OpenMDAO: Status and Directions

Justin Gray

July, 2011
Outline

1. Project Overview
2. Process Improvements
3. GUI Development
4. Auto-Architecture Implementation
5. Going Forward
Overview:

- Python based, open source engineering framework
- Designed to facilitate the linking together of analysis codes
- Provides an environment for rapid engineering code development
- Enables simple development and application of MDAO architectures

Goals:

- Facilitate collaboration among industry, academia, and government
- Enable the sharing/reuse of tools and analysis techniques by providing a common framework to build on
- Advance the state-of-the-art for engineering MDAO
Our current development efforts:

- Development Process Improvements
- Browser-based GUI
- Support for MDAO architectures: user can specify problem formulation and add an architecture that gets automatically configured
- Analytical Derivatives: using a system called “Fake” Finite Difference to easily allow mixed analytical and numerical derivatives
- Supporting two geometry-focused NRA efforts
Outline

1. Project Overview
2. Process Improvements
3. GUI Development
4. Auto-Architecture Implementation
5. Going Forward
New Code Hosting Site: GitHub

Transitioned the project to use Git version control system with GitHub code hosting. http://github.com/OpenMDAO
Documentation Updates:

- New plugin documentation tools: developers can now include plugin docs which are consistent with the OpenMDAO docs
- Reorganizing the documentation:
  - Combined the standard library and source documentation
  - Each sub-package has its own docs directory where developers can write “narratives”
- Now host two versions of the docs on the website:
  1. Release Docs: documentation for the latest release
  2. Dev Docs: documentation which tracks the latest changes to the dev branch
Outline

1 Project Overview

2 Process Improvements

3 GUI Development

4 Auto-Architecture Implementation

5 Going Forward
OpenMDAO is being designed with a browser-based GUI.

- Makes uses of state-of-the-art GUI development tools
- Can run server locally on machine; no Internet connection required
- Built-in ability to access OpenMDAO remotely, if OpenMDAO GUI server is exposed to the Internet
- Heavy reliance on Javascript, “Web 2.0 Technologies”
Prototype implementation
Built-in code editor with syntax highlighting
Outline

1. Project Overview
2. Process Improvements
3. GUI Development
4. Auto-Architecture Implementation
5. Going Forward
Idea: Allow the user to specify the problem formulation and provide the disciplines, and then automatically implement an architecture for them.

- New class, *Architecture*, where MDAO architectures are implemented
- *Architecture* can be used for optimization algorithms as well (e.g., Efficient Global Optimization)
- New section of standard library, openmdao.lib.architectures. Currently have implemented MDF, BLISS, CO, EGO.
- Can apply multiple architectures to the same problem, to compare them
Architectures are constructed, at runtime, from a combination of drivers and workflows.

- Architectures are just a “thin” layer on top of standard OpenMDAO classes: Component, Driver, Workflow, Assembly
- Very flexible: after an architecture is applied, the user can “tweak” it if necessary.
Example: Sellar Test Problem

\[
\begin{align*}
\min & \quad x_1^2 + z_2 + y_1 + e^{-y_2} \\
\text{w.r.t} & \quad z_1, z_2, x_1 \\
\text{such that:} & \quad \frac{y_1}{3.16} - 1 \geq 0 \\
& \quad 1 - \frac{y_2}{24} \geq 0 \\
& \quad -10 \leq z_1 \leq 10 \\
& \quad 0 \leq z_2 \leq 10 \\
& \quad 0 \leq x_1 \leq 10
\end{align*}
\]

 Discipline 1:

\[
y_1(z_1, z_2, x_1, y_2) = z_1^2 + x_1 + z_2 - 0.2y + 2
\]

 Discipline 2:

\[
y_2(z_1, z_2, y_1) = \sqrt{y_1} + z_1 + z_2
\]

The MDF formulation of the Sellar problem
The BLISS formulation of the Sellar problem
class Sellar(ArchitectureAssembly):
    """ Optimization of the Sellar problem using MDF
    Disciplines coupled with BroydenSolver.
    """
    def __init__(self):
        """ Creates a new Assembly with this problem
        Optimal Design at (1.9776, 0, 0), Optimal Objective = 3.18339"
        super(Sellar, self).__init__()
        #add the discipline components to the assembly
        self.add('dis1', SellarDiscipline1())
        self.add('dis2', SellarDiscipline2())
        #START OF MDAO Problem Definition
        #Global Des Vars
        self.add_parameter(('dis1.z1','dis2.z1'),low=-10,high=10)
        self.add_parameter(('dis1.z2','dis2.z2'),low=0,high=10)
        #Local Des Vars
        self.add_parameter('dis1.x1',low=0,high=10)
        #Coupling Vars
        self.add_coupling_var('dis2.y1','dis1.y1')
        self.add_coupling_var('dis1.y2','dis2.y2')
        self.add_objective('(dis1.x1)**2 + dis1.z2 + dis1.y1 + math.exp(-dis2.y2)')
        self.add_constraint('3.16 < dis1.y1')
        self.add_constraint('dis2.y2 < 24.0')
        #END OF MDAO Problem Definition
        self.dis1.z1 = self.dis2.z1 = 5.0
        self.dis1.z2 = self.dis2.z2 = 2.0
        self.dis1.x1 = 1.0
        self.dis1.y2 = 3.0
With just a few lines of code, you can run different architectures on the same problem.

```python
# Use the MDF Architecture
prob = Sellar()
set_as_top(prob)
prob.architecture = MDF()
prob.run()
```

```python
# Use the BLISS Architecture
prob = Sellar()
set_as_top(prob)
prob.architecture = BLISS()
prob.run()
```
Using MDF Architecture
Minimum found at (1.977639, 0.000000, 0.000000)
Minimum differs from expected by (0.000039, 0.000000, 0.000000)
Couping vars: 3.160002, 3.755279
Minimum objective: 3.18339561109
Elapsed time: 0.338097810745 seconds

Using BLISS Architecture
Minimum found at (1.978209, 0.000000, 0.000000)
Minimum differs from expected by (0.000609, 0.000000, 0.000000)
Couping vars: 3.162028, 3.756419
Minimum objective: 3.1853960100145242
Elapsed time: 0.719602108002 seconds
1. Project Overview

2. Process Improvements

3. GUI Development

4. Auto-Architecture Implementation

5. Going Forward
- Paying close attention to the development process and trying to make it easy for external developers. Suggestions are welcome.
- Working hard to expand our library of MDAO architecture implementations. Outside contributions very welcome!
- Our GUI development has just started, but is moving along nicely.
- Did not give many specifics on our work with derivatives, but we expect to publish some work on this soon.
- We’ve just started a large, multi-year effort focused on very broad support for geometry integrated tightly into the framework.
OpenMDAO: Status and Directions

Going Forward

24 July, 2011
www.nasa.gov