Service Oriented Computing EnviRonnement (SORCER) for Large Scale, Distributed, Analysis & Optimization

Ray Kolonay
Mike Sobolewski
US Air Force Research Lab
**Typical Target Application**

**ESAV Configuration**

**Requirements:**
- Range = 4000 nmi
- Payload = 5-10% weight fraction
- Maneuver Loads at Cruise – 2.5 g
- Cruise speed: Mach 2.0
- Cruise L/D: 8.5-9
- Level 1 Flying Qualities

**Configuration Data**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cruise Speed</td>
<td>M=2.5</td>
</tr>
<tr>
<td>Wing Area</td>
<td>3299 sq ft</td>
</tr>
<tr>
<td>Aspect Ratio</td>
<td>1.20</td>
</tr>
<tr>
<td>LE Sweep</td>
<td>70 deg</td>
</tr>
<tr>
<td>Payload</td>
<td>20,000 lb</td>
</tr>
<tr>
<td>TOGW</td>
<td>285,092 lb</td>
</tr>
<tr>
<td>Empty Weight</td>
<td>106,357 lb</td>
</tr>
<tr>
<td>Fuel Volume</td>
<td>154,364 lb</td>
</tr>
<tr>
<td>Fuel Fraction</td>
<td>0.54</td>
</tr>
<tr>
<td>Cruise L/D (M=2.5)</td>
<td>8.2</td>
</tr>
</tbody>
</table>

- Baseline configuration and performance data from previous STAV and CESTA studies

- Warm-up, Taxi, Takeoff Allowance: 20 Minutes Idle + Max Pwr Takeoff to Initial Climb Speed
- Landing Reserves: 20 Minutes Loiter at 10,000 ft + 5% Internal Fuel

- Opt Mach/Alt Cruise (min 2.2M)
- Descent (no range credit)
- Climb to Opt Mach/Alt
- Combat @ Ingress M/Alt
  - Drop A/G Weapons
  - 180° Turn

- 20 min Loiter at 10k ft
- Accel and Climb (Max ROC)
- 2000 n.mi Combat Radius

- Full Fuel 20,000 lb Payload

- 1.05 Fuel Flows
ESAV “Design Drivers”
“Best in Class Approach (BCA)”

Components are one or more engineering computational applications that need to be “glued” together to execute a process. (BTW, components may be on a distributed heterogeneous network)

MDMFMS-AO-UQ Computations Require Real Time Seamless Access to Applications, Data (100s-1000s) & Compute Resources by ALL
A service-oriented product development environment

• Provides an open flexible design environment which allows universal availability and incorporation of existing data, tools/methods, processes and hardware as services.

• Provides a common way to model your analysis and design process in conjunction with your product data.

A network-based distributed framework

• Supports collaboration among geographically distributed engineering and business partners.
SORCER Enables Globally Distributed MDMFMS-AO-UQ Enterprise Computing

Requestors request services from the service providers (machines/humans). Requestors may not care where or who supplies the services.

Service: An entity that publishes (by type) functional and dynamic capabilities and data on the network.
Communication Between “Communities”

Good “Community Members” Publish Information for Collaborators

Structural Sciences People, Tech., Data & Tools

Aeronautical Sciences People, Tech., Data Tools

Control Sciences

Real Time Access to Applications & Data by ALL Communities Interdependent Collaboration!
Service Oriented Computing Environment - SORCER

A network-based distributed framework

Service Oriented

Neutrality:
1. Location
2. Protocol
3. Implementation
**SORCER Terminology**

- **Service provider**: a remote object accepting *exertions* from service requestors and performs calculations. Can provide one or more services.
- The **grid/cloud**: a collection of service providers on the network.
- **Exertion** (*think process representation or workflow*): defines collaborations - service-oriented programs. An object that represents a process by specifying the relationship between services and the information passed between them.
Exertion

Elementary Exertion

Composite Exertion

Engineering Example

Task
Signature
Context
Control Strategy

Job
Task
Signature
Context
Control Strategy

Process
Turbine Blade Analysis

Actions
Generate Solid Geom. Model
Mesh Solid Geom. Into FEM
Apply BC To FEM
Apply Materials To FEM
Solve FEM for Mech. Stresses

Data
Seed Geometry
Shank Parameters
Solid Shank & Airfoil
Mesh Strategy
Turbine Blade FEM
Boundary Condition (BC) Model
FEM with BCs
Materials Model
FEM with BCs & Materials
Mech. Solution Strategy
Turbine Blade FEM Stresses

All are Objects
**Models** – consists of a collection of **Variables**.
- Current available models – ResponseModel (including sensitivities), ParametricModel, OptimizationModel

**Variables** - can be dependent on other **Variables** enabling distributed **functional programming**. Variables can have multiple evaluators enabling **multi-fidelity calculations** for a specific variable’s value.
- Current Variable Types – DESIGN, DEPENDENT, DERIVATIVE, GRADIENT, RANDOM, BOUNDED, LINKED, PARAMETER

**Evaluators** – are used to determine the value of variables and their partial derivatives (chain rule works) with respect to their dependencies (Variables).
- Current Evaluator Types – ModelEvaluator, ExertionEvaluator, ExpressionEvaluator, GroovyEvaluator, JepEvaluator, MethodEvaluator, SOAEvaluator, FiniteDifferenceEvaluator

**Filters** - are used to map the results of **Evaluators** to **Variable Values**. Think unix shell piping. Filters can be concatenated $n$ times.

**All are Objects**
Variable - Value, Filters, Evaluators

Variable

Model

Computational Entity. Creating Large Amounts of Data
The same engineering quantity can be computed in many different fashions.

- Analytic expressions
- Historical databases/empirical relations
- Solutions to various levels of DE & PDE
- Surrogates to DE & PDE solutions

For Example - Induced Drag
- Closed form (high Aspect Ratio assumption)
- Trefftz-Plane analysis
- Force integration
- Various Surrogates to the above

Analyst/Designer will need access to these different levels of fidelity depending on;
- Task at hand (Design Space Exploration, Validation etc.)
- Flight Condition
- Configuration being developed

The Computational Framework Needs to Support this Multi-Fidelity
Variable Structure

\[ y_1(x_1, x_2, x_3), \quad y_2(x_4, x_5, x_6, y_1), \quad y_3(x_6, x_7, x_8, y_2) \]

**Functions of Functions**

- **Variables**
  - \( y_1 \)
  - \( y_2 \)
  - \( y_3 \)

- **Evaluators**
  - \( y_{1e1} \)
  - \( y_{1e2} \)

**Multidisciplinary**

- Drag via N-S Calc.
- Drag via Surrogate Calc.

**Multi-fidelity**

- Gradient
  - \( y_{1e1g1} \)
  - \( \text{dy1e1dx1} \)
  - \( \text{dy1e1dx2} \)
  - \( \text{dy1e1dx3} \)
  - \( \text{dy1e1dx4} \)

- \( \text{dy1e1dx1e1} \)
  - \( \text{dy1e1dx1e2} \)
  - \( \text{dy1e1dx4e1} \)
  - \( \text{dy1e1dx4e2} \)

- \( \text{dy1e2dx1e1} \)
  - \( \text{dy1e2dx1e2} \)
  - \( \text{dy1e2dx4e1} \)
  - \( \text{dy1e2dx4e2} \)

**Observe-Observable Pattern (demand driven calculations)**
Model Example: Induced Drag Optimization

- Center Line: \( \alpha \)
- Left Semi-span: 20'
- Right Semi-span: 20'
- Chord: 6'
- 40 Control surfaces at 20% c
  - cs1-cs20 on right hand side
  - Cs21-cs40 on left hand side

\[ M, q \]
Trefftz-Plane Induced Drag

\[ D_I = \frac{-1}{4\pi\rho_\infty V_0^2} \sum_{i=1}^{n} Lpus_i \Delta y_i \sum_{k=1}^{n} \left[ Lpus_k - Lpus_{k+1} \right] \left[ \frac{1}{y_i - y_k} - \frac{1}{y_i + y_k} \right] \]

\( Lpus_j(\alpha, \beta_i) \) – Lift per unit span (Euler calculations)
\( \alpha \) - angle of attack
\( \beta_i \) - control surface deflection of \( i \)th surface

Design Variables \( \bar{x} = \{\alpha, \beta_i\}^T \)

\( Lpus_k = \rho_\infty V_\infty \Gamma_k \) (Kutta-Joukowski theorem)
Optimization Problem

Minimize: \( D_I(Lpus_i(\alpha, \beta_i, xl_i)) \)

Such That: \( L_T(Lpus_i(\alpha, \beta_i, xl_i)) = L \)

\(-5.0 \leq \alpha \leq 5.0\)

\(-10.0 \leq \beta_i \leq 10.0\)

Design Variables \( x_i = \{\alpha, \beta_{1-20}\}^T \)

Linked Variables \( xl_i = \{\beta_{21-40} = \beta_{1-20}\}^T \)

Dependent Variables: \( D_I(Lpus_i(\alpha, \beta_i, xl_i)), L_T(Lpus_i(\alpha, \beta_i, xl_i)), Lpus_i(\alpha, \beta_i, xl_i) \)

\(D_I(Lpus_i(\alpha, \beta_i, xl_i)) \) and \(L_T(Lpus_i(\alpha, \beta_i, xl_i)) \) are functions of functions and \(Lpus_i(\alpha, \beta_i, xl_i) \) is a function of the independent and linked design variables

At Transonic Mach #'s Computation of Lpus requires Euler Solution
<table>
<thead>
<tr>
<th>Fidelity</th>
<th>$L_{pus_i}$</th>
<th>$\frac{\partial L_{pus_i}}{\partial x_i}$</th>
<th>$D_i$</th>
<th>$\frac{\partial D_i}{\partial x_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Potential Method (LPM)</td>
<td>X</td>
<td>FFD</td>
<td>X</td>
<td>FFD</td>
</tr>
<tr>
<td>Standard One Point Approximation with LPM</td>
<td>X</td>
<td>Constant</td>
<td>X</td>
<td>Constant</td>
</tr>
<tr>
<td>Modified One Point Approximation with LPM</td>
<td></td>
<td>X</td>
<td>X</td>
<td>Semi-Analytic</td>
</tr>
<tr>
<td>Kriging with LPM</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Euler Method (EM)</td>
<td>X</td>
<td>FFD</td>
<td>X</td>
<td>FFD</td>
</tr>
<tr>
<td>Standard One Point Approximation with EM</td>
<td>X</td>
<td>Constant</td>
<td>X</td>
<td>FFD</td>
</tr>
<tr>
<td>Modified One Point Approximation with EM</td>
<td></td>
<td>X</td>
<td>X</td>
<td>Semi-Analytic</td>
</tr>
<tr>
<td>Kriging with EM</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constructing a Model Consists of Two Steps

- **Model Definition (Skeleton)**
  - Defines the model Variables (independent, dependent, parameters, objectives, constraints.
  - Defines the Variable Realizations, Evaluations, and Differentiation

- **Model Configuration (Muscle)**
  - Develops the evaluators and filters for all variables and derivative variables.

*Models can be created programmatically with or without Operators (Functional Programming)*
OptimizationModel omodel = optimizationModel("Induced Drag"
  designVars(vars(loop("i",1,20),"beta$i$", 0.2, -10.0, 10.0)),
  linkedVars(names(loop("i",21,40),"beta$l$i$")),
  designVars(var("alpha", 5.0, -5.0, 10.0)),
  designVars(var("mach",0.75, 0.7, 0.85)),
  designVars(var("q", 5.0, 4.5, 10.0)),
  designVars(var("static pressure", -1., -1., 14.696))
  parameter("gamma", 1.4),
  dependentVar("DI",
    realization(
      evaluation("LinearPotentialExact", "DIAstrosExacte",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"q"), gradient("DIAstrosExacteg1")) ),
      evaluation("LinearPotentialSOA", "DIAstrosSOAe",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"q"), gradient("DIAstrosSOAeg1")) ),
      evaluation("LinearPotentialMOA", "DIAstrosMOAe",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"q"), gradient("DIAstrosMOAeg1")) ),
      evaluation("LinearPotentialKriging", "DIAstrosKriginge",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"q"), gradient("DIAstrosKrigingeg1")) ),
      evaluation("EulerExact", "DIAvusExacte",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"static pressure", "mach"), gradient("DIAvusExacteg1")) ),
      evaluation("EulerSOA", "DIAvusSOAe",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"static pressure", "mach"), gradient("DIAvusSOAeg1")) ),
      evaluation("EulerMOA", "DIAvusMOAe",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"static pressure", "mach"), gradient("DIAvusMOAeg1")) ),
      evaluation("EulerKriging", "DIAvusKriginge",
        differentiation(wrt(names(loop("i",1,40),"Lpus$i$"),"static pressure", "mach"), gradient("DIAvusKrigingeg1")) )
    )
  ),
dependentVars(loop("i",1,40),"Lpus$i$",
realization(
    evaluation("LinearPotentialExact", "LpusAstrosExacte",
        differentiation(wrt(names(loop("k",1,20),"beta$k$"), "alpha", "q"), gradient("Lpus$i$AstrosExacteg1"))),
    evaluation("LinearPotentialSOA","Lpus$i$AstrosSOAe",
        differentiation(wrt(names(loop("k",1,20),"beta$k$"), "alpha", "q"), gradient("Lpus$i$AstrosSOAeg1"))),
    evaluation("EulerExact","LpusAvusExacte",
        differentiation(wrt(names(loop("k",1,20),"beta$k$"), "alpha", "static pressure", "mach"), gradient("Lpus$i$AvusExacteg1"))),
    evaluation("EulerSOA","Lpus$i$AvusSOAe",
        differentiation(wrt(names(loop("k",1,20),"beta$k$"), "alpha", "static pressure", "mach"), gradient("Lpus$i$AvusSOAeg1")))
)
),

objectiveVars(var("Dlo", "DI", Target.min)),
constraintVars(var("LTc", "LT", Relation.eq, 1000.0))
Configuring the Variables in the Model

Consider the necessary Evaluators for $D_{I.EulerExact}$

$$D_{I.EulerExact} = D_I(Lpus_{i.EulerExact} (\beta, \alpha, p_0, M, \gamma), M, p, \gamma)$$

Configure the $Lpus_{i.EulerExact}$ Evaluators

Configure the $D_{I.EulerExact}$ Evaluators

Method Evaluator “evaluateIDrag”

$$D_I(Lpus_{i.EulerExact}) = Lpus_{i.EulerExact}(\beta, \alpha, p_0, M, \gamma)$$

$$D_I = \frac{-1}{8\pi q} \sum_{i=1}^{n} Lpus_i \Delta y \sum_{k=1}^{n}[Lpus_k - Lpus_{k+1}] \left[ \frac{1}{y_i - y_k} - \frac{1}{y_i + y_k} \right]$$
Simple SORCER Network Configuration for Example

SORCER

NMEDA – Linux Beau Wolf Cluster

AVUS Provider
Euler Lpus Calculations

Client

MacDNA – MACBookPro

Gladius – Linux Desktop

Astros (LPM) Lpus Calculations
Optimization Model – SOA, MOA, FFD Calculations

Once the Variables & Model are configured Anywhere on the network the following API will work.

API Example 1:

```java
// set “beta1” to 5.0
Omodel.getVar("beta1").setValue(5.0);
Omodel.getVar("Lpus1").getValue();
```

Behavior - kick off Lpus calculation (run the Avus Task) due to dependency of “Lpus1” on “beta1” (observe – observable pattern)

```java
Omodel.getVar("Lpus1").getValue();
```

Behavior – Will NOT kick off Lpus calculation (run the Avus Task) since all “Lpusi” use the same evaluator(only different filters) and none of its dependencies have changed.
API Example 2:

// set “beta12” to -8.0

Omodel.getVar("beta12").setValue(-8.0);
Omodel.getVar("DI").getValue();

Behavior - kick off Lpus calculation (run the Avus Task) due to implicit dependency of “DI” on “beta12” (observe – observable pattern)
Omodel.getVar("iDrag").getDerivative("Euler Exact", "DIAvusExacteg1", "beta1", "beta2", "beta15")

Behavior - kicks off chain rule differentiation for

\[
\frac{\partial D_i}{\partial \beta_1} = \sum_{i=1}^{\# Lpus} \frac{\partial D_1}{\partial Lpus_i} \frac{\partial Lpus_i}{\partial \beta_1}
\]

\[
\frac{\partial D_i}{\partial \beta_2} = \sum_{i=1}^{\# Lpus} \frac{\partial D_1}{\partial Lpus_i} \frac{\partial Lpus_i}{\partial \beta_2}
\]

\[
\frac{\partial D_i}{\partial \beta_{15}} = \sum_{i=1}^{\# Lpus} \frac{\partial D_1}{\partial Lpus_i} \frac{\partial Lpus_i}{\partial \beta_{15}}
\]

\[
\frac{\partial Lpus_i}{\partial \beta_1}, \frac{\partial Lpus_i}{\partial \beta_2}, \frac{\partial Lpus_i}{\partial \beta_{15}}
\]

by FFD Evaluators run the Avus Task – Note only 7 Avus runs done )

\[
\frac{\partial D_1}{\partial Lpus_i}
\]

done analytically by a Method Evaluator
MSTC SORCER Applications to Date

- **Hi-Fidelity Aeroelastic Analysis**
  - Euler (AVUS), FEM (MSCNASTRAN), MDICE

- **Hi-Fidelity Induced Drag minimization**
  - Euler (AVUS), Optimization (CONMIN, MATLAB)

- **2-D large amplitude airfoil motion trajectory Optimization**
  - Unsteady Vortex Lattice Method (in-house UVLM), Opti (DOT)

- **Shape and sizing aeroelastic optimization**
  - Euler (AVUS), FEM (MSCNASTRAN), MDICE, Opti (DOT) FD sensitivities

- Demonstrated tight integration with C, C++, and FORTRAN using JNA (can make direct calls to any shared object)

- **Network configuration**
  - Linux workstations
  - Linux cluster(2), Mac cluster(1)
  - SGI Irix
  - Windows
  - Mac Desktop
  - Mac laptop
**Concluding Remarks**

- Very Brief Overview of SORCER that supports *Large Scale Distributed* MDMFMS-AO-UQ

- Developed Computational Environment (Variable, Filter, Evaluator and Model) to support Functions of Functions and Multi-Fidelity Analysis/Sensitivities.

- Optimization algorithms implemented through event driven architecture (Not presented)

- Java Spaces utilized for “pull” execution(not presented)
Questions?

Some AFRL Identified Research Topics in MDO

- Nonlinear analysis in Design
- Dynamic analysis in Design
- Multi-Fidelity Analysis in Design
- Multi-Disciplinary Analysis in Design
- Multi-Scale analysis in Design
- ROM
- Modeling for Design
- Simultaneous Shape, Sizing, Topology Optimization
- Large Scale Distributed MDO
- UQ in design
- ..
Questions