Co-design of Transmission & Distribution for improved power system planning and operation

Aadil Latif PhD,

2022 OpenMDAO Workshop

October 24th, 2022

The Team: Bringing together grid and software experts from across NREL

Bryan Palmintier

Principal Investigator

Nadia Panossian

NYS Data Lead & Co-Simulation Coordination

Patrick Brown

Analysis Lead & Capacity Expansion

Aadil Latif

Software Lead & Distribution

Ashreeta Prasanna

DER Adoption and Rate/Incentives

Meghan Mooney

Data Flow and Geo-Spatial Analysis

Brady Cowiestoll

Bulk System and Resource Modeling and Integration

Eric Wood

Transportation Integration

Zhaocai Liu

Transportation Integration

Miranda McDevitt

Project and Financial Support

Co-design of Transmission & Distribution for improved power system planning and operation

Technology Summary

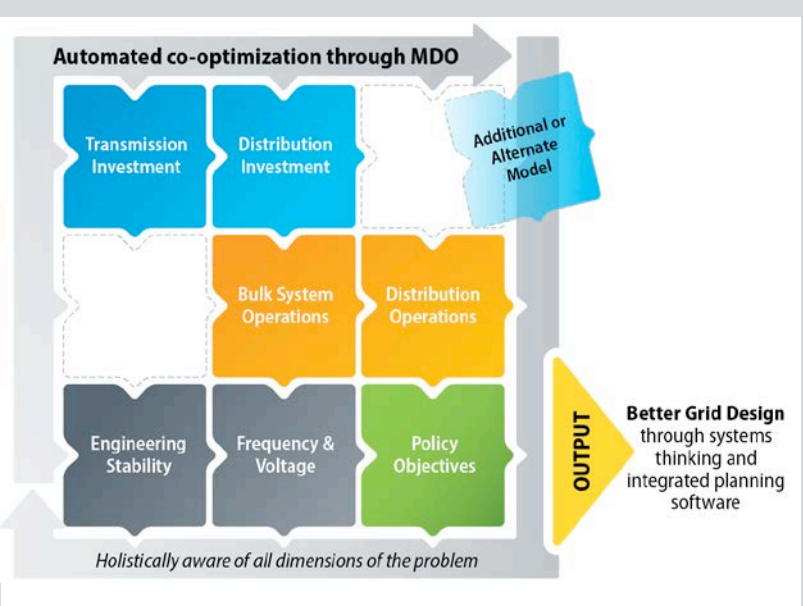
- Adapt aerospace-developed multi-disciplinary design and analysis optimization (MDO) techniques to transform electric power grid planning
- Modular design to mix-and-match existing trusted analysis tools
- Incorporates technical, economic, and social considerations into one framework
- Provides key links required to plan modern grids, including stability with widespread inverter-based resources, DER-wholesale interactions (FERC 2222)

Technology Impact

- Partner, NYSERDA provides natural pioneering test case at a scale that matters by planning grid to meet aggressive low-carbon policies in New York State
- Broader tech-to-market through multi-state technical review committee & examples

Proposed Targets

Metric	State of the Art	Proposed
Simulation scale	One neighborhood for distribution or one BA for transmission	Entire state/region with customer level resolution
Time to conceptual design decision	Sequential, semi-manual process lasting months	Holistic solution in hours or days
Design Improvement	Separate processes may overlook higher value options	>$1B cost savings and/or improved resiliency/stability

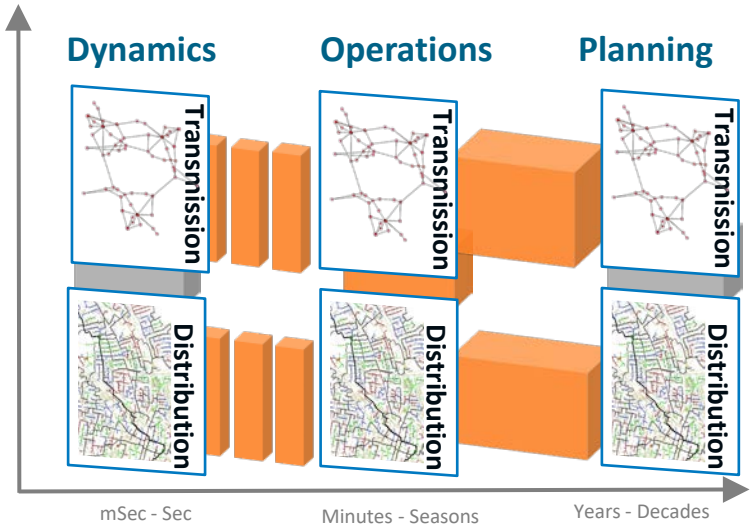


Simplified TD-Plan software block diagram, highlighting how MDO enables automated interactions among a wide range of off-the-shelf investment, engineering, and simulation tools identify better grid designs and accelerate the transition to zero-carbon energy.

Holistic T&D Planning to rapidly plan clean, reliable electric grids

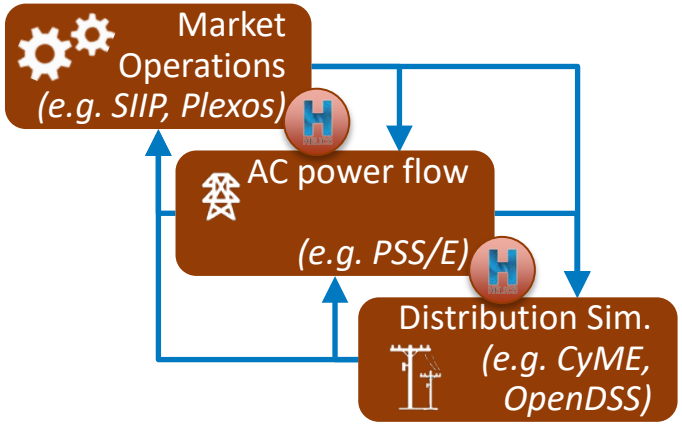
Background

- Many legacy tools and processes exist for operations, planning, and markets:
  - Trusted by stakeholders, Continuously improved
  - Siloed Transmission vs. Distribution, engineering vs. economics
- But, it is now clear that grid & technology evolution increasingly require
  - Coordinated decision making
  - Integrated T&D resource planning and management
  - Maximized asset value and utilization



HELICS optimized for operations simulations

Existing Capability:



Example uses:

- T-D Market Interactions
- Grid services from DERs

Integrated **Co-Simulation**

- Data passing and time synchronization
- Developed by NREL with other National Labs

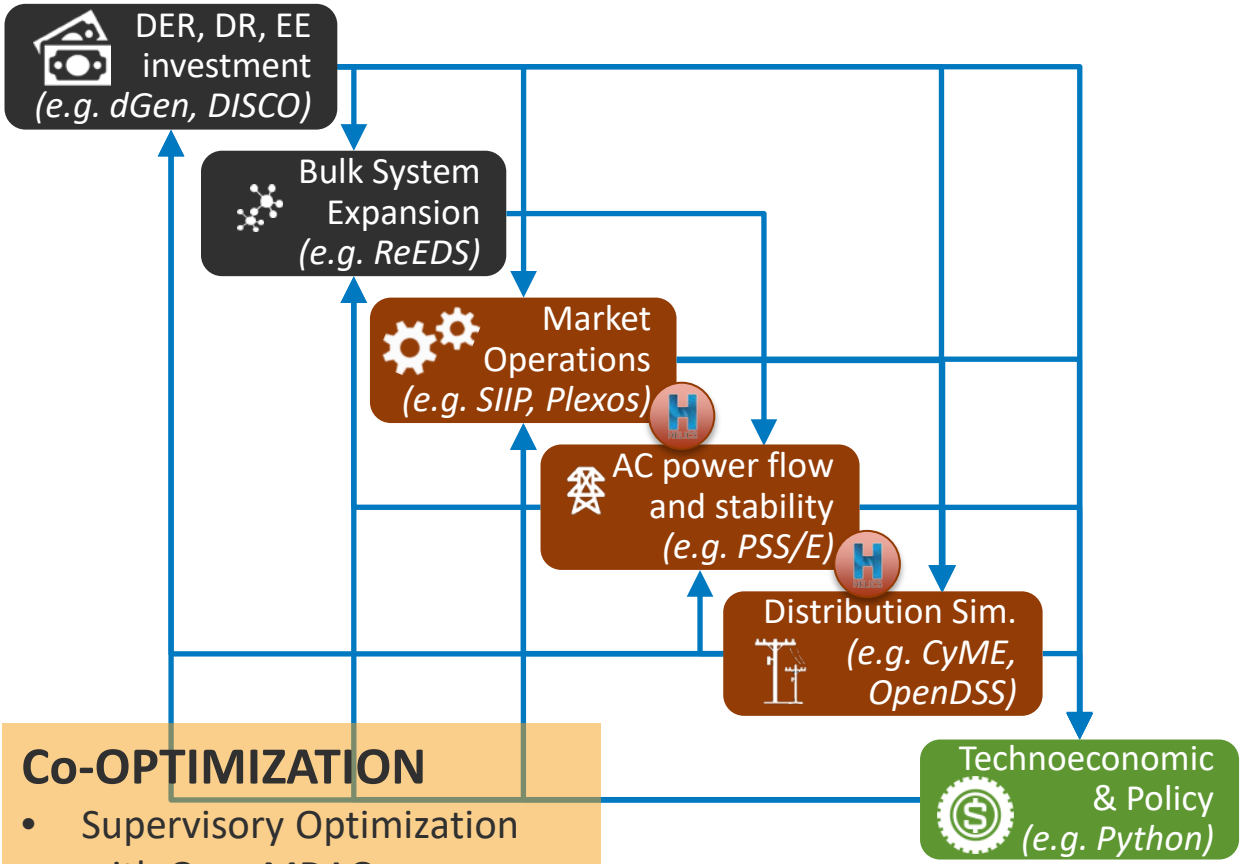
HELICS

Note: Grid design/investment fixed exogenously. Co-simulation "just*" simulates how the grid (and other infrastructures) behave in operations.

** Not easy and already provides many key insights*

MDAO enables modular Investment Planning

MDAO=Multi-disciplinary Design Analysis and Optimization



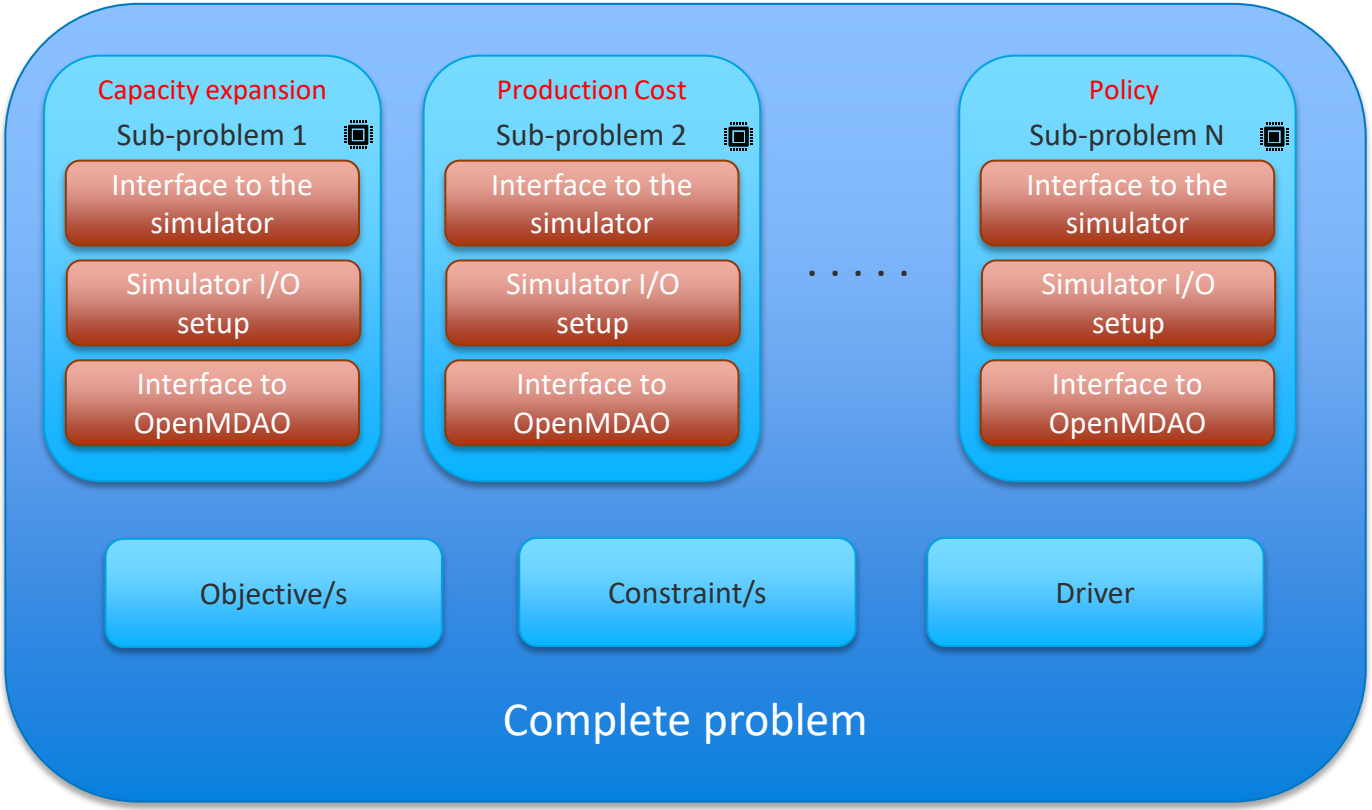
Co-OPTIMIZATION

- Supervisory Optimization with OpenMDAO
- Adapted from Aerospace

Example uses:

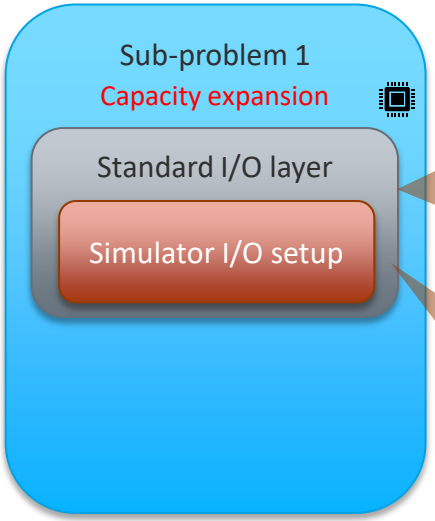
- T-D Market Interactions
- Grid services from DERs
- Value of DER
- Joint T&D Investment planning…
  - With rich operations
  - And stability assessment
- Policy Target Evaluation
- Many others

Approaching the problem in a systematic manner



1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

Defining a standard I/O layer



The standard I/O layer is a work in progress.

A. Wraps around simulator I/O
B. Will cover all power system subdomain simulators eventually.

Standards such as CIM (common information model) MultiSpeak may be used to implement the standard I/O layer

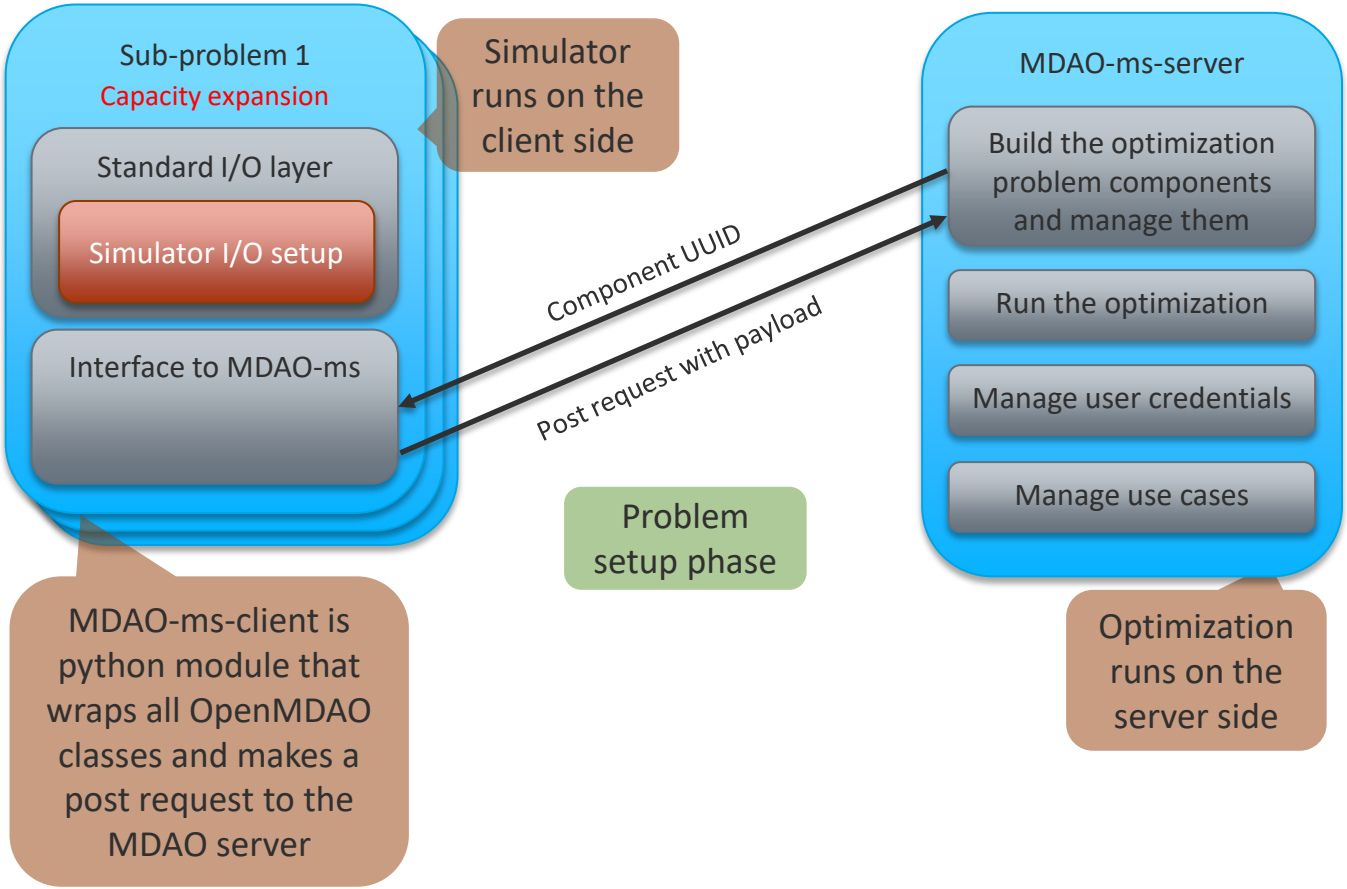
1. How do I standardize variable names such that I/O from any sub problem is understood by all others?

2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?

3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?

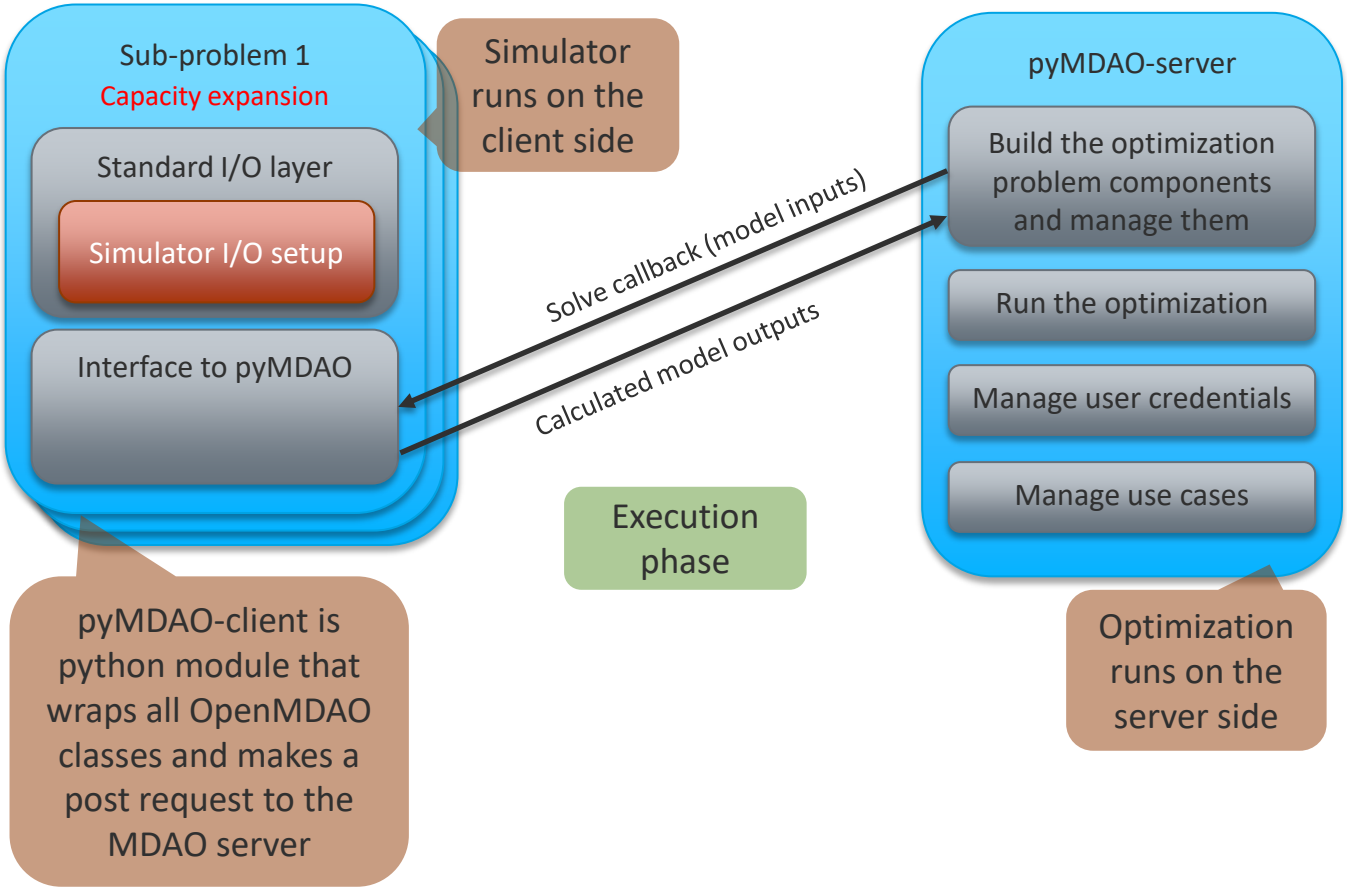
4. How do I make the tool intuitive and easy to use for end users

A microservice architecture-based approach to enable scaling



1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

A microservice architecture-based approach to enable scaling



1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

A microservice architecture-based approach to enable scaling

```python
import openmdao.api as om
#import openmdao_cli.client.client as om
from openmdao_cli.client.compute_manager import ComputeServer

class Paraboloid(om.ExplicitComponent, ComputeServer):

    def setup(self):
        self.add_input('x', val=0.0)
        self.add_input('y', val=0.0)
        self.add_output('f_xy', val=0.0)

    def setup_partials(self):
```

```
(mdao) C:\Users\alatif\Documents\GitHub\openmdao-prototype\examples>python "example 2 - Wroking with classes.py"
[-15.]
Optimization terminated successfully    (Exit mode 0)
            Current function value: -27.333333333333
            Iterations: 5
            Function evaluations: 6
            Gradient evaluations: 5
Optimization Complete
-----------------------------------
[-27.33333333]

(mdao) C:\Users\alatif\Documents\GitHub\openmdao-prototype\examples>
```

```python
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options["optimizer"] = "SLSQP"

    prob.model.add_design_var("parab_comp.x", lower=-50, upper=50)
    prob.model.add_design_var("parab_comp.y", lower=-50, upper=50)

    prob.model.add_objective("parab_comp.f_xy")

    prob.setup()

    prob.set_val('parab_comp.x', val=3.0)
    prob.set_val('parab_comp.y', val=-4.0)
    prob.run_model()
    print(prob['parab_comp.f_xy'])

    prob.run_driver()
```

1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

A microservice architecture-based approach to enable scaling

Client side

```
(mdao) C:\Users\alatif\Documents\GitHub\openmdao-prototype\examples>python "example 2 - Wroking with classes.py"
Callback server starting @ port: 54448
[-15.]
[-27.33333333]

(mdao) C:\Users\alatif\Documents\GitHub\openmdao-prototype\examples>
```

Server side

```
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
Connected to database:  C:\Users\alatif\Documents\GitHub\openmdao-prototype\openmdao_cli\server\database.sqlite
C:\Users\alatif\.conda\envs\mdao\lib\site-packages\flask_sqlalchemy\__init__.py:872: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  warnings.warn(FSADeprecationWarning(

 * Debugger is active!
 * Debugger PIN: 553-675-990
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ExplicitComponent/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/add_subsystem HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ScipyOptimizeDriver/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/_set_driver HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ScipyOptimizeDriver/__setitem__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/add_design_var HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/add_design_var HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/__init__ HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Group/add_objective HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ExplicitComponent/add_input HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ExplicitComponent/add_input HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ExplicitComponent/add_output HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/setup HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/set_val HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/set_val HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/ExplicitComponent/declare_partials HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/run_model HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/_get_variable_value HTTP/1.1" 200 -
Optimization terminated successfully    (Exit mode 0)
            Current function value: -27.333333333333
            Iterations: 5
            Function evaluations: 6
            Gradient evaluations: 5
Optimization Complete
-----------------------------------
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/run_driver HTTP/1.1" 200 -
127.0.0.1 - - [18/Oct/2022 12:54:10] "POST /openmdao/Problem/_get_variable_value HTTP/1.1" 200 -
```

1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

Advantages of the proposed architecture

- OpenMDAO users can port existing with ease.
- The architecture enables scaling to multiple machines
- The developed prototype make use of OpenMDAO's documentation (no need for parallel documentation effort)

```
(mdao) C:\Users\alatif>python
Python 3.10.0 | packaged by conda-forge | (default, Nov 20 2021, 02:18:13) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import openmdao_cli.client.client as om
>>> help(om.Group)
Help on class Group in module openmdao_cli.client.client:

class Group(Client)
 |  Group(*args, **kwargs)
 |
 |  Class used to group systems together; instantiate or inherit.
 |
 |  Parameters
 |  ----------
 |  **kwargs : dict
 |      Dict of arguments available here and in all descendants of this Group.
 |
```

1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

Standardizing simulator interfaces

- The MDAO-ms Python library provides base classes user would need extend to add a new simulation to the MDAO-ms echo system.
  - This higher-level interfaces requires no knowledge of OpenMDAO from the user
  - Enables user to define custom I/O
  - Enables standard simulator interface.

1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

Multiple interfaces with varying level of complexity

- The MDAO-ms framework provides users the option of developing simulator interface with varying levels of complexity
  - **Advanced** users can choose to use only the server-side implementation.
    - Users would need to write their own client-side code. This includes setting up the callback framework
  - **Intermediate** level users can choose to use both the client and server-side implementation.
    - This requires working knowledge of OpenMDAO
  - **Beginners** can use the high-level interface.
    - This only requires familiarity with the simulator API
    - OpenMDAO knowledge is not required

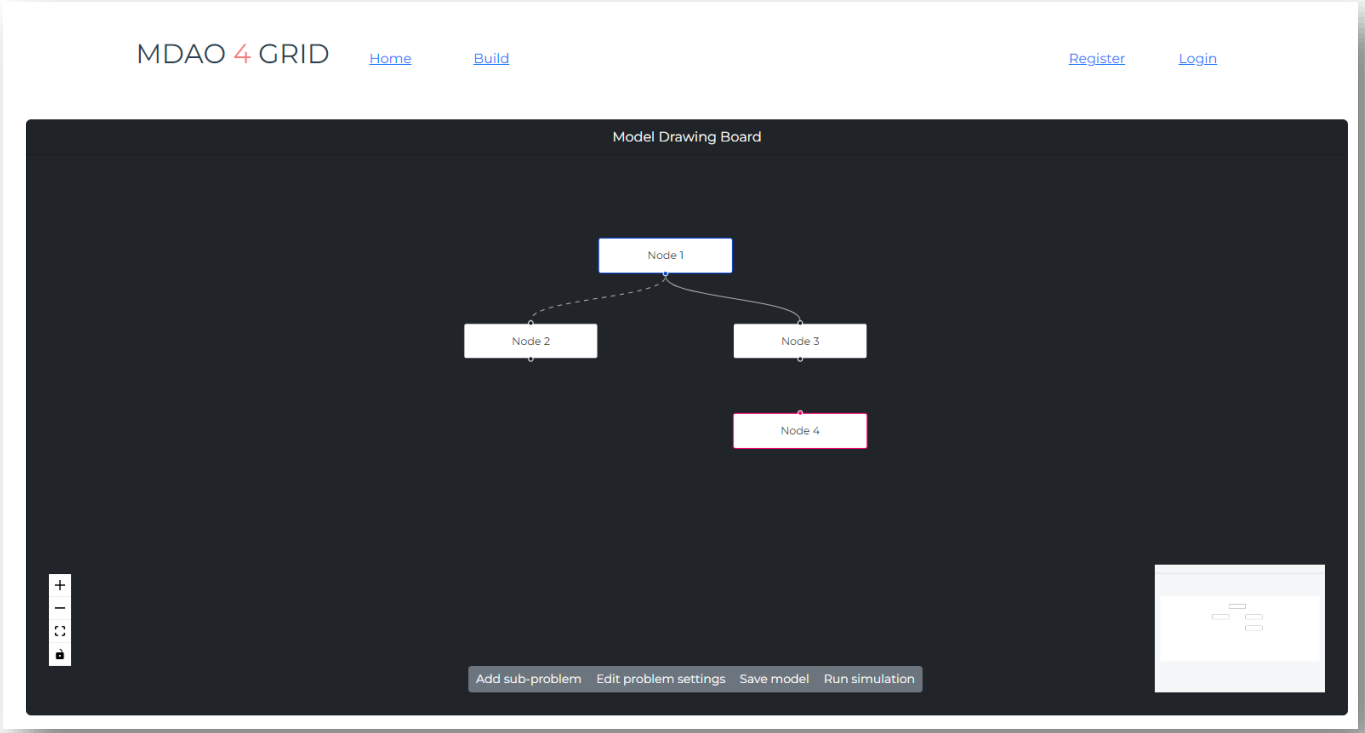
1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

UI development (in progress)

- UI currently in development (uses Vue JS)
- UI allows users to build use cases using the drag drop connect approach (think MATLAB Simulink).
  - Standardized interfaces can be added as a subproblem using a drag drop action
  - Variable exchange can be simplified
- The UI enables credential management, use case management (save / load / edit), and viewing of saved simulation results
- The implementation is minimal but can be extended in the future

1. How do I standardize variable names such that I/O from any sub problem is understood by all others?
2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?
3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?
4. How do I make the tool intuitive and easy to use for end users

UI development (in progress)



1. How do I standardize variable names such that I/O from any sub problem is understood by all others?

2. Most power system simulators are computationally expensive. Use of a single machine is prohibitive. How do I scale to multiple machine ?

3. Interface for each simulator is unique. Instead of producing one-off ad hoc interfaces, how do I design a standard interface for all subproblem?

4. How do I make the tool intuitive and easy to use for end users

Future work

- Support all modules within the OpenMDAO library
- Add support for uploading / using surrogate models on server side
    - Enables workaround NDA requirements which very often result in project delays
    - Enables surrogate models to run on server side given they have computational burden
- Add support of external surrogate model building libraries
- Build interface for power system sub-domain simulators

Discussion

www.nrel.gov