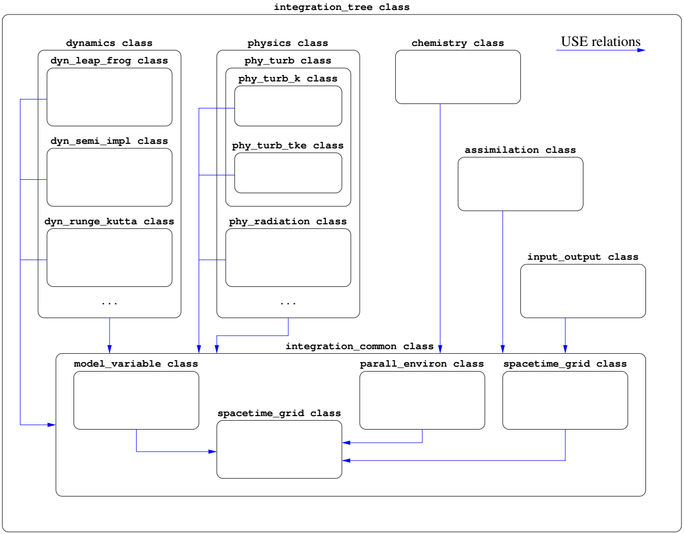
- these classes are assumed independent one of each other, this makes the management of the code easier but may restrict the freedom in developing numerical modules
  - ⇒ two classes (modules) cannot reference each other (circular references forbidden in f90), so if two classes (e.g. convection and turbulence parameterisation) have to share some piece of code or data, this piece should be extracted from the two classes and placed in a separated module `USE`'d by both.

Integration tree class contents

The `integration_tree` class contains classes representing all the numerical packages of the model: dynamics, physics (each parameterization should reside in a separate subclass), assimilation, chemistry, etc. + Input/Output

- the classes (e.g. model grid, model variables, parallel computing environment) that need to be accessed by most of the numerical classes are grouped in a special class `integration_common` contained by `integration_tree`

Scheme of the `integration_tree` class



`USE` relations are indicated with blue arrows; every class in the scheme also makes `USE` of the class(es) immediately contained in it (not indicated explicitly in the scheme). ⇒

Management of model variables

In order to simplify the management of variables and variable tables, the variables are handled by the `model_variable` class (contained in `integration_common` class) with the following phases:

Management of model variables

- `model_variable` class reads the variable table from a file
- all the numerical modules, after reading the configuration, make a “reservation” for the variables they need (identifying them by name), by calling a `reserve` method in `model_variable` class, specifying e.g. whether the variable is prognostic, how it is staggered, etc.
- the `alloc` method of the `model_variable` class is called, which does a sanity check and allocates the variables which have been “reserved”
- at the beginning of every time step, the numerical modules call the `get` method of the `model_variable` class in order to get a pointer to the variables/tendencies needed, at the required time levels

Management of model variables

- model_variable class reads the variable table from a file
- all the numerical modules, after reading the configuration, make a “reservation” for the variables they need (identifying them by name), by calling a reserve method in model_variable class, specifying e.g. whether the variable is prognostic, how it is staggered, etc.
- the alloc method of the model_variable class is called, which does a sanity check and allocates the variables which have been “reserved”
- at the beginning of every time step, the numerical modules call the get method of the model_variable class in order to get a pointer to the variables/tendencies needed, at the required time levels

Management of model variables

- `model_variable` class reads the variable table from a file
- all the numerical modules, after reading the configuration, make a “reservation” for the variables they need (identifying them by name), by calling a `reserve` method in `model_variable` class, specifying e.g. whether the variable is prognostic, how it is staggered, etc.
- the `alloc` method of the `model_variable` class is called, which does a sanity check and allocates the variables which have been “reserved”
- at the beginning of every time step, the numerical modules call the `get` method of the `model_variable` class in order to get a pointer to the variables/tendencies needed, at the required time levels

Management of model variables

- `model_variable` class reads the variable table from a file
- all the numerical modules, after reading the configuration, make a "reservation" for the variables they need (identifying them by name), by calling a `reserve` method in `model_variable` class, specifying e.g. whether the variable is prognostic, how it is staggered, etc.
- the `alloc` method of the `model_variable` class is called, which does a sanity check and allocates the variables which have been "reserved"
- at the beginning of every time step, the numerical modules call the `get` method of the `model_variable` class in order to get a pointer to the variables/tendencies needed, at the required time levels

Management of model variables

This way, adding a model variable is just a matter of inserting its description in a configuration file and inserting the proper "reservation" where the variable is needed
$\Rightarrow$ easier maintenance and more freedom for scientific developers.

Management of model variables

Another feature —planned but only partially implemented— is the possibility to group different variables into a single one with an additional dimension, in order to simplify the treatment of high numbers of microphysical or chemical species, e.g. in the advection

Benefits from F2003

Procedure pointers to associate each diagnostic variable to the corresponding method for its computation, may simplify the I/O module.
Procedure pointers to associate prognostic variables to the method for computing falling velocity field, may simplify the advection code.

Numerical computing classes

They should have the following methods:

- `init` (constructor) which sets up the storage, reads the configuration and makes the variable “reservation”
- `compute` performs computation for a single timestep
- other standardized methods could be added for performing additional operations, like `compute_tangent_linear`, `compute_adjoint`, `checkpoint`, `restart`...

Numerical computing classes

- numerical classes are in principle free to allocate their own private arrays or other kind of data
- these data should be placed in the instance storage (the main `TYPE` definition) if they have to be conserved between calls and have to be unique to each grid
- they can either update the tendencies of the prognostic variables or the variable themselves in a time-splitting manner

Numerical computing classes - alternative schemes

If more than one numerical scheme is available for the dynamics or for a physical parameterization, this could be implemented as a driving class with pointers to the specific numerical classes, rather than physical inclusion of them.
Only the pointer to the desired scheme/class will then be `ASSOCIATED()`. See also the class scheme.

Benefits from F2003

"True" classes with type extension, polymorphism and bounded procedures can help in avoiding code repetition and simplifying the management of the alternative numerical schemes.

Numerical computing classes - software interoperability

- if a more traditional programming approach is desired, where, e.g., prognostic variables are to be called `u`, `v`, `p`, etc. and not `this%u`, `this%v`, `this%p`, then the `compute` method can act as a wrapper to a traditional routine whose parameters are all the needed variables and configuration parameters
- this would allow to call modules written according to the old "Rules for interchange of physical parameterizations" (Kalnay et. al), either including the old code in the module (better) or leaving it as external routines

Numerical computing classes - software interoperability

- if a more traditional programming approach is desired, where, e.g., prognostic variables are to be called `u`, `v`, `p`, etc. and not `this%u`, `this%v`, `this%p`, then the `compute` method can act as a wrapper to a traditional routine whose parameters are all the needed variables and configuration parameters
- this would allow to call modules written according to the old “Rules for interchange of physical parameterizations” (Kalnay et. al), either including the old code in the module (better) or leaving it as external routines

Outline

Using xml files instead of namelists

The possibility to use xml files through a public domain, f90 package to provide model run configuration has been investigated:

- a special tool reads a description of the "namelist", relative to a single module/class, from an xml file and generates a f90 module source that can read the desired xml structure into a proper f90 derived type
    - the description includes variable types, rank, dimensions (fixed or allocatable runtime) and initial default value
- the module/class needing configuration `USE`s the automatically generated module and `CALL`s the corresponding reading routines
- if all the configuration for a module/class is contained in a single derived type (without `POINTER` variables), it is simpler to exchange it in parallel mode (this would be true for a namelist too, but it is uncomfortable to read a derived type in a namelist)

Using xml files instead of namelists

The possibility to use xml files through a public domain, f90 package to provide model run configuration has been investigated:

- a special tool reads a description of the “namelist”, relative to a single module/class, from an xml file and generates a f90 module source that can read the desired xml structure into a proper f90 derived type
  - the description includes variable types, rank, dimensions (fixed or allocatable runtime) and initial default value
- the module/class needing configuration `USE`s the automatically generated module and `CALL`s the corresponding reading routines
- if all the configuration for a module/class is contained in a single derived type (without `POINTER` variables), it is simpler to exchange it in parallel mode (this would be true for a namelist too, but it is uncomfortable to read a derived type in a namelist)

Using xml files instead of namelists

The possibility to use xml files through a public domain, f90 package to provide model run configuration has been investigated:

- a special tool reads a description of the "namelist", relative to a single module/class, from an xml file and generates a f90 module source that can read the desired xml structure into a proper f90 derived type
  - the description includes variable types, rank, dimensions (fixed or allocatable runtime) and initial default value
- the module/class needing configuration `USE`s the automatically generated module and `CALL`s the corresponding reading routines
- if all the configuration for a module/class is contained in a single derived type (without `POINTER` variables), it is simpler to exchange it in parallel mode (this would be true for a namelist too, but it is uncomfortable to read a derived type in a namelist)

Using xml files instead of namelists

The possibility to use xml files through a public domain, f90 package to provide model run configuration has been investigated:

- a special tool reads a description of the “namelist”, relative to a single module/class, from an xml file and generates a f90 module source that can read the desired xml structure into a proper f90 derived type
    - the description includes variable types, rank, dimensions (fixed or allocatable runtime) and initial default value
- the module/class needing configuration `USE`s the automatically generated module and `CALL`s the corresponding reading routines
- if all the configuration for a module/class is contained in a single derived type (without `POINTER` variables), it is simpler to exchange it in parallel mode (this would be true for a namelist too, but it is uncomfortable to read a derived type in a namelist)

Advantages and disadvantages

- \+ adding new configuration parameters requires less effort
- \+ variable-size arrays (allocated according to the input size) allowed
- \+ xml may allow easier interfacing with other applications
- \- may be trickier in case of errors
- \- input files more verbose
- \- requires a change of habit by the users

Advantages and disadvantages

+ adding new configuration parameters requires less effort
+ variable-size arrays (allocated according to the input size) allowed
+ xml may allow easier interfacing with other applications
- may be trickier in case of errors
- input files more verbose
- requires a change of habit by the users

Advantages and disadvantages

+ adding new configuration parameters requires less effort
+ variable-size arrays (allocated according to the input size) allowed
+ xml may allow easier interfacing with other applications
- may be trickier in case of errors
- input files more verbose
- requires a change of habit by the users

Advantages and disadvantages

+ adding new configuration parameters requires less effort
+ variable-size arrays (allocated according to the input size) allowed
+ xml may allow easier interfacing with other applications
- may be trickier in case of errors
- input files more verbose
- requires a change of habit by the users

Advantages and disadvantages

+ adding new configuration parameters requires less effort
+ variable-size arrays (allocated according to the input size) allowed
+ xml may allow easier interfacing with other applications
- may be trickier in case of errors
- input files more verbose
- requires a change of habit by the users

Advantages and disadvantages

+ adding new configuration parameters requires less effort
+ variable-size arrays (allocated according to the input size) allowed
+ xml may allow easier interfacing with other applications
- may be trickier in case of errors
- input files more verbose
- requires a change of habit by the users

Outline

Need for new coding rules

- an approach like the one described, especially in the absence of a true O-O language like F2003, requires a preliminar agreement about some strict standardization rules not covered by the known “European standards for... tran 90 code”
- the code to be developed should be “F2003-ready”, in order to switch to the new syntax when the new compilers will be available and popular enough

Need for new coding rules

- an approach like the one described, especially in the absence of a true O-O language like F2003, requires a preliminar agreement about some strict standardization rules not covered by the known “European standards for... tran 90 code”
- the code to be developed should be “F2003-ready”, in order to switch to the new syntax when the new compilers will be available and popular enough

Continuing the development in COSMO

Comments and discussion

- the prototype is available to COSMO (a demonstration can be done on my Linux laptop here, compiled with gfortran)
- suggestions and exchange of experience are welcome
- does this work meet any of the needs or willings of COSMO/DWD?
- if so, how can we proceed?

Continuing the development in COSMO

Comments and discussion

- the prototype is available to COSMO (a demonstration can be done on my Linux laptop here, compiled with gfortran)
- suggestions and exchange of experience are welcome
- does this work meet any of the needs or willings of COSMO/DWD?
- if so, how can we proceed?

Continuing the development in COSMO

Comments and discussion

- the prototype is available to COSMO (a demonstration can be done on my Linux laptop here, compiled with gfortran)
- suggestions and exchange of experience are welcome
- does this work meet any of the needs or willings of COSMO/DWD?
- if so, how can we proceed?

Continuing the development in COSMO

Comments and discussion

- the prototype is available to COSMO (a demonstration can be done on my Linux laptop here, compiled with gfortran)
- suggestions and exchange of experience are welcome
- does this work meet any of the needs or willings of COSMO/DWD?
- if so, how can we proceed?

References

- Object Oriented programming in F90
  `http://www.cs.rpi.edu/~szymansk/oof90.html`
- J.E. Akin, Object-Oriented Programming Via F95, Cambridge University Press, 2003
  `http://www.owlnet.rice.edu/~mech517/`
- XML Fortran web site (Arjen Markus, Delft Hydraulics)
  `http://xml-fortran.sourceforge.net/`
- Fortran standards committee `http://j3-fortran.org/`
- Fortran 2003 draft specification (hurry up until it's available)
  `http://www.dkuug.dk/jtc1/sc22/open/n3661.pdf`
- J. Reid, The new features of Fortran 2003 `ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/N1648.pdf`