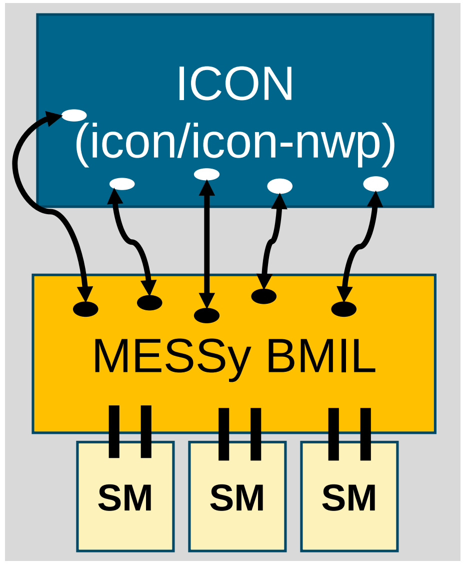
MESSy as a ComIn plugin

Kerstin Hartung, Bastian Kern (DLR-PA)

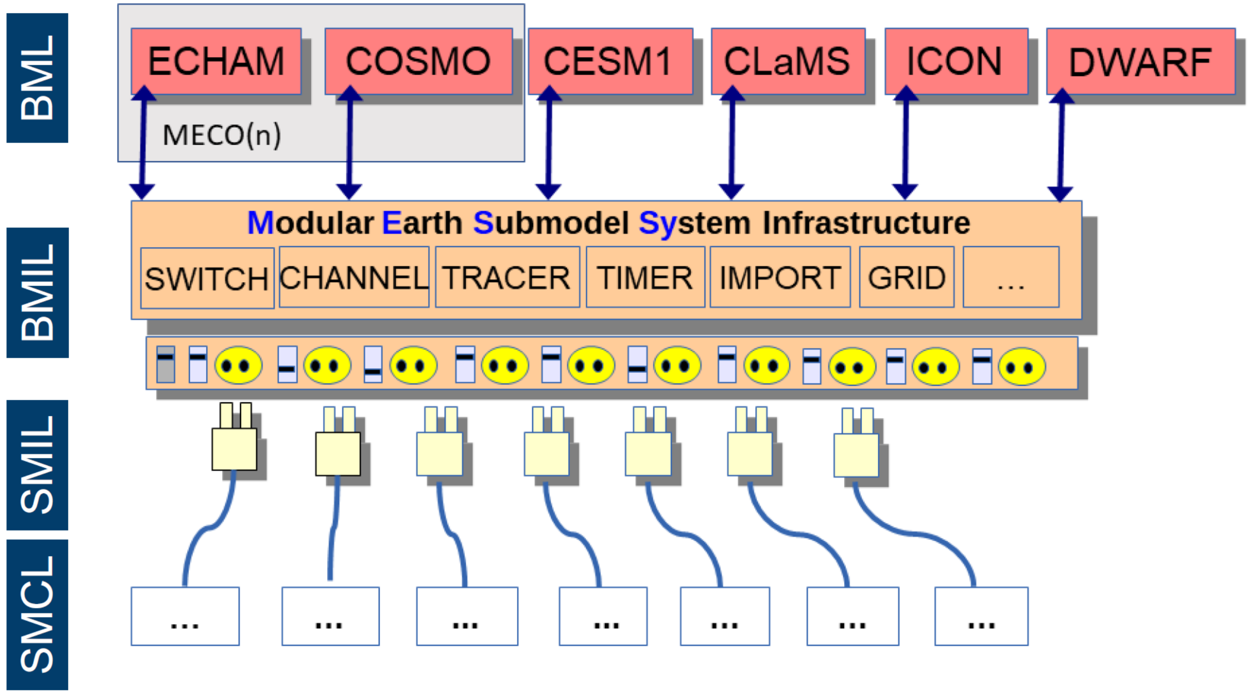
Wilton Loch[1], Astrid Kerkweg[2], Patrick Jöckel[3] (1: DKRZ, 2: FZJ IEK-8, 3: DLR)

What is ICON/MESSy?

Current status

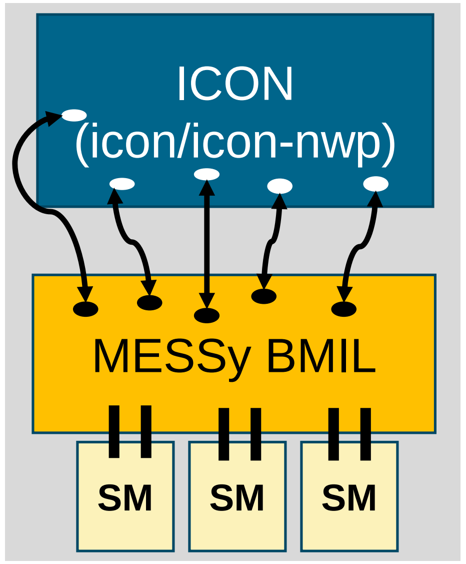


MESSy code structure

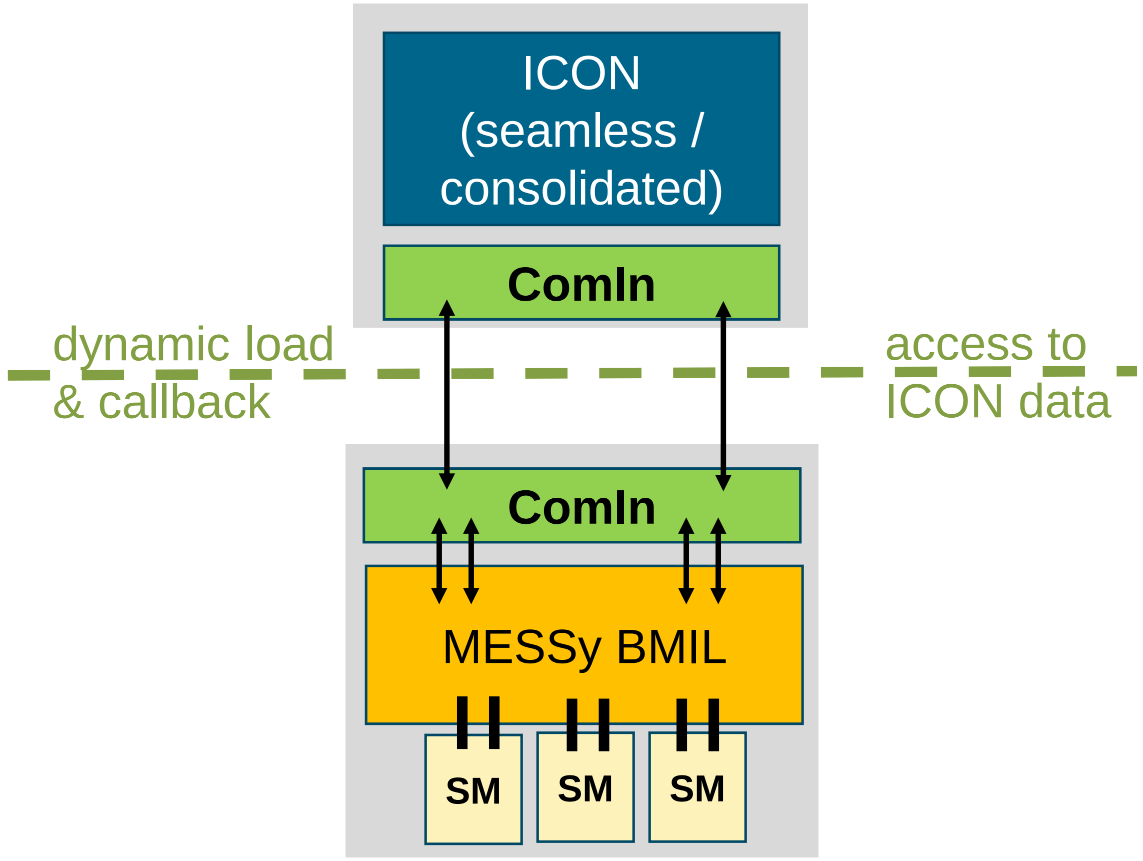


Why do we need ComIn? And why is MESSy not obsolete?

Current status



Goal: reduced effort to keep up-to-date with ICON developments



natESM sprint: Couple MESSy to ICON via the ICON Community Interface (ComIn)

- 07/2023 – 01/2024
- implement ComIn in MESSy, advance towards a working setup
- co-existence of original and new implementation during development
- document steps and challenges for other plugins

- open call for proposals from model-development groups across Germany
- sprints are focused on technical objectives, flexible, tailored to research goals and timelines
- up to 6 months, in-depth partnership between applicant and Research Software Engineers (RSE)

Implementation approach

- iterative implementation updates
- testing functionality of intermediate steps
- workarounds/updates if either MESSy or ComIn do not support functionality at the moment
- direct feedback to ComIn development
- shaping future ComIn version: short term and long term development goals

- ComIn is a lightweight interface
- ComIn provides grid and decomposition information, variable meta-data, …
- ComIn provides a MPI communicator but no communication patterns

Recipe to prepare software as ComIn plugin

Preparation:

Gather potential issues, for example in the data fields the plugin is expecting, the order of plugin routines relative to the ICON control flow.

In MESSy, for example:

- workaround to not require hybrid vertical coordinate from ICON
- MPI functionality, previously accessed through ICON

Note: some points of the recipe are only relevant if you have already integrated your code to ICON.

Recipe to prepare software as ComIn plugin

1. Prepare code as **shared library** (unless using Python, where everything is easier).

Results from the natESM sprint

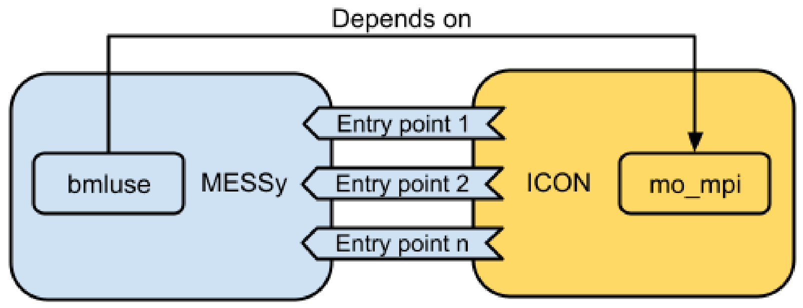


Diagram by Wilton Loch

Recipe to prepare software as ComIn plugin

1. Prepare code as shared library (unless using Python, where everything is easier).
2. Decide which **entry points** will be accessed and which **data** need to be **added to ICON** (e.g. regular data fields and tracers). From this point onwards **descriptive data** can be accessed. This already defines the content of the **primary constructor**.

In MESSy, so far, only the entry points of the initialization phase are considered.

Results from the natESM sprint

```
SUBROUTINE messy_comin_setup()

   CALL init_icon_get_mpi()

   CALL comin_callback_register(EP_SECONDARY_CONSTRUCTOR,
           messy_comin_constructor, ierr)
   CALL comin_callback_register(EP_ATM_INIT_FINALIZE,
           messy_comin_atm_finalization, ierr)

   CALL messy_setup()
   CALL messy_initialize
   CALL messy_new_tracer

   CALL messy_request_tracers

END SUBROUTINE messy_comin_setup
```

some workarounds

register callbacks

initialize MESSy and MESSy submodels

gather info on new tracers, use descriptive data

prepare MESSy tracer metadata and call *comin_var_request_add*

Recipe to prepare software as ComIn plugin

1. Prepare code as shared library (unless using Python, where everything is easier).
2. Decide which entry points will be accessed and which data need to be added to ICON (mainly tracers). From this point onwards descriptive data can be accessed. This already defines the content of the primary constructor.
3. Decide which ICON data the plugin should access. In case of a larger/ more complex plugin: decide how to associate existing data structures with those provided by ComIn. This is a main component of the **secondary constructor.**

Results from the natESM sprint

```
SUBROUTINE messy_comin_constructor()

  CALL messy_get_tracer_metadata_comin

  CALL messy_init_memory

END SUBROUTINE messy_comin_constructor
```

receive tracer metadata

receive pointers to ICON variables and tracers, initialize submodel memory

```
SUBROUTINE messy_comin_atm_finalization()

    CALL messy_init_coupling

    CALL messy_read_restart
    CALL messy_init_tracer

END SUBROUTINE messy_comin_atm_finalization
```

incomplete because MPI not fully set up

read restart and initialize tracer fields

Recipe to prepare software as ComIn plugin

1. Prepare code as shared library (unless using Python, where everything is easier).
2. Decide which entry points will be accessed and which data need to be added to ICON (mainly tracers). From this point onwards descriptive data can be accessed. This already defines the content of the primary constructor.
3. Decide which ICON data the plugin should access. In case of a larger/ more complex plugin: decide how to associate existing data structures with those provided by ComIn. This is a main component of the secondary constructor.
4. Associate routines registered to entry points with plugin routines.
5. Update ICON runsript (*comin_nml* section) to apply plugin (prepared as shared library).

Results from the natESM sprint

Valuable **feedback from first complex plugin** for (early) ComIn development, e.g.:

- expanded metadata and access/set routines
- all cell centre fields shared via ComIn
- convenience function for time steps
- some additional descriptive data

→Simplified interaction with ICON AND MESSy is not obsolete!

- flexible addition of variable fields and tracers
- data fields from other plugins can also easily be accessed, e.g. YAC for I/O
- some auxiliary routines (and descriptive data) named more intuitively than in ICON

Outlook and open questions

final steps of initialization

- set up MPI with YAXT

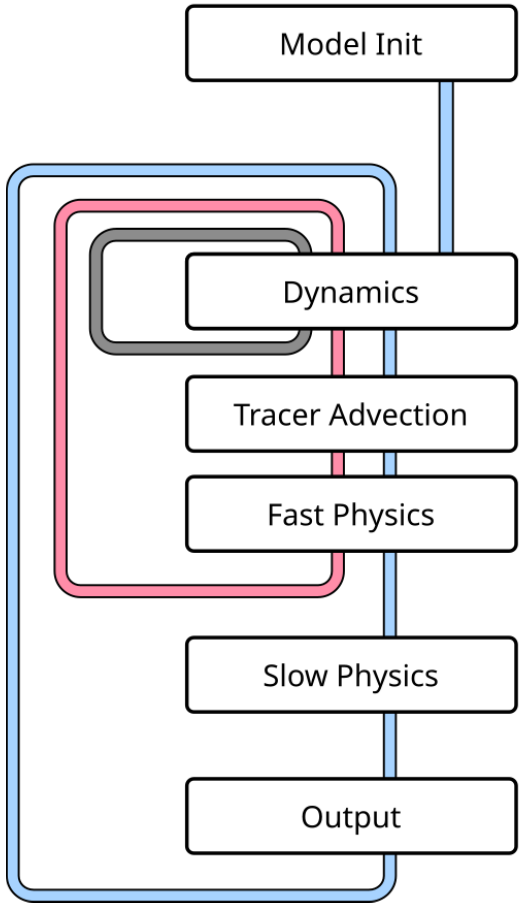
using ComIn during time loop

- add calls to current entry points in MESSy
- some workarounds, e.g. for access to currently (sub)routine-local fields, masking regions for parameterizations

recommendations for community

evaluation of implementation

2nd natESM sprint on MESSy-ComIn (queued)



Prill et al., ICON tutorial

Impressum

Topic:	**MESSy as a ComIn plugin**
Date:	2024-07-17
Author:	Kerstin Hartung and Bastian Kern
Institute:	DLR-PA-ESM
Image credits:	All images „DLR (CC BY-NC-ND 3.0)“ unless otherwise stated