Scientific Data Analysis Web Application

CSCI 4308 Senior Capstone

Seongmin Choi, Robert Crimi, Connor Guerrieri, Bo Han, Hannah Keller, and Hannah Thomas

Documentation

NCAR

University of Colorado Boulder

Project Overview

Overview

Over the past decade it has become increasingly clear that the Earth’s climate is changing rapidly. Governments and private institutions have progressively invested more resources into the study of the impacts of climate change, but the are many barriers that currently slow down the advancement of scientific understanding. Due to the variety of computer simulation models, data formats, and data analysis languages, one of the largest barriers facing the progression of scientific understanding is reproducibility, or the ability to reproduce and verify the results of another’s analysis. The goal of our project is to create a python web based application that not only allows users to easily build and run data analysis workflows, but do so without the need of specific programming and climate analysis knowledge.

Current Problem

The goal of the climate modeling community is to provide reliable, accurate, and credible information to those who make decisions on how or whether to respond to climate change. Two of the largest barriers currently slowing down the advancement of scientific understanding are the expertise needed to run analyses and the inability to effectively share and reproduce these analyses. Climate scientists interested in expanding their knowledge not only need to be experts in their scientific field, but also in the installation and configuration of software, translating data formats, and variety of analysis tools. Impact users, decision makers outside the climatology field that are invested in the impacts of climate change, such as city planners, utility managers, or park rangers, need easy access to climate data and a way to analyze this data without scientific expertise. Not only are these users facing challenges to analyze and access climate data, but there is specific expertise around handling the data, such as understanding model interactions and projections. For both scientists and impact users, being able to access and analyze climate data as it stands today requires too much overhead to make it an effective process.

The Solution

The overall goal of the project was to break down the different aspects of creating a climate data analysis workflow into steps that were easy-to-use, reusable, and easily shareable. Our solution is a python-based web application that allows users to dynamically create data workflows by connecting different pre-packaged analysis steps. To make it intuitive for a user to keep track of their workflow, we incorporated a visualization that displays each step and their connections to other steps. As each step is added, the workflow as a whole is run and the user is given the option to download the data created after the latest step. Lastly, users are able to save their workflows and are given a serial number for that workflow. Using the serial number, a user can either upload that workflow and update or share the serial number with another user to access.

The application is using a python based web framework, Tangelo, which was built specifically to support agile data management and visualization. To structure the workflows on the back end, we modified a python workflow library called pyutilib.workflow, which creates workflow objects containing multiple tasks, or steps. Each step available for the workflows are either NCL, NCAR Command Language, or R scripts which perform the actual analysis on the data and are called through python. For the scope of the project we focused on using NARCCAP, North American Regional Climate Change Assessment Program, data in NetCDF format, a common climate data format. After each analysis step, a new NetCDF file is created and the filename is then passed between steps and becomes available for the user to download. As steps are added to the workflow, the workflow library sends an updated workflow to a plugin in Tangelo, called Nodelink, that dynamically updates the visualization seen by the user. Workflows and their corresponding output files get saved in a MongoDB database via another Tangelo plugin, making them available for uploading or sharing given a corresponding serial number.

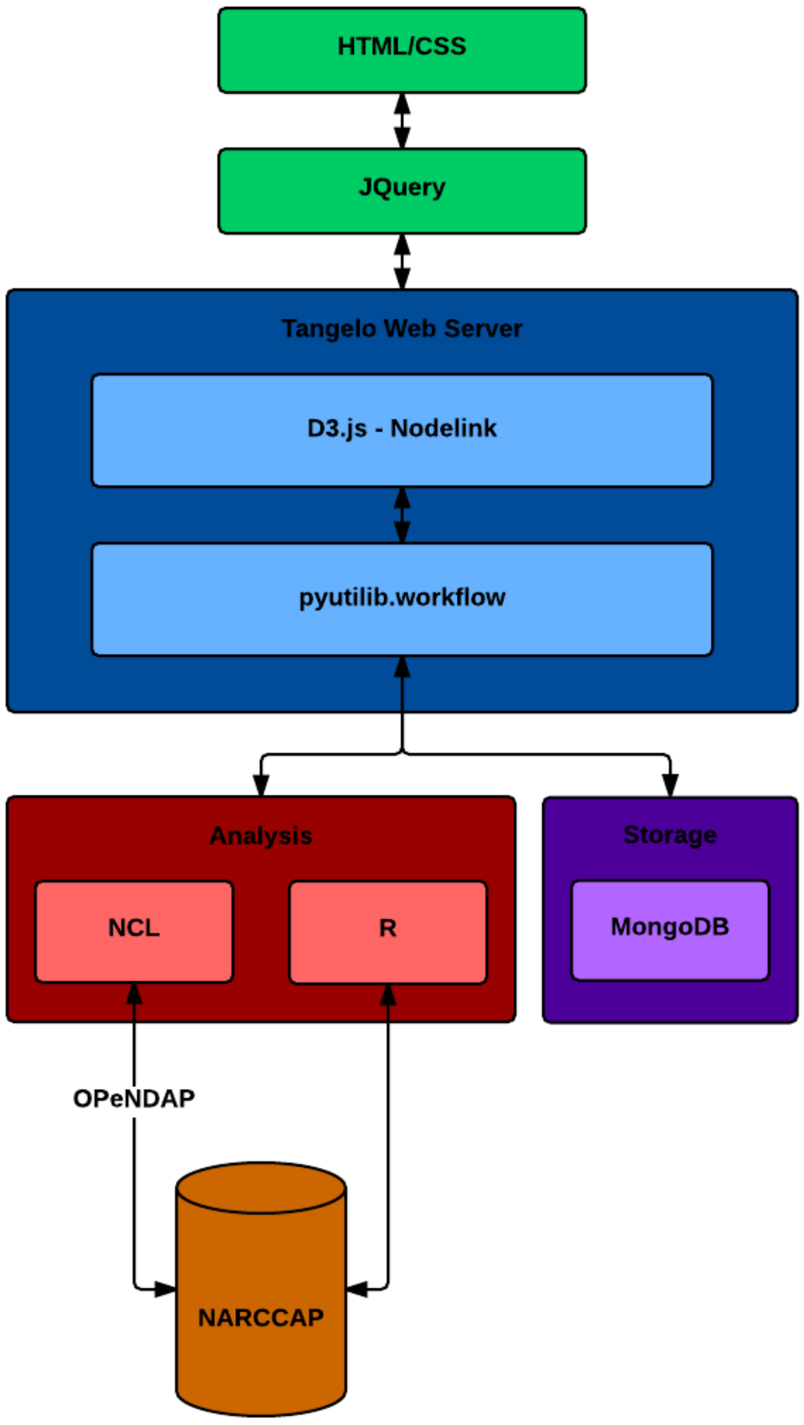
Through python we were able to run a web server, utilize different analysis languages, and support visualization to create an application for easy build out of scientific data analysis workflows that can be reused and shared.

Future Steps

This project's purpose was a proof of concept for a team at NCAR with the idea of expanding on the platform we created. The idea is to add access to more databases other than NARCCAP, add more analysis steps and steps that utilize more analysis languages, and create a template in which users in the future are able to create the own steps and submit them to be added to the application.

Architecture

Architecture Diagram and Explanation



The front end of Scientific Data Reproducible Workflows is purely HTML, CSS, and Javascript.

The backend of Scientific Data Reproducible Workflows is mostly Python, with specific tasks and analyses being implemented in NCL and R. The database for storing workflows is MongoDB, and climate model data comes from the OPeNDAP NARCCAP servers.

To connect these two pieces, we use Tangelo. Tangelo is a web framework specializing in data visualization. Javascript sends an HTTP GET request to the server for a python service. The server runs the called service and returns a JSON string that can then be parsed by the Javascript.

Folder Structure and Files of Note

- **Scientific-Data**
    - **Documentation**
        - index.html - *The Documentation File. View this to see all code documentation.*
        - **naturaldocs**
            - Languages.txt - *A configuration file for adding language support to NaturalDocs*
            - Menu.txt - *A configuration file for changing the organization of the menu*
            - Topics.txt - *A configuration file for adding topics/categories to the documentation.*
    - **NaturalDocs** - *The NaturalDocs binary and related files.*
    - **pyutilib.workflow-3.5.1** - *The modified pyutilib.workflow library.*
    - **r_library** - *The R libraries necessary for running R-scripts*
    - **tangelo_html**
        - **ncarworkflow**
            - **css**
                - main.css - *The css for the site.*
                - sidebar.css - *The css for the analysis sidebar.*
            - **javascript** - *The site javascript*
                - **analysis -** *The javascript for individual tasks*
                - loadWorkflow.js - *The javascript for loading a workflow from the database*
                - workflowBuilder.js - *The javascript for creating a workflow*
                - workflowvis.js - *The javascript for modifying a workflow and visualizing a workflow.*
            - **plugin -** *Tangelo plugins*
                - **ui -** *The ui plugin. Contains the code for the slide-out panel.*

                - **vis -** *The vis plugin. Contains the code for visualizing a workflow.*
                - **workflow -** *The workflow plugin.*
                    - **python**
                        - **customTasks** - *Contains the task class files*
                            - **ncl -** *Contains the ncl files*
                            - **r -** *Contains the r files*
                        - __init__.py - *Contains all the functions for manipulating workflows.*
            - **python**
                - updateWorkflow.py - *Contains service code that handles temporary storage of workflows as well as handles calling the correct functions to modify workflows.*
            - **stepHTML -** *A folder containing the html for all available steps.*
            - config.yaml - *A configuration file that tells tangelo which plugins to load.*
            - index.html - *The home page.*
            - loadWorkflow.html - *The html page for loading a workflow.*
            - workflow-builder.html - *The html page for building a workflow.*
- **templates -** *Template files for each part of adding a task*
    - PluginTaskTemplate.py
    - template.html
    - template.js
    - template.ncl
- requirements.txt - *The python package requirements for this software.*
- HowToUseNaturalDocs.txt - *Instructions on how to use NaturalDocs*
- CommentingGuidelines.txt - *Examples for formatting comments*

Installation

Installing the Reproducible Scientific Workflows Applications is very straightforward.

First, install these languages and packages:

- R (Version 3.1.3) - Install from http://cran.cnr.berkeley.edu/
- Python (Version 2.7.6) - Install from https://www.python.org/downloads/
- NCL (6.2.1 OPeNDAP Enabled) - Install from https://www.ncl.ucar.edu/Download/install.shtml
- MongoDB (Version 2.6.9) - Install from http://docs.mongodb.org/manual/installation/

Second, clone the GitHub repository into your desired location.

```
$ git clone git@github.com:NCAR-Scientific-Data/Scientific-Data.git
```

OR

Fork the repository at https://github.com/NCAR-Scientific-Data/Scientific-Data and clone that copy to your machine.

Third, `cd` into the newly created Scientific-Data folder. Then cd into the pyutilib.workflow-3.5.1 folder and run the following command:

```
$ python setup.py install
```

(Note: You may have to run the command as root or with sudo)

Then cd back into the Scientific-Data folder and run the following command:

```
$ pip install -r requirements.txt
```

Finally, to make sure the R tasks use the correct libraries, run the following command:

```
$ echo 'R_LIBS_USER="<Path_to_Scientific-Data>/r_library"' >
$HOME/.Renviron
```

Now all dependencies of the Reproducible Scientific Workflows are installed and you can safely run the app.

Running the Application Server

To run the application after installation, simply run this command from tangelo_html/ncarworkflow:

```
$ tangelo --config config.yaml
```

Optionally, if you need to run on a different port and the default port, use:

```
$ tangelo --config config.yaml --port <portnumber>
```

To specify a hostname besides localhost, you can use:

```
$ tangelo --config config.yaml --hostname <yourhostname>
```

And of course, you can combine these commands as necessary. To view all options for running tangelo, go here: http://tangelo.readthedocs.org/en/v0.9/tangelo-manpage.html

Pyutilib.workflow

Version

pyutilib.workflow-3.5.1

Installation

To install:

1) Navigate into pyutilib.workflow-3.5.1 as root
2) run "python setup.py install"

Any time there are changes made to the workflow library, this setup needs to be re-run. If the changes are made by the library developers, the changes with comments "# Robert Crimi" must be copied into the new library files

Summary

Pyutilib.workflow is the main framework for creating workflows in the system. With this package, we are able to define a structure for scientific workflows. We are able to serialize and deserialize these workflows to implement reproducible workflows. This package also allows for workflows to be modified and to be included in other workflows. Finally, we can modify the defined tasks to include a visualization for the site. For example, we can create symbols for specific tasks, inputs, and outputs.

As of now, workflow objects are re-run every time there is a new, deleted, or modified task; however, there is functionality in the workflow library to implement this. For example, the task class implements the function "set_ready()." This function lets the workflow know that the task already has a value for an output and to not run the task during workflow execution.

Integration

There have been several added methods to the workflow library. These can be found within the workflow directory of the library. There are changes made to "task.py", "workflow.py", and "port.py." All changes are marked with the comment "# Robert Crimi." These changes allow for serialization and modification of workflows. Additional functions can be found in the Tangelo plugin folder. These methods allow for the creation of workflows as well as adding, deleting, and modifying tasks.

R Integration

Summary

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

Integration

A few data analysis calculations are done with R. Each calculation is implemented in a function and all functions are stored in a R script. The R functions are called directly within Python using the python library called rpy2 by:

```
# import rpy2
import rpy.robjects as ro

# load the R script
scriptName = <Path_to_the_R_script>
ro.r[scriptName]

# call the R function
ro.r[<Name_of_the_R_function>]
```

Each python task can call one or more R functions from a single R script as needed for a calculation.

Integration with R is currently a proof-of-concept as those R functions need to be enhanced for further reliability.

Version

3.1.3

Install the packages

R packages can be installed in R console by

```
> install.package("<name_of_the_package>")
```

The packages will be automatically install to:
<path_to_the_project_repo>/Scientific-Data/r_library

NetCDF Packages

ncdf

Summary: This package provides a high-level R interface to Unidata's NetCDF data files, which is portable across platforms and includes metadata information in addition to the data sets. With this package NetCDF files can be opened and data read in easily. It also allows for easy creation of new NetCDF dimensions, variables, and files, and manipulate existing NetCDF files. The interface provides considerably more functionality and is not compatible with the old ‘netCDF’ package for R. There is a newer package that works with NetCDF called ncdf4, the syntax of ncdf4 is more user friendly. It is recommended to switch to ncdf4 from ncdf in the future.

For developers

1. All the R packages are under Scientific-Data/r_library. Before starting the server, make sure R is searching for the libraries in the correct folder using this command:

```
$ echo 'R_LIBS_USER="<Path_to_the_where_packages_are_installed>"' > $HOME/.Renviron
```

Otherwise, R will be unable to run, as it won’t find the necessary libraries

2. GCMs generally use non-standard calendars. Usually they are 365-day or "noleap" calendars, which means they have 365 days every year, and there is never a leap year with 366 days. However, some GCMs use a 360-day calendar with 12 months of 30 days each. R's date format doesn't understand alternate calendars, so the results will be incorrect. You may be able to solve this problem using the PCICt package, but it might also be easier to calculate dates yourself.

3. The time coordinate in NARCCAP data comes at the *end* of the averaging period. The daily maximum temperature for Jan 31st is the maximum temperature between 12:00 AM and 12:00 PM on January 31st. In 24-hour time, that's 00:00 to 24:00. But 24:00 January

31st is actually 00:00 February 1st, and since the time coordinate comes at the end of the period, the time for the January 31st maximum looks like it's on February 1st instead.

NCL Integration

Summary

NCL is used currently to do the majority of handling of the NetCDF files. Each NCL script correlates to one task in the pyutilib.workflow. The python tasks create subprocesses that run the NCL scripts with the arguments set by the user. Each NCL script should take in either an OPeNDAP url or NetCDF filename on which to process and should output an updated NetCDF file in the unique workflow folder.

Currently the unit conversion task, implemented in NCL, is a proof-of-concept task as it only does temperature unit conversion and only checks for three variations of unit descriptions in the NetCDF. For example the unit conversion looks for these variations within the source file:

”degC”/”degK”/“degF”
”C”/”K”/”F”
”C “/”K “/”F ”

The output units are then written with the syntax “degC.” In the future it may be possible to write a wrapper around the Unidata’s UDUNITS library in order to dynamically convert many unit and variable types.

Version

6.2.1 OPeNDAP enabled

For developers

The subset task adds an attribute to its output file called MainVariable, which is used in subsequent steps to keep track of the variable being processed, therefore each NCL script added is required to maintain that attribute in its output NetCDF files.

Due to restrictions with the NARCCAP thredds server, we were not able to download full variable files. In order to cut down on file size, the subset NCL file does an initial subset of the data to the Continental United States (Lat: 25° - 45°, Lon: 245° - 285°). After the initial subset, then the data is further filtered by the coordinates entered by the user.
Note: We have seen the subset task fail due to rejection of the OPeNDAP call from the Thredds server, but often re-running the task succeeds.

Each NCL task script is responsible for checking if a workflow folder has already been created and if not, creating one. Within that workflow folder is where each script will create its output file with the naming convention ‘taskId_taskname.nc.’ Only one NetCDF file per task should be created, so the scripts should first remove old files before creating new ones.

NCL Error Codes

0 =	success
1 =	standard error
2 =	missing parameter
3 =	lat/lon out of range
4 =	date out of range
5 =	incorrect input
6 =	conversion error

To add error codes to NCL scripts, use the ‘status_exit(#)’ function and then add a catch to the parent python code to return an appropriate message to the user.

Tool Versions

The following list provides references to the tools used in the system, including version numbers. These requirements can also be found in “requirements.txt”.

```
CherryPy==3.6.0
pyutilib.common==3.0.7
pyutilib.component.config==3.8
pyutilib.component.core==4.6.4
pyutilib.component.executables==3.5
pyutilib.misc==5.9.1
pyutilib.services==3.4
pyutilib.subprocess==3.6.2
pyutilib.workflow==3.5.1
PyYAML==3.11
rpy2==2.5.6
singledispatch==3.4.0.3
six==1.9.0
tangelo==0.9.0
ws4py==0.3.2
pymongo==3.0.1
```

To install these dependencies:

```
pip install -r requirements.txt
```

In addition to these tools, R, Python, and NCL were used. The following list contains version numbers for these tools:

R == 3.1.3
Python == 2.7.6
NCL == 6.2.1

There is no current mechanism for installing these dependencies, so they must be installed manually.

Maintainability

Adding New Tasks

Tangelo comes with built-in plugins which provide the functionality for representing workflows as a graph. In addition, Tangelo allows for custom plugins to be added. The “workflow builder” portion of the system has been implemented as a Tangelo plugin. These can be found in the “plugin/workflow/python” directory. Any changes that need to be made to the functionality of adding, deleting, and modifying tasks should be made in this folder. Within this folder is a file named “__init__.py”. This file holds the logic for the above mentioned functionality.

Also located in this directory is the folder named “customTasks”. This folder contains all of the task files that users of the system can use in their workflows. Directly in this folder are python files that contain the structure for implementing analysis scripts (NCL, R, etc.). This directory also contains two folders (ncl and r). These folders contain the corresponding scripts that are called within the python tasks. For example, the script in the ncl folder named “aggregate.ncl” will be called in the python file named “taskAggregate.py”.

The following instructions provide the methods to add a new task:

Develop Analysis Scripts

The main data manipulation and analysis occurs in different analysis scripts, currently written in NCL and R. To add another task script requires creating a script that takes in either an OPeNDAP url or NetCDF filename as its datasource and output an updated NetCDF file. The script must also create and/or maintain an attribute in the output file called MainVariable which holds the main variable used in the analyses (ex: tasmin, tasmax, etc) . This is done in order for the variable to be cascaded throughout the workflow process without the user specifying it in each step.

Scripts should also make sure to copy all the metadata from the input NetCDF into its output, updating any metadata corresponding to changes made in the script. Maintaining a file’s projection information is particularly important for the plot steps, especially to allow the user to plot in the file’s native projection.

The task scripts are also responsible for checking and/or creating a unique workflow folder within the tangelo_html/ncarworkflow/python/data folder. These folders are simply named the workflow Id, which is passed through the python task that calls the analysis script. Within that folder is where scripts should create their output files, with the naming convention ‘taskid_taskname.nc’. Only one NetCDF file per task should be created, so scripts should remove previous task files before writing new ones.

Scripts should be stored in their appropriate language folder under tangelo_html/ncarworkflow/plugin/workflow/python/customTasks/

There are template scripts for NCL (template.ncl) and R for further details and creating analysis task scripts.

Create Workflow Task

Once the analysis scripts have been created, python tasks need to be created in order to integrate them in the workflows. Each task is a new class which get instantiated by the workflow library. The task files need to be created in tangelo_html/ncarworkflow/plugin/workflow/python/customTasks with the naming convention PluginTaskName (ex: PluginTaskSubset). Each python task class consists of two parts, a constructor and execute function.

Before adding the constructor and execute functions, the class needs to be added to the alias repository in the pyutilib library. In order to do this these two lines must be added at the beginning of each class:

```
pyutilib.component.core.alias("taskName")
alias = "taskName"
```

The constructor for a new task should follow the template below:

```
def __init__(self,*args,**kwds):
     pyutilib.workflow.Task.__init__(self,*args,**kwds)
     self.inputs.declare(‘input1’)
     self.inputs.declare(‘input2’)
     self.outputs.declare(‘result’)
```

The inputs are the arguments needed for the corresponding analysis script and the output should generally be named ‘result’ and contain the filename of the result NetCDF.

The execute function will be different depending on the language the analysis script is written in. For example, NCL scripts are called in the execute function by creating a subprocess, where as R scripts are using a python library to access R functions. Regardless of the language, the execute functions should follow the below template:

```
def execute(self):
     Input1 = self.input1
     Input2 = self.input2
     wid = self.workflowID
     tid = self.uid
```

```
        args = [Input1, Input2]
        try:
             callAnalysisScript(args)
        catch:
             script errors
        self.result = “data/wid/tid_task.nc”
```

The wid and tid are necessary parameters for every analysis script as they are used to create workflow folders and output files.

Lastly, new python tasks need to be added to the __init__.py file in tangelo_html/ncarworkflow/plugin/workflow/python/customTasks:

```
from PluginTaskName import PluginTaskName
```

There is a template PluginTaskTemplate for a more detailed example of how to create the python classes for a new task.

Create an HTML Form for the Task

After you have successfully created your analysis script and integrated it into a workflow task, you are now ready to create an HTML form.

For each input of your task, you will need at least one corresponding form input (for dates, you may need up to three inputs per date, depending on your implementation).

For example, if a task had inputs of latitude and longitude, the form would need two corresponding text boxes in that form.

There is a template HTML file for creating this form, with an example of using text boxes, radio buttons, and drop down menus, located in templates/template.html. Because these pages get loaded directly into an already live page, you do not need to include standard doctype, head, or body tags. You can follow the template to make your HTML file fairly easily.

Once you have your HTML form, you can add it to the tangelo_html/ncarworkflow/stepHTML folder. In order to make the task available in the app, you will then need to modify 2 more files: analysismenu.js and analysis.html.

In tangelo_html/ncarworkflow/stepHTML/analysis.html, you will need to add your task to the sidebar-nav unordered list. It can be of the format:

```
<a onclick=”newAnalysis()”>Analysis</a>
```

where *Analysis* is the name of your new task.

After that, open tangelo_html/ncarworkflow/javascript/analysismenu.js. You will then add a function that looks like this:

```
function newAnalysis() {
      "use strict";
      $("#analysisHTMLLoadSection").empty();
      $("#analysisHTMLLoadSection").load("stepHTML/analysis.html");
}
```

Where *analysis* in “newAnalysis” and “stepHTML/analysis.html ” is the name of your task.

Create Javascript Interface for the Task

After you have added the HTML and updated the corresponding HTML and Javascript files, you can implement the necessary parsing to have your task add to a workflow. You can find a template files located in templates/template.js, which gives an example of all required functions. This will include callAnalysis(), anlaysis(), updateAnalysis(), and (most likely) generateNodeSelect() (where, once again, *Analysis* is replaced with the name of your new task). After implementing these functions, you will have successfully created a new task for use in the workflow.

Changes to Workflow Library

All current changes to the library are marked with the comment “# Robert Crimi” above the change. This includes class methods and fields.

To make a change to the workflow library:

1) Make any necessary changes to library
2) Sign into terminal as root
3) Navigate into the pyutilib.workflow-3.5.1 folder
4) Run “python setup.py install”

Workflow Visualization

Workflows are visualized using the “vis” Tangelo plugin, but the x and y coordinates of nodes are computed using Javascript found in the tangelo_html/ncarworkflow/javascript/workflowvis.js file. If you notice a bug in the calculation of X and Y values, you will want to check the `assignXValue` and `assignYValue` functions respectively. You can read more about that in the code documentation, provided in Documentation/index.html.

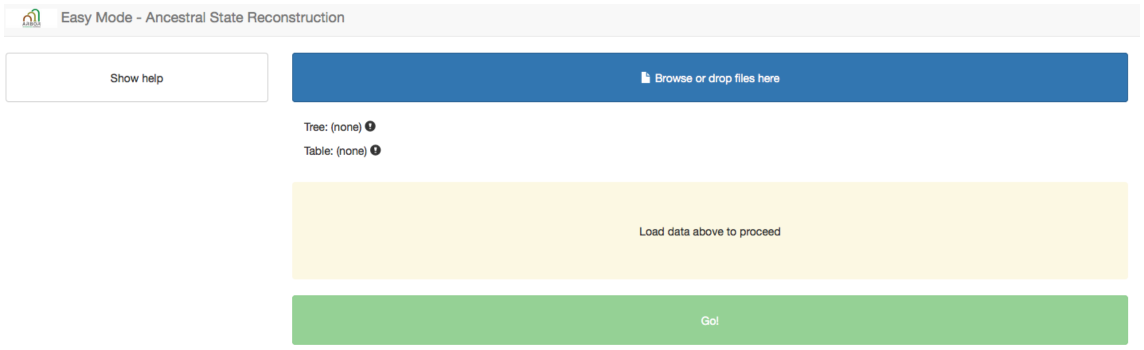
Research

Tangelo and Arbor

Summary

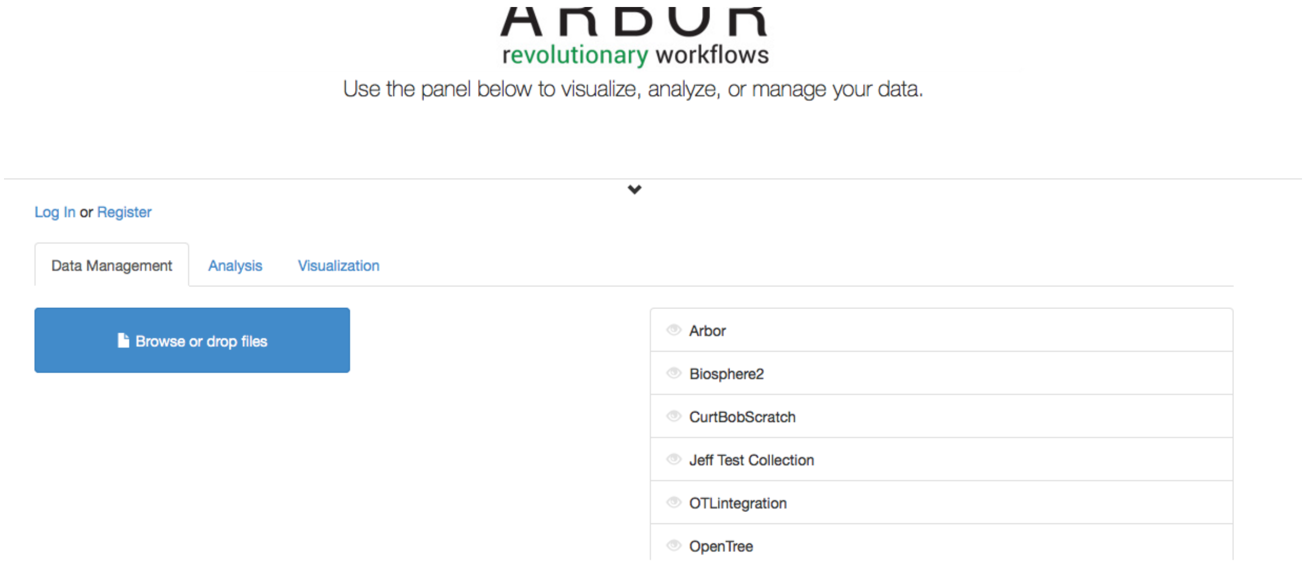
Arbor is specifically designed to be used with biological data, specifically phylogenetics. Their interface is very simple and really does not provide the workflow system we were originally looking for.

Easy Mode:


Easy Mode - Ancestral State Reconstruction
Show help
Browse or drop files here
Tree: (none)
Table: (none)
Load data above to proceed
Go!

Their demo allows you to upload a .phy and .tsv file, and then it spits out some information. The information can be changed by clicking on some names, but there isn't really a workflow like Alteryx, for example.

Expert Mode:



The “Expert Mode” demo is super confusing, I never figured out how to use the data. But from what I could tell it also relies mostly on text-based links and dropdowns, not on an “Alteryx”-type workflow.

The Workflow GUI

The Workflow GUI they were designing for their system is still in its infancy. GitHUB shows that the WorkflowGUI repository has really only had its initial commit. Not only that, but everything in the repository is still hardcoded. We would more or less be rewriting the entire code to use it with our system. There is also no documentation for the GUI.

Arbor Status

Most of the Arbor repositories were last updated in May 2014. The most recent update was about a month ago. Because of the “newness” of the project, I don’t think it would be wise to use it as part of our system.

pyutilib.workflow

https://pypi.python.org/pypi/pyutilib.workflow/3.5.1

Summary

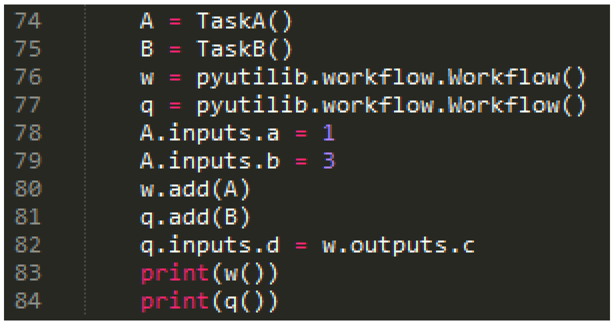
After reviewing a variety of scientific workflow tools, pyutilib.workflow shows the most promise. Pyutilib.workflow is a Python package that allows for workflow definitions. These workflows are composed of tasks, which have inputs and outputs. This package also allows for workflows to be composed of other workflows.

With this package, we will be able to define a structure for scientific workflows. This structure will allow for reproducibility as we will only need to save these objects in a database. This package will also allow for workflows to be modified for inclusion in other workflows. Finally, we can modify the defined tasks to include a visualization for the site. For example, we can create symbols for specific tasks, inputs, and outputs.

Workflows and Tasks can be represented as strings. From these strings, we can parse information for use in the site. For example, we can parse a workflow string to determine the order in which tasks are computed. Tasks also have an EmptyTask subclass to represent a task which has not been initialized. We can use this as a representation of a new task when added to a workflow.

Nested Workflows

The workflow engine allows for workflows to be nested within pre-existing workflows. This will allow users to have their workflows be pieces of larger scale projects.



```
74      A = TaskA()
75      B = TaskB()
76      w = pyutilib.workflow.Workflow()
77      q = pyutilib.workflow.Workflow()
78      A.inputs.a = 1
79      A.inputs.b = 3
80      w.add(A)
81      q.add(B)
82      q.inputs.d = w.outputs.c
83      print(w())
84      print(q())
```

To show a simple example of a nested workflow:

Here, we create an instance of two tasks. Each task is then delegated to a unique workflow. On line 82, there is a link set between the output of workflow “w” and the input of workflow “q.” The values of inputs and outputs for the workflows are the same values of the first and last task inputs and outputs respectively. This script will first run workflow “w” and then send its answer to workflow “q.”

MongoDB

Summary

- When user access our app, a session id is being generated, it becomes the key of one document in our MongoDB
- Collection -> Table, Document -> row
- Our Database consists of a collection of documents, a document is a list of JSON, each JSON has the data that needs to store for one step
- Permanent storage: If the workflow is complete, clean out the NetCDF path and plots, save the steps only and then generate a script that can runs the whole workflow, otherwise delete the whole thing
- NetCDF: Maybe only store a path to the NetCDF file
- Pymongo (version 3.0.1)

Saving a Copy of a Workflow

The current Workflow plugin presented here can save and load workflows to and from a database, but it cannot save copies of workflows. Below is a brief overview of the steps involved to implement this functionality:

First, you will need to modify the loadWorkflow.html page found in tangelo_html/ncarworkflow. I recommend adding a checkbox that says “Load a Copy.” If this checkbox is selected, then the workflow would be loaded with a new unique ID.

In order for this to be feasible, the loadWorkflow function in tangelo_html/plugin/workflow/python/__init__.py would not need to be changed. Instead, the loadWorkflow elif in the run function of tangelo_html/python/updateWorkflow.py would be updated with an additional check. After the workflow is loaded, if the user wants to load a copy, then the workflowID of the workflow would be set to a new workflowID. Then the workflow can be run and edited as normal, and when the user hits save, they will have a new, unique workflow they can use.

Contact Information

Brian Bonnlander, NCAR: bonnland@ucar.edu
Seongmin Choi, Deployment Lead: Seongmin.Choi@colorado.edu
Robert Crimi, Architecture/Research Lead: Robert.Crimi@colorado.edu
Connor Guerrieri, Source Control Lead: Connor.Guerrieri@colorado.edu
Bo Han, Test Lead: boha4482@colorado.edu
Hannah Keller, Team Lead: Hannah.Keller@colorado.edu
Hannah Thomas, Documentation Lead: imaginationandtech@gmail.com