OOPS technical meeting Number 4

about Jb code & Fortran coding standards

held on Wednesday April 20 ;
participants (MF) : Claude Fischer, Thibaut Montmerle, Pierre Brousseau, Ryad El Khatib
participants (EC) : Deborah Salmond, Mike Fisher

1. Jb code for CY38 and mid-term considerations

Mike recalls the basics about his present, new version of the Jb code: at this stage, the main aim is to move all Jb variables from global module variables into derived types. The different derived types are themselves grouped in a comprehensive top-level derived type (JB_STRUCT) which is the target to which a C_PTR (C++ compatible) pointer would point to. Mike already has tested his code in CY37R3 and it produces bit-identical results in the IFS.

Discussion took place on some specific issues: which modules should be considered as belonging to Jb (YOMCVA & control_vectors do NOT; the parameter JPAEROCV should be moved from YOMJG to a control vector related module; some LAM modules will need to be added => YEMVARGP, YEMWAVELET; YOMFGER needs to be checked by Mike). Phasing of LAM code seems feasible until CY38 and would consist in :

(1) adding LAM derived types into JB_STRUCT
(2) update the LAM code for using the overall new IFS derived types
(3) maybe a specific brainstorming still needed at some stage for how to treat the mean wind components ?

MF has no objection that Mike's code enters CY38, and LAM phasing will be done in Toulouse.

The discussion also made clear that several other steps of “oopsization” of the Jb code would follow post-CY38:

1. encapsulate the setup routines,
2. make the link with other objects (geometry, model): Mike mentions that the B will need to know about the geometry with which it is instantiated. EC mentions that they are starting to think about the geometry object; MF points out that this probably is a crucial part also for understanding the articulation one should have between objects/methods (the OO concepts) and the global/LAM paradigm (at least in the Fortran code: this is the historical backbone of the development of the Aladin code). The other LAM partners will need to be involved in the loop of information and design. Deborah recalls that the geometry will be part of the discussion at EC when Karim will visit the Center in May.
3. rationalize the link between Jb and control vector: Claude recalls the plans in the LAM community for solving the variational problem using extensions of the initial condition control vector (multiple B's based on geographic masks for rainy/non-rainy, cloudy/non-cloudy, fog/no-fog conditions; hybrid 3D-VAR/LETKF using additional weights for each ensemble member). Mike explains that the control vector should indeed be initialized by various pieces of the variational problem (initial condition, model error, VarBC, Tb coefficients, and so on ...).
4. others ? ...

According to EC, the 3DVar demonstrator should be available this autumn. This would especially imply the coding of the control vector inside OOPS which should be available after this summer.

Claude asks for some explanation about Mike's point on how to handle the structures and derived types inside the existing Fortran code by pointers. Mike recalls “Tomas' trick” which consists in copying the pointers from the derived types into global module variables (to be used in the model then) in order not to have to drastically modify all the existing code. This is the option retained for the present implementation of the “fieldsets” in CY38 (refer to Monday April 20 discussion), but this is potentially dangerous with multiple instances as one may forget about copying at an instance, in which case wrong variables would be read or overwritten. Within the Jb code, the choice is to make the derived type pointers specific inside the computational code everywhere from scratch (or alternatively, to use only dummy arguments in low level routines). The same code adaptations will need to be done little by little in the model code for “fieldsets” in the future cycles.

MF also asked about what sort of logic would be suggested for deriving LAM objects with respect to global ones ? For instance, would a target Object-Oriented design of the IFS potentially allow to define in a single binary first a global state X1, then a global B1, compute B1.X1, then transform X1 into a LAM state X2, instantiate a LAM B (B2) and compute B2.X2 ?
Mike says 'yes', this should be possible (even when using templates in C++).
Claude says that this type of analysis and brainstorming will need to be further investigated.

Another comment by MF was that so far, the most straightforward way of encapsulating various elements in Fortran was by merging LAM and global code into common modules (and sometimes common derived types and module procedures). See for instance what was done in the past with “control_vectors” and “spectral_fields”, or today with the LAM parameters inside JB_STRUCT. Mike says this may not be mandatory, but it eventually depends on what one wants to have in the end.

Decision about content in CY38:

- Mike's Jb OK
- Deborah will further add encapsulated GOMs (not yet included in the pre-cycle CY37R3 version sent to MF on Tuesday)
- Parameter specifications for known-to-be-global variables
- this “technical” CY37R3 will be sent to MF by end of next week (week 17)
- additional scientific changes will follow (RTTOV-10, some IFS physics changes) and the full CY37R3 would be sent to MF in early June
- Thibaut expects that he can send the global+LAM-phased Jb code back to Mike by end of May (NB: no specific visit by Thibaut to EC is planned so far)

2. <u>Fortran coding standards updated new documentation </u>:

Mike and Olivier have been having some further exchanges in early April. Olivier has issued a new version (April 14) for which EC still has a few comments. Those have been discussed and, apart from a few issues where further discussions or a later evaluation will be needed, a consolidated version of the Fortran coding rules should be ready soon. Here is the list of agreed changes:

- change order of Sections and titles: conception => preliminary design; control of the code => detailed design; Order (using the present numbering): 1, 4, 2, 3
- add an example of a file header in Appendix
- no naming convention required for derived types

- Ryad's extended Coding norm document will be kept as a background reference (and detailed norm checker compatibility)
- no overwrite of namelist variables: this item requires more discussion. We all agree that there are overwritten namelist variables in IFS/Arpège, and it is actually quite difficult to avoid this presently in 4D-VAR for instance (eg. Physics parameters overwritten below CVA1). Mike thinks that the norm should anyhow tell what the code should look like, while not reflecting exactly how it looks like now. Ryad proposes that one should list which type of namelist parameters should not be overwritten.
- allocate/deallocate: huge arrays should be ALLOCATABLE; small or low-level arrays should be automatic ones, for performance and Open-MP compliance. Automatic arrays should be the preferred way of declaring arrays.
- Prohibit "cut & paste" of code: OK, but the rule will be written out less strongly in the coding norm documentation
- remove item 4.6: OK, no rules about lengths of loops (too much dependent on both machine aspects and scientific constraints)
- ALLOCATEd arrays should be DEALLOCATEd, especially when they're used only in a restricted section of the execution (allocation and deallocation takes place in close-by routines)
- commented Fortran code should be avoided. It is however acceptable to have commented Fortran code as part of an explanatory section (explaining a new piece of code by reference to an older one, or a simpler version of the same computation, or a LAM versus global code section)
- continuation lines: the ampersand & is mandatory, and an indentation should be done but no fix rule for how many blanks
- NULOUT only should receive deterministic printouts. Timing information should be re-directed into a specific output file ("timing.info" ??). this will ease the reading of diff files for output listings
- long CALL sequences should be broken at the same places than the continuation marks in the SUBROUTINE sequence, in order to ease the reading of long lists of dummy arguments
- rule 3.22 is misspelled: the rule should be about how to compare variables (exact equality assessment or bounded interval). We propose that explicitly set variables (parameters, constants, namelist variables, …) should always be exactly compared (== or /=, .EQ. Or .NE.) while evaluated variables should be tested against a reference using a typical threshold (about 100 times machine precision for instance).
- Rule 1.5: add a more explicit example (case of newly created module). The file name and the module name must be the same – i.e. the file must be called eint_mod.F90 and it should contain MODULE EINT_MOD (or both without _mod and _MOD).

<u>3. AOB</u>:

question by Ryad: should modules always be in the "arp/module" directory ? Do our present compilation systems support modules placed in various directories ? Deborah says she needs to check for IFS' compilation system, which might not be able to solve dependencies in this more general context. All agree that the present logic (all modules in arp/module) will have to be changed with the arrival of more and more encapsulation (modules). MF already does have some relocated modules in its pre-cycle CY37T1 => Deborah will check and tell what we should do with these .