Hierarchical Testing of ESMF/NUOPC Components

Ufuk Turuncoglu
(turuncu@ucar.edu)

14 Jun 2023, 28th Annual CESM Workshop

This work is supported by the NOAA Joint Technology Transfer Initiative (JTTI)
NA21OAR4590167: Advancing Land Modeling Infrastructure in the UFS for Hierarchical Model Development

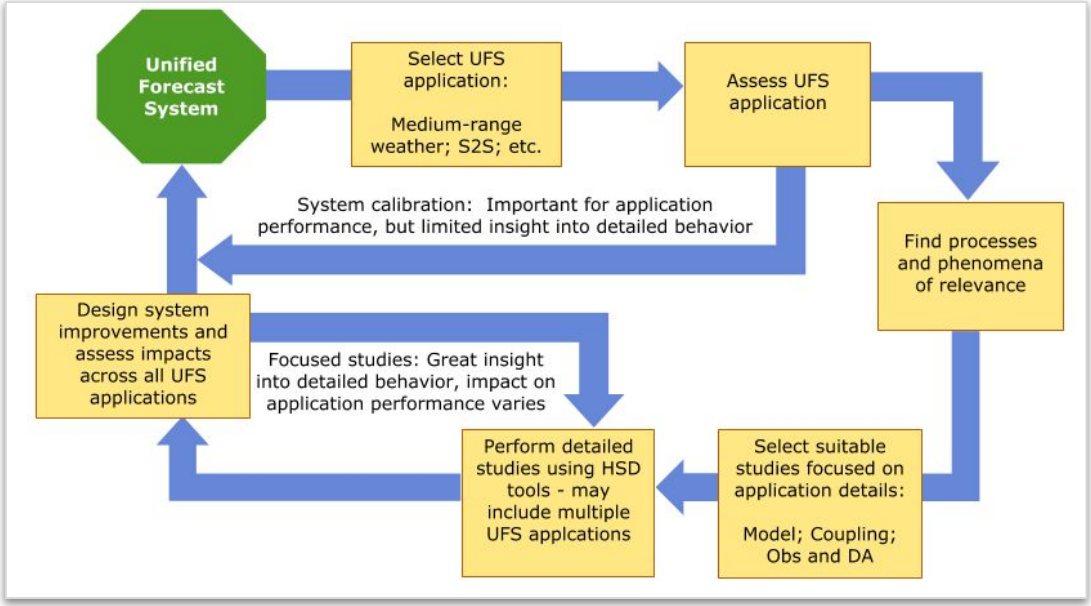
Outline

- Hierarchical System Development & Testing
- UFS as an example (in terms of its complexity)
- What is the limitations and how it can be improved?
- Initial attempt
  - Testing ESMF/NUOPC cap in an isolated environment
    - Noah-MP land component example

Hierarchical System Development & Testing

- Developing multi-component earth system model is a challenging task and requires extensive testing of the application
- Hierarchical system development (HSD) refers to the ability to engage in development and testing at multiple levels of complex prediction software such as UFS (Ek et al, 2019)

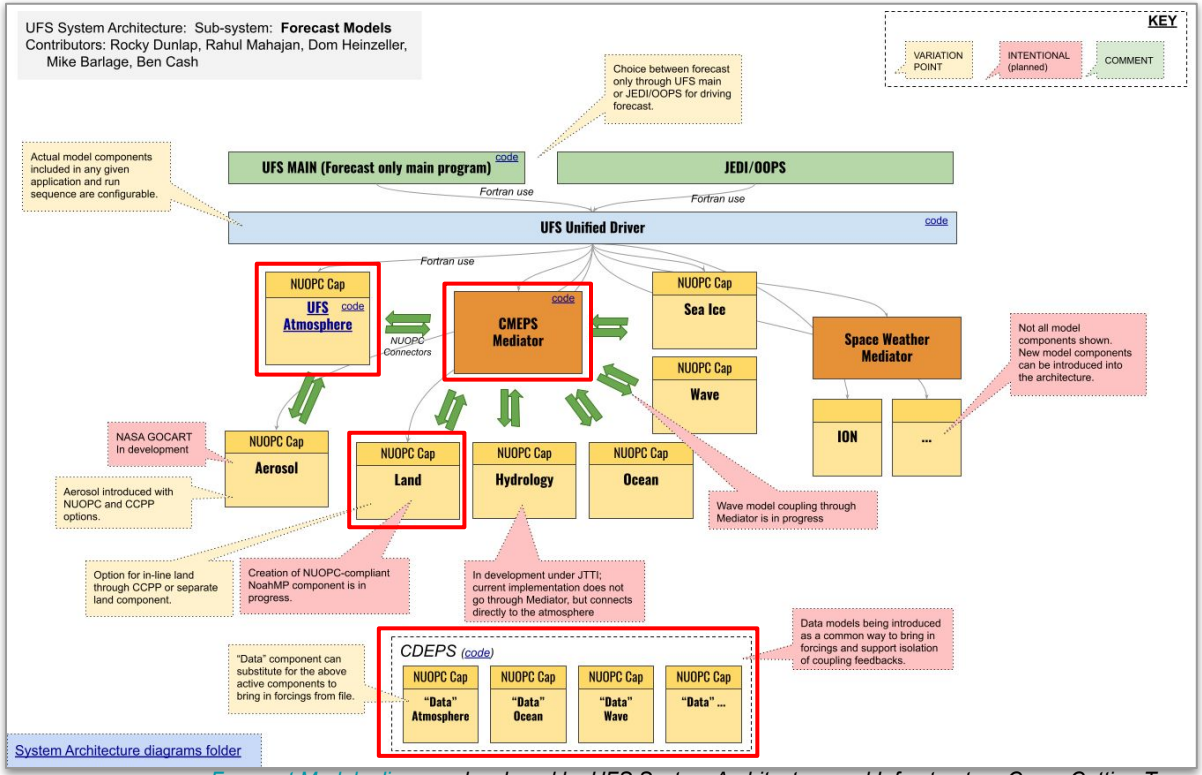
HSD in the UFS Development Process
(Figure adapted from Christian Jakob, BAMS 2010)



- The HSD approach mainly aims to test the entire system from very simple configurations (single-column, aqua planet) to more complex ones (fully-coupled)

Example: NOAA's Unified Forecast System (UFS)

- The UFS is a community-based, coupled, comprehensive Earth modeling system.



Forecast Models diagram developed by UFS System Architecture and Infrastructure Cross-Cutting Team.

Unified modeling requires a modular and flexible approach to building complex coupled systems.

Uses ESMF library for coupling.

Example UFS applications:

UFS GFS:
FV3ATM-MOM6-CICE6-WW3-GOCART-CMEPS

UFS Hurricane (HAFS):
FV3ATM-HYCOM-WW3-CMEPS

UFS RRFS:
FV3ATM

Example: UFS Testing System

- **Automated**:
  - CI/CD (Continuous Integration/Continuous Development) testing on the cloud
  - AutoRT on NOAA R&D platforms (i.e. Cheyenne, Orion, Hera). This is also integrated with GitHub CI/CD system and run on every PR.
- **Manual**:
    - Same RTs used by AutoRT can be triggered manually. This is the first step that needs to be done before having PR.
    - opnReqTests: reproducibility (std,thr,mpi,dcp,rst,bit,dbg,fhz)
- The UFS RT system includes: **(1)** standalone model tests, **(2)** configurations coupled with CDEPS data components and, **(3)** fully coupled configurations. The total number of tests are **234** and it keeps growing along with new applications and configurations.

More information: https://ufs-weather-model.readthedocs.io/en/latest/AutomatedTesting.html

Limitations

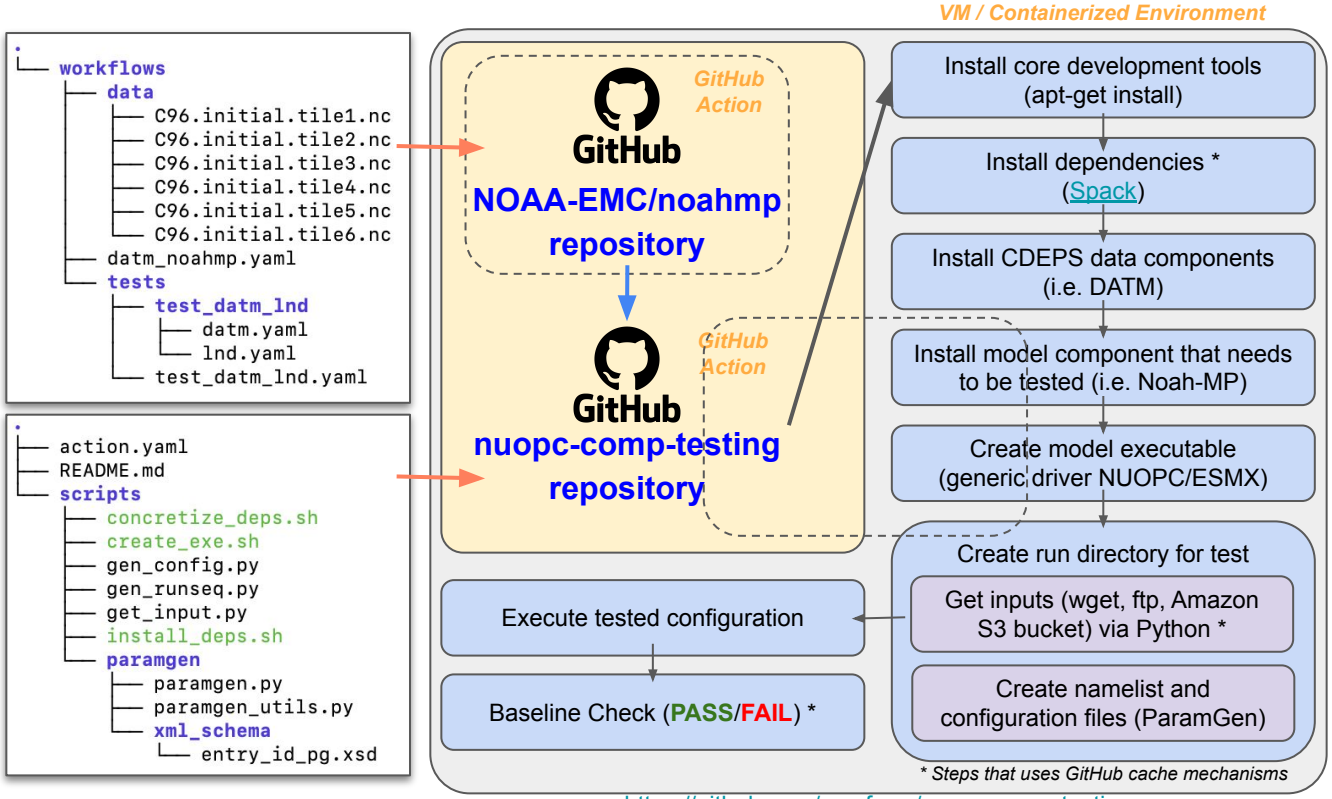
- In general, users/developers need to checkout entire application to perform testing
  - It is hard if user/developers have no access to supported platforms
  - It is not trivial to port the application to a custom platform (requires expertise)
- There is no easy way to test the standalone model component and its coupling interface in an isolated environment regularly
  - Requires lots of manual interaction (building dependencies, staging input files, creating namelist files etc.)
  - Lots of complexities: application specific workflow system, driver, mediator etc.
- Existing testing system/s do not cover all the possibilities. Just small portion of it.
- There is no any convention/standard in terms of testing systems (CIME, UFS RTs etc.)

Solution for Perfect World?

- Each individual model component exposes its tests and configurations using standardized way independent from used modeling system/application
  - The configurations range from very simple to complex ones
  - Published along with the source code (probably through the GitHub)
  - The top-level application inherits those tests and blend them together in a specified way to create more complex tests
  - Enable to use same unit tests (provided by model) across different modeling systems.
  - Allow to integrate simple configurations with CD/CI
- Requires definition of the rules/standards to represent individual tests, their requirements and the way of blending them together

Testing ESMF/NUOPC Interface?

- New hierarchical testing capabilities (example from **Noah-MP** land component):



Global NoahMP (C96) forced by GSWP3 (CDEPS DATM)

https://github.com/esmf-org/nuopc-comp-testing

Handling Dependencies

- Spack package manager is used to install dependencies of the specified configuration

```
dependencies: |
  esmf@${{ matrix.esmf }}+external-parallelio
```

```
▼ Create spack.yaml
  spack:
    concretizer:
      targets:
        granularity: generic
        host_compatible: false
      unify: when_possible
    specs:
    - esmf@8.5.0b10+external-parallelio %gcc@11.3.0 target=x86_64
    packages:
      all:
        target: ['x86_64']
        compiler: [gcc@11.3.0]
    view: /home/runner/.spack-ci/view
    config:
      source_cache: /home/runner/.spack-ci/source_cache
      misc_cache: /home/runner/.spack-ci/misc_cache
      test_cache: /home/runner/.spack-ci/test_cache
      install_tree:
        root: /home/runner/.spack-ci/opt
      install_missing_compilers: true
```

- The dependencies are defined as a input argument to nuopc-comp-testing composite action
- The composite action creates spack.yaml based on user request and install all dependencies such as ESMF, MPI, FMS etc.
- The compiler is installed using apt package manager but it is possible to install using Spack too.

Creating Executable

- ESMF/NUOPC provides generic driver layer to run the test configuration
- **E**arth **S**ystem **M**odeling e**X**ecutable (ESMX)

New Coupling Application Layer:

- provides an executable
- provides a NUOPC-based coupled system driver
- uses CMake to embed components into a system - YAML based specification

- Improved version of ESMX layer will be available in the next public release (8.5.0)

Motivations:

- Accelerate development of new NUOPC-based systems.
- Introduce mechanism for testing model components and coupling systems.
- Reduce maintenance cost for established NUOPC-based systems.
- Standardize processes for NUOPC-based systems. (configuration files, build procedures, etc.)
- Accelerate new feature roll-out for NUOPC/ESMF.

Adapted from Dan Rosen's March 30th, 2023 UFS presentation

Handling Inputs

- Generic Python interface to retrieve input (wget, ftp, s3 via Python or s3 CLI)

wget from GitHub

```
      input:
        field_table:
          protocol: wget
          end_point: 'https://raw.githubusercontent.com'
          files:
            #- /ufs-community/ufs-weather-model/develop/tests/parm/fd_nems.yaml
            - /uturuncoglu/ufs-weather-model/feature/noahmp/tests/parm/fd_nems.yaml
          force: True
```

wget from SVN

```
  input:
    forcing:
      protocol: wget
      end_point: 'https://svn-ccsm-inputdata.cgd.ucar.edu'
      files:
        - /trunk/inputdata/atm/datm7/atm_forcing.datm7.GSWP3.0.5d.v1.c170516/Precip/clmforc.GSWP3.c2011.0.5x0.5.Prec.1999-12.nc
        - /trunk/inputdata/atm/datm7/atm_forcing.datm7.GSWP3.0.5d.v1.c170516/Precip/clmforc.GSWP3.c2011.0.5x0.5.Prec.2000-01.nc
```

s3 from AWS

```
  input:
    fixed:
      protocol: s3
      end_point: noaa-ufs-regtests-pds
      files:
        - input-data-20221101/FV3_fix_tiled/C96/C96.maximum_snow_albedo.tile1.nc
        - input-data-20221101/FV3_fix_tiled/C96/C96.maximum_snow_albedo.tile2.nc
```

Handling Namelist Files

- The model configuration is defined by the set of YAML files
- The basic configuration to test coupling interface is data component forced model
- The namelist handling is supported by slightly modified version of ParamGen (Thanks to Alper!)

```
68    config:
69      nuopc:
70        name: esmxRun.config
71        content:
72          no_group:
73            LND_model:
74              values: noahmp
75            LND_petlist:
76              values: 0-5
          LND_attributes:
            Verbosity:
              values: 0
            Diagnostic:
              values: 0
```

```
72      nuopc2:
73        name: datm.streams
74        content:
75          no_group:
76            stream_info:
77              values:
78                - CLMGSWP3v1.Solar01
79                - CLMGSWP3v1.Precip02
80                - CLMGSWP3v1.TPQW03
81                - topo.observed04
```

```
132     nml:
133       name: input.nml
134       content:
135         fms_nml:
136           clock_grain:
137             values: "'ROUTINE'"
138           clock_flags:
139             values: "'NONE'"
140           domains_stack_size:
141             values: 5000000
142           stack_size:
143             values: 0
```

Summary

- NUOPC component testing action provides standardized way to run low-res configuration of the ESMF/NUOPC complaint model component in an isolated environment through the GitHub Action
  - The current version is 1.1 and the documentation is in here.
  - Multiple tests can be defined
    - Same configuration forced by different datasets
    - Testing ESMF/NUOPC cap against different version of ESMF library
- Enables testing component through the development in an automatized way
- Provides set of standard/conventions to define tests
  - Tests live along with the code and tested against the new developments