# SATWIN: Jules model and data assimilation

## Work summary and notes

1. Requirements
2. Main design points
3. Running the system
4. Engineering changes to Jules codebase
5. Engineering changes to DA codebase
   * Rules-Based Comparison
6. Notes from doing the work
7. TODO List
8. Outstanding Issues & Requirements
9. Setting up NetBeans

## Requirements

|  |  |  |
| --- | --- | --- |
| 1 | Optionally enable multiple instances of Jules model | ✓ |
| 2 | If more than one instance of Jules, then enable particle filter (one or more) | ✓ |
| 3 | Specify number of instances of each of Jules and particle filter | ✓ |
| 4 | Specify host and processing details | ✓ |
| 5 | Use three state vectors: smcl, tstar\_tile and t\_soil | ✓ |
|  | - build in expansion for more state vectors | not solved yet |
| 6 | Allow different versions of initial conditions | ✓ |
|  | [- different versions of driver files] | - |
|  | - different versions of nml files | ✓ |
|  | - drive.nml to specify ./drive/\*.dat input data file | ✓ |
| 7 | Read observation data into particle filter | delegate to DA code |
|  | - use observation data only after initial spinup | ✓ and delegate to DA code |
|  | - only use particle filter after initial spinup | ✓ |
|  | - support time-ordered ascii file | delegate to DA code |
|  | - build in capability for NetCDF file | delegate to DA code |
|  | - all particle filters will use only one set of observation data | delegate to DA code |
| 8 | Specify output directory for each Jules particle's data files | ✓ |
| 9 | Specify input directory of each Jules particle's nml files | ✓ |

## Main design points

Various multiple dependencies have been removed and the top layer of controlling code made more modular. This is to allow current and future requirements for greater flexibility in running the DA code under different model circumstances. The ‘main’ program in pf\_couple.f90 has been replaced by a much simplified ‘main’ in empire.F90 with all functionality delegated to an interfacing layer in empire\_mod.F90. This carries out the bulk of the MPI work, thus removing it from the central control, dispensing with the lonely bit in comms.f90, and conferring much greater clarity.

It was an intention to wrap all MPI calls so that no other code dealt directly with the MPI interface - a common standard approach which would enable easier alterations and enhancements, and remove the direct dependency altogether. However, the dinosaurian characteristics of F95 meant that this became to difficult to achieve to a tight deadline (although I have kept the changes attempted in another code branch). The main driver for this was so that all MPI\_RECV calls would be in one location thereby allowing the model to send a zero length state vector as an indicator that it had finished running. The filter code could then deal with this and it would negate the need to know the number of timesteps up front for those types of filters not needing it. With the MPI\_SEND/RECV calls embedded in various places, this is not practical and is too prone to error.

The main JULES\_trunk directory is a NetBeans project with the build commands set. You will need to edit them to reflect your own netcdf installaiont location (\*see below if you don’t know how to do this) and edit two Makefiles (they’re not my area of expertise so there may be a better way to do this):

JULES\_trunk\Makefile : set JULESDIR

JULES\_trunk\src\control\standalone\da\_filter\Makefile : set JULESDIR and RUNDIR

You then only need to press F11 to build the whole thing including the DA code. If you’re not using an IDE, then you’re making life hard for yourself! (NetBeans is free to download, it can be installed locally, then run from any terminal on any host)

To run the system, use the run\_mpi\_jules shell file which is in the directory where all the \*.nml files reside (currently harvard/). The run\_mpi\_jules script allows you to set the number of processes you want to run for each executable. The hostfile (in the same directory) allows you to specify hosts by name but is currently set to ‘localhost’. Changes to pf\_parameters.dat in this location will be over-written by the next build since the source file is copied across from “JULES\_trunk\src\control\standalone\da\_filter\controllers”. I should like to stop that happening if it’s been changed but that needs a little more Makefile know-how.

You will also need to ensure that your PATH environment variable is set to pick up the jules and empire executable files which are in the same place (JULES\_trunk) unless you wish to specify them exactly of course ☺

Running the system - changes already made:

pf\_parameters.dat:

use\_traj=.FALSE.

init='N'

and for ‘twin’ experiment

gen\_data=.TRUE.

together with modification of run\_mpi\_jules to contain only one model and one filter process

Currently, compile time controls are as follows:

* model\_params%state\_dim

is set via **mpi\_comms\_mod.F90**:empi\_send\_state\_info():states & the derived state\_dim\_arr

* model\_params%obs\_dim

is set via **mpi\_comms\_mod.F90**:empi\_send\_obs\_info():obs\_size

I have yet to implement these as runtime variables (which is what they most certainly should be!)

## Engineering changes to Jules codebase

* New source folder: JULES\_trunk\src\control\standalone\mpi\_model

*New files:*

Makefile

mpi\_comms\_mod.F90

* Existing model code in JULES\_trunk\src\control\standalone

*Changes:*

jules.f90

* Makefiles:

*New:*

JULES\_trunk\Makefile.comp.gfortran.mpi

*Changes:*

JULES\_trunk\Makefile : set MPI\_DEFINED and JULESDIR and PFPATH

new source directory, new target ‘pf’ dependent on variable \_MPI\_

## Engineering changes to DA codebase

Note that these have been kept to a bare minimum to improve modularity and encapsulation, and primarily to allow for different requirements of a model linked to the da code. The da source code has been placed in a sensible location: JULES\_trunk\src\control\standalone\da\_filter.

### Directory structure:

## 

### Rules-Based Comparison

11 files match exactly

filters\eakf\_analysis.f90

filters\etkf\_analysis.f90

filters\letkf\_analysis.f90\_working\_Serial

tests\alltests.f90

tests\test\_h.f90

tests\test\_hqhtr.f90

tests\test\_q.f90

tests\test\_r.f90

utils\histogram.f90

utils\quicksort.f90

utils\random\_d.f90

14 files don’t match

data\Rdata.f90

filters\enkf\_specific.f90

filters\equivalent\_weights\_step.f90

filters\letkf\_analysis.f90

filters\proposal\_filter.f90

filters\sir\_filter.f90

filters\stochastic\_model.f90

operations\gen\_rand.f90

operations\operator\_wrappers.f90

operations\perturb\_particle.f90

operations\resample.f90

utils\data\_io.f90

utils\diagnostics.f90

utils\genQ.f90

Changes are:

* remove dependency on sizes.f90 (which had only 2 variables!) and instead use a new type, model\_params, in pf\_control.f90. *Reason*: clearer design, encapsulation, dependency removed.
* remove comms.f90 which didn’t really do very much! Replaced with empire\_mod.F90. Dependencies in files changed accordingly, no need to keep including mpif.h as it’s done once. *Reason*: clearer design, multiple low-level includes removed, encapsulation of more functionality.
* formatting - gen\_rand.f90 has no other changes. *Reason*: unreadable!
* renaming variables - resample.f90: requests->open\_requests, and statuses->test\_statuses. *Reason*: clarity, use of similare names elsewhere.
* error message printouts prepended with digit - data\_io.f90. *Reason*: confusing to have the same message from several places.

Files removed:

pf\_couple.f90

sizes.f90

comms.f90

New files:

empire.F90

empire\_mod.F90

Moved files:

model\_specific.f90

Makefile

## Notes from doing the work

**Setup**

Install NetBeans

Import Jules code into NetBeans project

Set up linker and flags

Customise project properties:

Set up ‘make’ commands

Set up ‘run’ command

Use conditional compilation of MPI features within model code

Set debugger command: /usr/bin/gdb

**Multiple instances**

Link and compile with MPI library

Set up hosts file to specify process allocation (located in data/run folder)

Set up compiler flags to optionally include MPI code

Create new makefile for MPI configuration

Create MPI communications module ---🡪 see below

Create and test ‘run’ command using 2 models only with internal send/recv pair

Specify output file name stem -> each Jules model writes logging info. to own file

Change working directory of model process to sub-folder (unique set of startup and output files)

Using pf\_minimal example code:

* **Particle filter**

Create new executable code to implement particle filter

Modularise behaviour

Weird state variable sizes, and timestep information

Unpack state variables

Pack state variables

* **MPI communications**

New functions: initialise, test, do-work and tear-down

Create master process and several Jules sub-processes

Create particle filter processe(s)

Assign Jules processes to particle filters

Dynamically assign arrays for the state vectors - pass their sizes to the particle filter(s)

Decouple timestep knowledge: use non-blocking probe for the next data message within timeout

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

New DA baseline to integrate: arrghhh!

## TODO LIST

|  |  |
| --- | --- |
| **DONE** | **TASK** |
| YES | re-integrate new DA code: rework the lengthy and complex main program as before |
| YES | remove dependency on comms.f90 everywhere as it’s actually not encapsulating much at all! |
| YES | make sure all ‘comms’ uses are removed, replaced with empire\_mod |
| YES | remove pf\_couple.f90 completely too - it’s a mess |
| YES | extra split in the filter process |
| YES | placeholder for model running partial geo regions using MPI (add extra split) |
| YES | send state vector at end of spin-up only - find the Jules flag - using ‘is\_spinup’ flag |
| YES | how does da code know about state vector number, size and dimensions? - hard coded?! - replace with send/recv pair with state vector information |
| YES | only do mpi/da at start of model run - using ‘is\_spinup’ flag |
| NO (F95 doesn’t support dynamic typing) | pull send/recv out of da codes and wrap so that this is only in one place - 7 files only. This would allow one place for checking whether the model run had finished rather than having to know the number of timesteps a priori. |
| NO (ditto) | pull out allgather calls in filters too |
| YES | makefile for DA stuff - disable testing as won’t build, call from main Makefile, new source & dependencies, put .exe in JULES\_trunk directory |
| YES | ensure DA process can still access ‘parameters.dat’: copy rule in Makefile |
| YES | my filenames have changed too, also new dependencies for Makefile and different directories |
| YES | runtime name now ‘empire.exe’ so change run.sh to launch it |
| YES | how is model-filter allocation now done? allreduce in filter code has been removed: see http://www.met.reading.ac.uk/~darc/empire/Lorenz63\_empire.f90 |
| YES | use of particle number which is model\_rank+1 throughout da code for indexing arrays, however, then needs -1 to do MPI communications - should have two areas to store the info. as it’s different. |
| YES | check for instances of pf%type and change to pf%filter - ‘type’ is a reserved word |
| YES | **ensure new stuff from new code included:** random seed |
| YES | **ditto:** gbldisp and gblcount in comms:initialise\_mpi / ALLGATHER in comms:initialise\_mpi |
| YES | **ditto:** some weird things in model\_specific:configure\_model - now moved to hookup and do\_work functions in empire\_mod. Design improvement: separation of tasks into discrete functions. |
|  | is it possible to work out timesteps a priori? if not, put non-blocking probe and send timeout information, send a switch to determine which, OR send with size=0 |
|  | Create runtime model folders as required, copy template \*.nml files and create sub-directories if none is already present. |
| PARTIAL | Send details of observation space to da process ✓ensure it is tested prior to filter work ✓, read from nml file in model setup 🗶. |

## Outstanding Issues & Requirements

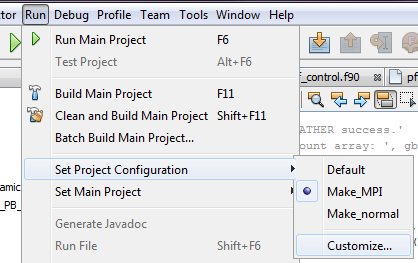
DA filter timestep loops assume:

1. observation stop coincides with model termination. If not, each MPI\_RECV will need to handle a zero-length state vector to indicate that the filtering should stop.
2. observation inputs are at regular intervals such that pf%time\_bwn\_obs remains constant.
3. observations start when model spinup stops - there would have to be a method to recognise timestamps on observation input to ensure it matches model -> implies a check **before** starting any DA filtering irrespective of spinup completion time.

## Setting up NetBeans

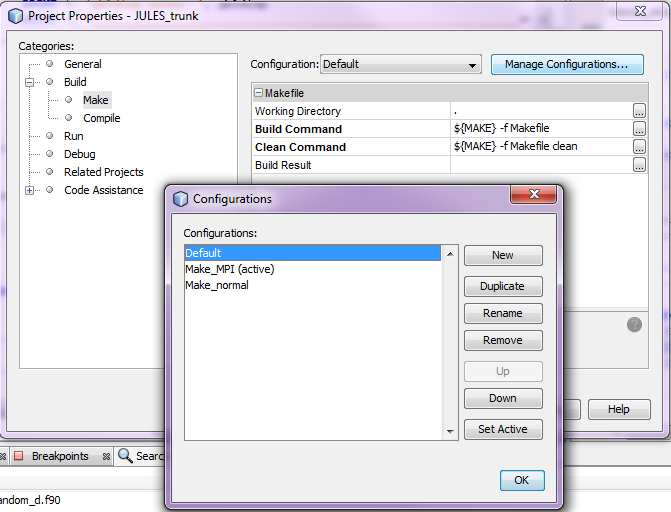
To set the build commands, create a new project configuration:

1. Run -> Set Project Configuration -> Customize…



2. In resulting dialogue, **either**

a) click on Manage Configurations… if you want to create a duplicate using another as a template:



**or**

b) choose the Make\_MPI configuration from the dropdown list and edit the commands. Ensure you have the Build->Make category selected in the left panel. Clicking the ellipses will open a text dialogue for you to edit the contents of the command.

