Decorating code to expose algorithmic descriptions of the code to the CIM

Simon Wilson NCAS-CMS
Introduction

• Part of IS-ENES2 WP3 on developing convergent model codes.

• Comprehensive documentation and analysis of models is required so that modelling groups can be better informed of other models’ formulation at a detailed level.

• Aim is to provide systems useful in developing strategies for convergent model code using the CIM (Common Information Model).

• **NOTE:** only concentrating on atmospheric models in the study.
Models and documentation

- Many ways to document models, no consistency.
- Still takes effort to “decipher” documentation to understand model structure and prognostic dependencies.
- ~20 model code bases available globally, so any model intercomparison will be a time intensive task.
- But fundamentally most (all?) models consist of a number of prognostics (state variables) such as wind, temperature, moisture, pressure/density,… altered and/or moved by a number of model components (usually) split into physical parametrisations (physics) and fluid motions and thermodynamics (dynamics).
- Should be able to clearly represent the relationship between these in a model timestep, something like this...
My Office Wall.
The anatomy of a timestep in the UM.
Original version long lost.
Requirements

- Investigate relationships between model components and prognostics, focusing on dependencies and flow.
- Initially a comparison of the atmosphere timestep and code structure of three different atmosphere models:
  - Unified Model from the Met Office in a HadGAM3 configuration
  - HiRAM from GFDL
  - OpenIFS from ECMWF
- By component mean separate sections such as LW radiation, GWD, Convection and the Dynamics (possibly split).
How can we describe and document a model?
Fully-compressible, deep atmosphere equations

\[
\frac{D_r u}{Dt} - \frac{uv}{r} \tan \phi - 2\Omega \sin \phi v + \frac{uw}{r} + 2\Omega \cos \phi w + \frac{1}{\rho r \cos \phi} \frac{\partial p}{\partial \lambda} = F_u,
\]

\[
\frac{D_r v}{Dt} + \frac{u^2}{r} \tan \phi + 2\Omega \sin \phi u + \frac{v w}{r} + \frac{1}{\rho r} \frac{\partial p}{\partial \phi} = F_v,
\]

\[
\frac{D_r w}{Dt} - \frac{u^2 + v^2}{r} - 2\Omega \cos \phi u + g + \frac{1}{\rho} \frac{\partial p}{\partial r} = 0,
\]

where

\[
\frac{D_r}{Dt} = \frac{\partial}{\partial t} + \frac{u}{r \cos \phi} \frac{\partial}{\partial \lambda} + \frac{v}{r} \frac{\partial}{\partial \phi} + w \frac{\partial}{\partial r},
\]

and

\[
\nabla_r \cdot \mathbf{u} = \frac{1}{r \cos \phi} \left[ \frac{\partial u}{\partial \lambda} + \frac{\partial (v \cos \phi)}{\partial \phi} \right] + \frac{1}{r^2} \frac{\partial (r^2 w)}{\partial r}.
\]

\[
\frac{D_r \rho}{Dt} + \rho \nabla_r \cdot \mathbf{u} = 0, \quad \frac{D_r \theta}{Dt} = F_\theta, \quad p = R\rho T
\]

from Terry Davis
<table>
<thead>
<tr>
<th>Layer Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SURFRAD.LAYER</td>
<td>Computes radiative properties of the surface.</td>
</tr>
<tr>
<td>CLDPRG.LAYER</td>
<td>Calls CLDPP to compute cloud parameters required for the post processing (e.g. total cloud cover).</td>
</tr>
<tr>
<td>RADFLUX.LAYER</td>
<td>Calls RADHEATN to compute the temperature tendencies and the downward radiation fluxes at the surface with updated (every time-step) values for the zenith angle.</td>
</tr>
<tr>
<td>GWDRAG.LAYER</td>
<td>Computes the tendencies for $u$, $v$ and $T$ due to the parametrization of subgrid-scale orographic drag. It also computes subgrid orographic coefficients for use in VDFMAIN.</td>
</tr>
<tr>
<td>TURBULENCE.LAYER</td>
<td>Calls VDFOUTER/VDFMAIN in two sub time steps for numerical stability. VDFMAIN computes the vertical exchange of $u$, $v$, $T$, $q$, $a$, $q_l$, $q_i$, $q_r$ by turbulence.</td>
</tr>
<tr>
<td>CLOUD.LAYER</td>
<td>Calls CLOUDSC as a first guess call of cloud scheme to determine preliminary entry profiles for convection.</td>
</tr>
<tr>
<td>CONVECTION.LAYER</td>
<td>Interface to call CUCALLN/CUMASTRN that controls the computation of the tendencies for $u$, $v$, $T$, $q$, chemical tracers and the cloud detrainment term due to the parametrization of moist convective processes.</td>
</tr>
<tr>
<td>CLOUD.LAYER</td>
<td>Calls CLOUDSC to compute tendencies for $T$, $q$, $a$, $q_l$, $q_i$, $q_r$ and $q_s$ due to the parametrization of cloud and precipitation processes.</td>
</tr>
<tr>
<td>GWDRAGWMS.LAYER</td>
<td>Calls GWDRAG.WMS to compute the tendencies for $u$, $v$ and $T$ due to the parametrization of non-orographic gravity waves.</td>
</tr>
<tr>
<td>METHOX</td>
<td>Computes tendencies for $q$ due to methane oxidation and water vapour photolysis.</td>
</tr>
<tr>
<td>SURFTSTP.LAYER</td>
<td>Calls SURFTSTP to control the soil/surface scheme.</td>
</tr>
<tr>
<td>STOCHPERT.LAYER</td>
<td>Optionally add stochastic perturbations to physics tendencies.</td>
</tr>
<tr>
<td>O3CHEM</td>
<td>Computes tendencies for $O_3$ due to ozone chemistry.</td>
</tr>
<tr>
<td>SLTEND.LAYER</td>
<td>Optionally average tendencies from radiation, convection and cloud in time and space along the semi-Lagrangian trajectory.</td>
</tr>
</tbody>
</table>
NAMELIST
&coupler_nml

current_date

The date that the current integration starts with. [integer, dimension(6), default: 0]

force_date_from_namelist

Flag that determines whether the namelist variable current_date should override the date in the restart file INPUT/coupler.res. If the restart file does not exist then force_date_from_namelist has not effect, the value of current_date will be used. [logical, default: .false.]

calendar

The calendar type used by the current integration. Valid values are consistent with the time_manager module: 'julian', 'noleap', or 'thirty_day'. The value 'no_calendar' can not be used because the time_manager's date function are used. All values must be lowercase. [character(maxlen=17), default: '']

months

The number of months that the current integration will be run for. [integer, default: 0]

days

The number of days that the current integration will be run for. [integer, default: 0]
Do level = 1, model_levels
! DEPENDS ON: trsrce
   Call trsrce(
       rows, row_length, offx, offy &
       &, halo_i, halo_j, model_levels, wet_levels &
       &, 0, 0 &
       &, theta, q_n, qcl_n, qcf_n, exner_rho_levels, rho &
       &, aerosol(:,:,level), aerosol_em (:,:,level) &
       &, level, timestep, val_hour, val_minute, amp &
   &)
   End Do
End If

If ( L_murk_bdry ) Then
! DEPENDS ON: trbdry
   Call trbdry(
       row_length, rows, n_rows, model_levels &
       &, offx, offy, at_extremity &
       &, p, u, v &
       &, aerosol, timestep &
   &)
   End If
• So, by inspection, from model documentation papers (which can number in the 10s), code comments and documentation, code analysis, speaking to model gurus (and some guesswork), generated these...
• Surely there must be a standard way to document models.
• Yes there is, **THE CIM**.
• CIM constructs exist already:
  • Variable - prognostics
  • **SoftwareComponent** – model component
CIM construct descriptions

software.SoftwareComponent: (  
    'composition',
    'connection_points',
    'coupling_framework',
    'dependencies',
    'depends_on',
    'grid',
    'language',
    'license',
    'sub_components',
    'description',
    'development_history',
    'documentation',
    'long_name',
    'name',
    'release_date',
    'repository',
    'version',
),

software.Variable: (  
    'description',
    'name',
    'prognostic',
),

This corresponds to the prognostic variables

The input SoftwareComponent dependencies
How to populate the CIM?

• Can use the structure of the model code.

• Automatic code analysis tools available, but these work on names and structure of functions/subroutines.

• No simple way to identify and separate top level subroutines into functional blocks of code for each model component (Radiation, GWD etc…) 

• Identification of prognostics via variable name equally as difficult

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of files</th>
<th>Number of code statements</th>
<th>Number of subroutines</th>
<th>Number of modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>HadGEM3A</td>
<td>1553</td>
<td>217972</td>
<td>2217</td>
<td>628</td>
</tr>
<tr>
<td>HiRAM</td>
<td>297</td>
<td>372394</td>
<td>4505</td>
<td>287</td>
</tr>
<tr>
<td>OpenIFS</td>
<td>2776</td>
<td>160286</td>
<td>3675</td>
<td>624</td>
</tr>
</tbody>
</table>

• Direct commenting of code by model experts required.
After examining a number of automatic code documenting systems, ROBODoc was chosen.

Specially formatted documentation headers are extracted from (Fortran) source files and stored in a different reformatted file.

Metadata can be configured by user.

ROBODoc can reformat the documentation in HTML, XML DocBook, TROFF, ASCII, LaTeX or RTF format.

Example format...
!!****f* main/main
!! NAME
!! Met Office Unified Model
!!***
...

!!****p* prog/u
!!* NAME
!!* U wind
!!* DESCRIPTION
!!* Wind blowing to the East.
!!***

!!****f* phy/Conv
!!* NAME
!!* Convection
!!* DESCRIPTION
!!* Parametrization of convective clouds, their precipitation, phase changes and transports of heat, water and momentum.
!!* INPUTS
!!* u v theta q
!!* USES
!!* LSCloud BLayer
!!* VERSION
!!* A05_6A
!!***
Workflow

• Header block in code containing base model description and prognostic details.

• For each Software component (GWD, Convection etc) control routine, a RODODoc comment block added.

• Model code tree analysed by ROBODoc, and a DocBook XML file generated.

• Python code to read in XML and fill CIM directly from contents.

• Once model details in CIM, simple (a few lines in python) to traverse SoftwareComponent list to generate graphviz dot format flowchart.
SoftwareComponent construct

```json
{"name": 'Convection',

'\textit{description}': '\textit{Parametrization of convective clouds, their precipitation, phase changes and transports of heat, water and momentum.}',

'depends_on': [
<pyesdoc.ontologies.cim.v2.typeset_for_software_package.SoftwareComponent object at 0x7f8303f8e0d0>,
<pyesdoc.ontologies.cim.v2.typeset_for_software_package.SoftwareComponent object at 0x7f8303f8d0d0>],

'version': 'A05_6A',

'connection_points': [
<pyesdoc.ontologies.cim.v2.typeset_for_software_package.Variable object at 0x7f8303f6ff90>,
<pyesdoc.ontologies.cim.v2.typeset_for_software_package.Variable object at 0x7f8303f6ff0d0>,
<pyesdoc.ontologies.cim.v2.typeset_for_software_package.Variable object at 0x7f8303f6fed0>,
<pyesdoc.ontologies.cim.v2.typeset_for_software_package.Variable object at 0x7f8303f6ff10>],

\hspace{1cm} \text{LSCloud}
```
HiRAM by inspection

HiRAM auto-generated via CIM
Can output relational information:

Prognostic: (u) U wind
Flow:
  Vertical Diffusion 2
  Gravity Wave Drag
  Vertical Diffusion 1
  Solver up sweep step
  Solver up sweep control (moist)
  Solver down sweep control (dry)
  Update physics
  Solver down dynamics step
  Rayleigh damping

Component: (Precip) Precipitation
Prognostics:
  Temperature
  Moisture
Depends on:
  Large-Scale Condensation
• Demonstrated a consistent way to analyse model structure directly from model code comments.

• Using the CIM.

• Very much a work in progress.

• Prognostic type (increments, deltas, timestep etc…) not defined.

• No grid info either.

• CIM construct population not fully instrumented.
Questions
People who understand model formulation and science

People who understand model code formulation and structure

People you want to keep