Unifying Workflows for UFS Applications

Christina Holt, Fredrick Gabelmann, Brian Weir, Venita Hagerty, Emily Carpenter, Janet Derrico
## Acknowledgements

### Funding
- UFS R2O
- NOAA EMC, in kind
- CIRES, in kind
- NOAA GSL, in kind
- GMU, in kind
- EPIC Program
- JTTI
- SENA

### Contributors
- Benjamin Cash, GMU
- Rahul Mahajan, NOAA EMC
- Arun Chawla, NOAA EMC
- *Fredrick Gabelmann, Element 84/EPIC
- Julie Prestopnik, NCAR/DTC
- Ryan Long, Redline/NOAA EMC
- *Emily Carpenter, CIRES/NOAA GSL
- *Naureen Bharwani, CIRES/NOAA GSL
- *Brian Weir, Raytheon/EPIC
- Janet Derrico, University of Colorado
- Venita Hagerty, CIRA/NOAA GSL
- *Paul Madden, CIRES/NOAA GSL
- and many others from other EPIC Teams

### Stakeholder Institutions
- NOAA EMC
- GMU
- CIRES
- NOAA GSL
- NCAR RAL
- DTC
- JCSDA
- NOAA PSL

*PI9 team
What is UFS?

The Unified Forecast System (UFS) is a community-based, coupled, comprehensive Earth modeling system. The UFS numerical applications span local to global domains and predictive time scales from sub-hourly analyses to seasonal predictions.
What is UFS?

... applications share agreed-upon numerical forecast system elements, including Earth-system model components (e.g. atmosphere, ocean, sea ice, land, chemistry, etc.), observation processing, pre-processing, data assimilation, forward forecasting, ensemble and probabilistic processing, and post-processing...[and] infrastructure such as model coupling tools and workflow software.

We all run the same components, configured in different ways
The Big 3+ Apps

- MRW Medium-Range Weather
- SRW Short-Range Weather
- HAFS Hurricane Application
- Land DA
- Total Coastal Water
- Exascale Prototypes
- RnR Reforecast & Reanalysis
- Atmospheric rivers
- S2S Subseasonal-to-Seasonal
Operational standards dictate a layered structure

- Configuration Layer
- Workflow Manager
- Batch System Job Card
- Run Scripts
- Utilities
- Compiled Executables
Build a modular, portable, robust framework for running the Unified Forecast System that supports research and operations

Take a services-based approach to ensure extensibility and usability by all of the UFS Applications

Develop a user-interface that the UFS Community can be comfortable using
Obtain more funding at a variety of institutions to join the UW Team

Ensure buy-in from NOAA leadership in charge of each App and Operations so that UW software can make it through the research funnel

Regularly release tools for inclusion in Apps to start iterative design process early

Apply software best practices to ensure a robust, well-tested, easy-to-use toolbox and framework for UFS workflows
Unification Strategy

**Short Term (Upcoming PIs)**
Develop a set of generic, **standalone tools** to address common high-maintenance problems.

Propose the changes necessary in the relevant UFS Apps and components.

**Medium Term (Upcoming year)**
Replace the configuration layer of the existing Apps with a **framework** that unifies them around a **service-oriented architecture (SOA)** to achieve a “plug and play” feel for a given experiment.

Requires developing necessary interfaces to the existing component drivers (e.g., bash run scripts) and existing workflow managers (e.g., ecFlow, Cylc, Rocoto).

**Long Term (Next few years)**
Use the SOA framework as a facade and apply the **strangler pattern** to gradually replace and unify the underlying component drivers.

Unify gradually, iterate often, add value ASAP.
Prioritizing Unification

Common Tools

EPIC

HAFS

Ocean Driver

Wave Driver

Make Grid Driver

Rocoto XML
cFlow Definition

Forecast Driver

GSI Driver

Make ICS/LBCS

Drivers

EnKF Driver

FuncX

Workflow

Exascale

SENA

Common Drivers

Real-time, hourly cycled LAM with stationary domains

Real-time, 6 hourly cycled ocean-coupled LAM with moving nests

3DEnVar LAM Prototype with ensemble members run on distributed resources
Configuration Management Tools for the UFS Weather Model

- Currently there are ~10 different types of parameter files the model uses to generate a forecast.
- These tools allow all parameter files to be managed from a single YAML configuration file
  - Improves organization, readability, and understanding compared to bash variables
- The tools will allow the Apps to use the parameter files (e.g., namelist, model_configure, etc.) directly from the model regression tests
  - Decouples the weather model version from the workflow
  - Promotes increased compatibility between the workflow and a variety of model versions
  - Reduces manual, repeated code maintenance when updating to a new version of the model (Apps usually keep a copy of these files in their own repositories, which is not ideal)
- A Python-based approach opens new doors for configuration validation (planned for future releases)
- The basis for the Unified Workflow configuration system
ecFlow and Rocoto Interfaces

- Takes in a YAML configuration file defining workflow
- Processes the information such as
  - Task resource requirements
  - Dependencies and triggers
  - Run-time environment
- Writes out necessary workflow definition files
  - Rocoto XML
  - ecFlow Suite Definition
  - ecFlow Job Cards
  - Standalone wrappers
- Dynamic and automatic at the time of experiment creation
- Tool is uncoupled to the experiment definition
File Mover

- Moves files between various locations
  - Cloud block storage, local file systems, HPSS, URLs
- Alongside a database of known data stores, this tool will help users stage the data they need as part of any workflow
- Enables users to work with data on various platforms with the same interface.
  - Local copies can have very similar commands as remote copies from a URL, for example.
  - Reduces overhead for understanding Python syntax and caveats for equivalents to cp, sync, wget, aws cli, and others.
GitHub Repository
https://github.com/ufs-community/workflow-tools

GitHub Wiki
https://github.com/ufs-community/workflow-tools/wiki

GitHub Discussion
https://github.com/ufs-community/workflow-tools/discussions

Read the Docs
A Contributor’s Guide and User’s Guide
Questions? Suggestions?

Contact: Christina Holt, christina.holt@noaa.gov

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.
Backup Slides
A SOA Unified Framework

What is Service-oriented Architecture (SOA)?

A service is software component that provides some functional capability.

Services can communicate with each other and across platforms and languages.

Services can be reused in different systems.

Multiple services can be combined to perform complex tasks.

https://aws.amazon.com/what-is/service-oriented-architecture
SOA is an architectural approach that can be applied to the design and development of various types of software systems, including enterprise applications, distributed systems, and integrations between different systems.

SOA is a broader concept that encompasses the design principles, patterns, and practices for building modular, interoperable, and scalable software systems based on services.

SOA is exactly what we need for unification!

Not to be confused with SaaS – Software as a Service.
What are **services**?

Chunks of code that should:

- Be **independent** – changes to one service should not impact other services
- Be **fully interoperable** – it doesn’t matter which App is running it
- Be **loosely coupled** and **stateless**
- Be **responsible for one thing**
- Employ **standardized communication protocols**

https://aws.amazon.com/what-is/service-oriented-architecture
Strangler isn’t as bad as it sounds

https://www.redhat.com/architect/pros-and-cons-strangler-architecture-pattern
Software as a Service

SaaS is a cloud computing model where software applications are provided over the internet, and users access them through a web browser.

With SaaS, users don't need to install or manage software locally, as the applications are hosted and maintained by the service provider.

SaaS is NOT where we’re going with unification!
Unification Approach
Strangler isn’t as bad as it sounds

The pattern is named after the Strangler Fig plant, which grows around a host tree and gradually chokes it, eventually replacing it entirely.

The Strangler, or Strangler Fig Pattern is a software design pattern that involves gradually replacing an existing system with a new one, using the old one as a foundation.
Strangler isn’t as bad as it sounds

Pros

The end user interface is **delivered early** in the process

**Reduces risk** when modernizing monolithic systems

Does *not* require a complete system overhaul on Day 1

A service transition **could be rolled back** if something goes wrong

Provides the development team ample time to **iterate on the system implementations** – what works and what doesn’t.

All UFS Apps are *not* required to be on the same schedule

Cons

There may be **many interfaces** needed in the facade

Requires a *lot* of ongoing **attention to changes** occurring in both the original system and the facade

It’s **hard** to modularize components that are tightly coupled

https://www.redhat.com/architect/pros-and-cons-strangler-architecture-pattern
Configuration Subsystem

Responsibilities include:

- Gathering user provided parameters
- Managing default settings for all portions of the System
- Validating that settings are appropriate and compatible
- Creating and populating experiment directories with experiment-specific files and data
- Optionally, starting the workflow manager of choice.
### Component Drivers as Services

#### <<configuration>> Interface

+ parse_config()
+ validate()
+ create_experiment()
+ create_workflow()

#### Workflow

+ config_object: type = dict
+ validate()
+ create_XML()

#### Forecast Driver

+ Config Object: type = dict
+ requires()
+ run()
+ output()
+ resources()
+ job_card()
+ validate()
+ create_namelist()
Summary

Why:
Too many tightly coupled workflows for UFS

What:
Building a Unified Framework

How:
Taking a Strangler Approach

WHAT:
- Standalone Services
  &
- Service Oriented Architecture Pattern

HOW:
- Replace existing components gradually