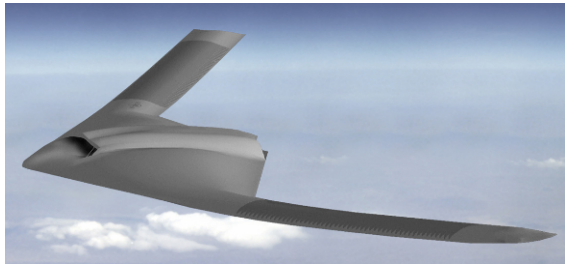
Service Oriented Computing EnviRonment (SORCER)

R. Kolonay
M. Sobolewski

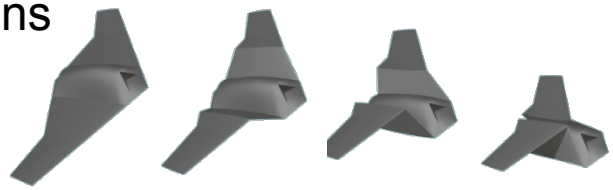
AFRL Air Vehicles Directorate
Revolutionary Aerospace Vehicles

AFRL Air Vehicles Directorate Multidisciplinary Sciences & Technology Center

Extreme Environments

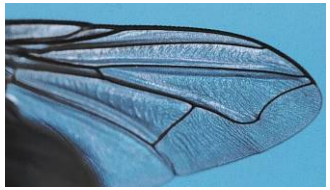


Large Deformations



Time Varying Shape

MAVs



Higher, Farther, Faster to
Lower, Closer, Slower

How to Capture the Physics Driving the Design Pre-Milestone A

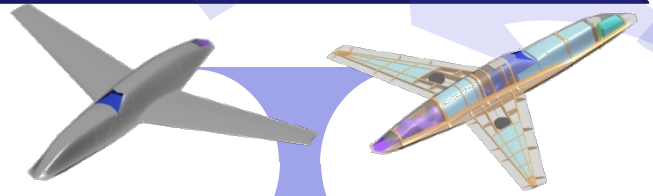
10000s configurations

Conceptual Design Studies

Technology Suite

Prototype Representation

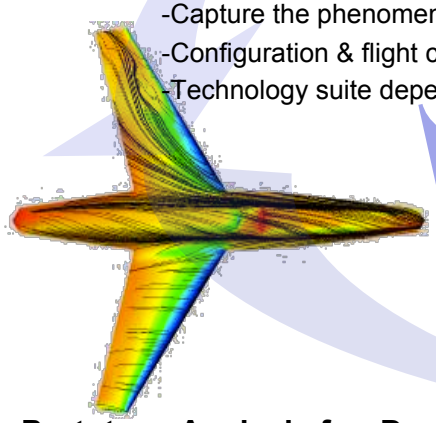
10s configuration



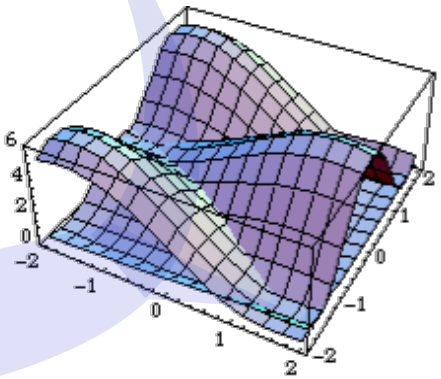
Dialable Fidelity

-Capture the phenomena driving the design
-Configuration & flight condition dependent
-Technology suite dependent

Prototype Experimental Validation



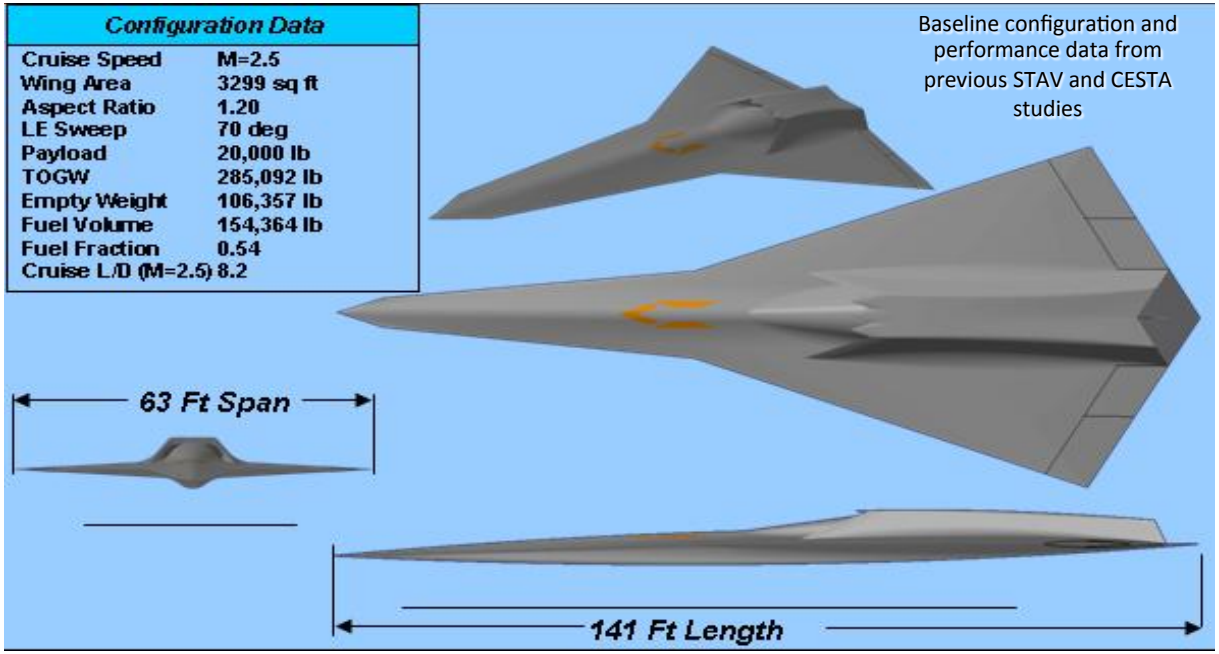
Prototype Analysis for Design



Prototype Design Space Exploration

Get more (and Better) Information … and get it Earlier

Typical Target Application LRS Configuration



SOO Requirements:

Range = 4000 nmi

Payload = 5-10 % weight fraction

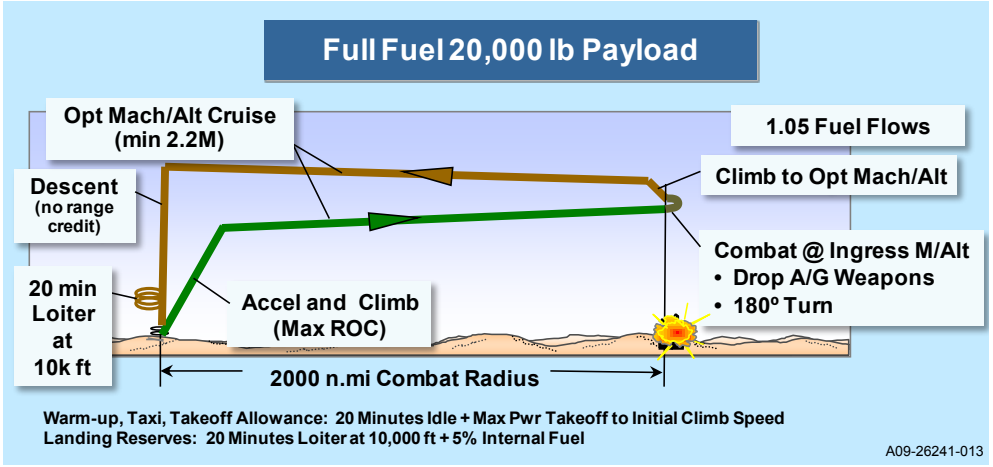
Maneuver Loads at Cruise – 2.5 g

Cruise speed: Mach 2.0

Cruise L/D: 8.5-9

Level 1 Flying Qualities

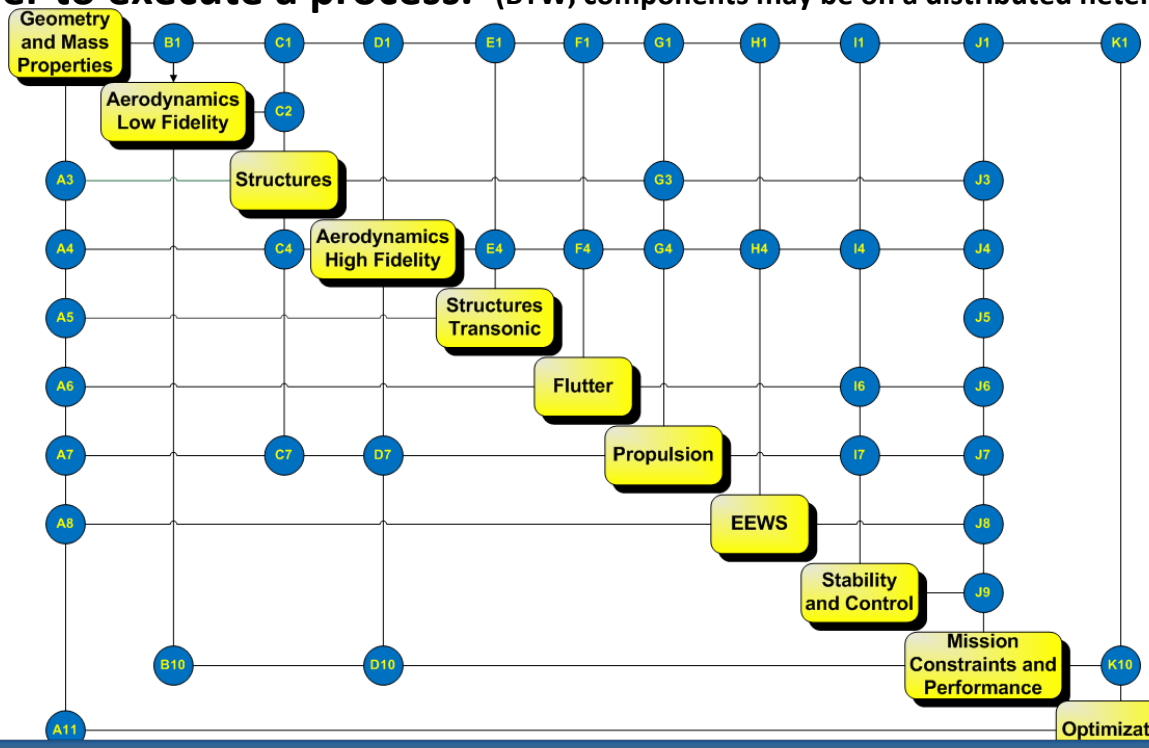
AF-specified mission will be "flown" through mission analysis to determine required fuel weight and TOGW



System Level MDMFMSAOwUQ

"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)



Need a Computational Environment to Achieve BCA MDMFMSAOwUQ

Need to Maximize Reuse

High Level Requirements for System Level MDMFMSAOwUQ

- # of components/applications/Services – 100's to 1000's
- Run times of services – secs to many days
- Data
  - kilobytes to terabytes
  - ascii, binary, databases
- Distributed (across organizational boundaries) heterogeneous computing environment
  - Hand held devices to HPC resources
  - Seamless access to data and services
  - Process representatioin with secure communications

SORCER is not for the Faint at Heart

Experience/Knowledge Needed

- *MDMFMSAOwUQ* *(MultiDisciplinary,MultiFidelity,MultiScale, Analysis/Optimizaiton with Uncertainty Quantification)*
- Eclipse (or similar IDE)
- Ant
- Java
- JNA, JNI, SWIG (wrapping native code)
- Network Computing
- Object Oriented Concepts
- SORCER Concepts

SORCER is a Service-Object-Oriented Architecture

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 Philosophy – Accessibility, Flexibility & Reusability

SORCER is Research Code!

Service Oriented Computing EnviRonment - SORCER

A service-oriented product development environment

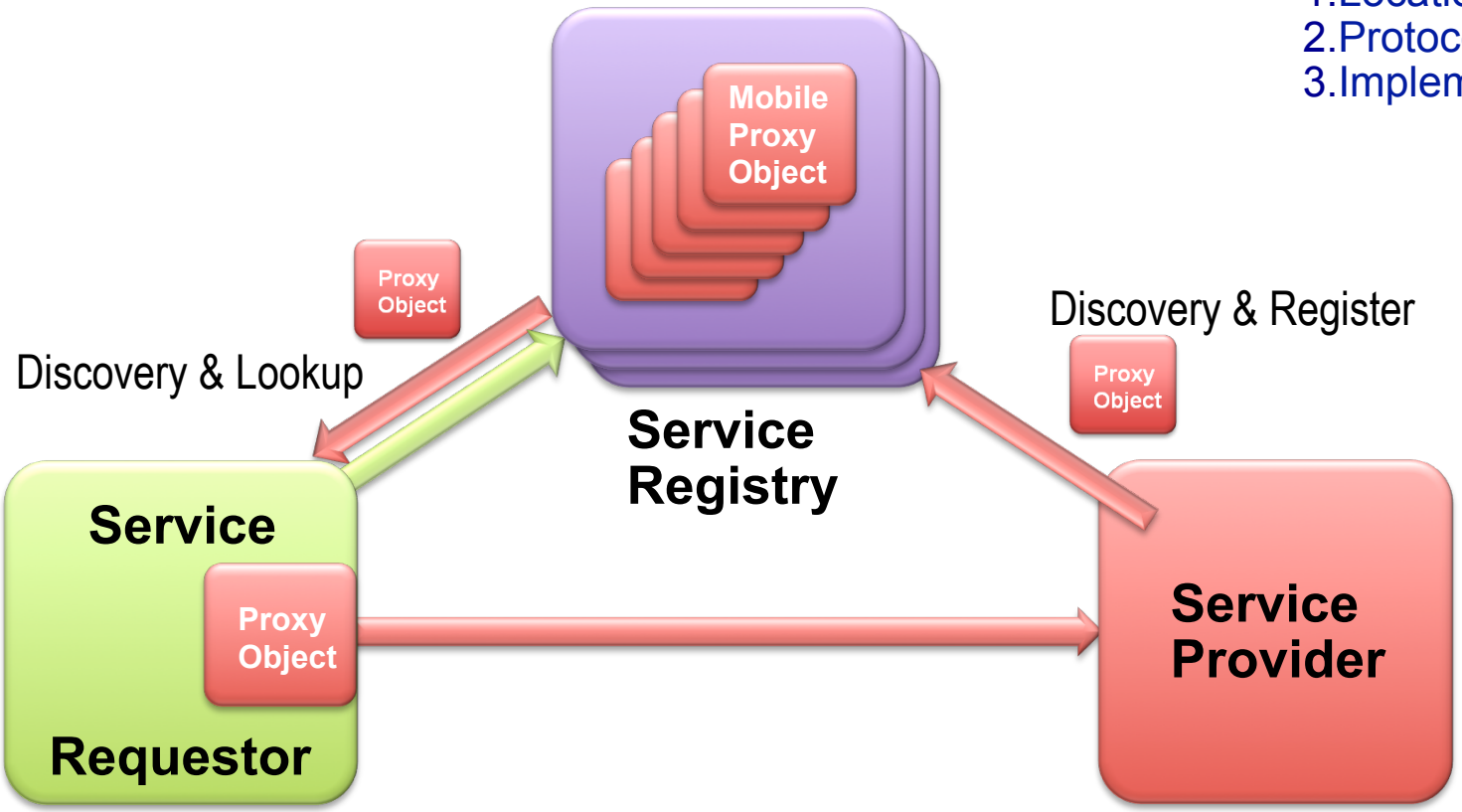
- **SORCER** is a programming and **computing environment** that enables one to perform large scale system level engineering analysis and design space exploration that may be in a distributed heterogeneous computing environment.
- SORCER federates a series of ***Service Providers*** (which may be distributed) in real time and **orchestrates the communication** between the ***Service Providers*** based on a ***Control Strategy*** **algorithm** defined in an ***Exertion****(process definition)* to perform a multi-disciplinary analysis and or design space exploration.

Service Oriented Computing EnviRonment - SORCER

A network-based distributed framework

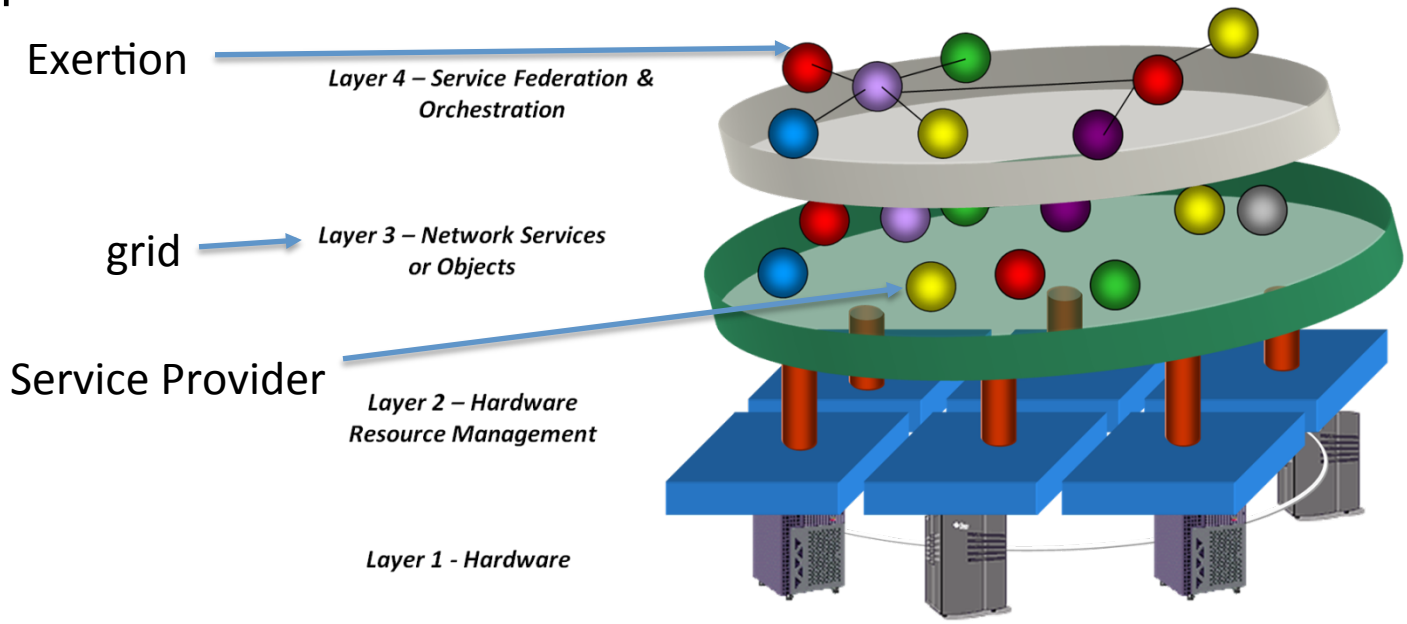
Service Oriented

Neutrality:
1.Location
2.Protocol
3.Implementation

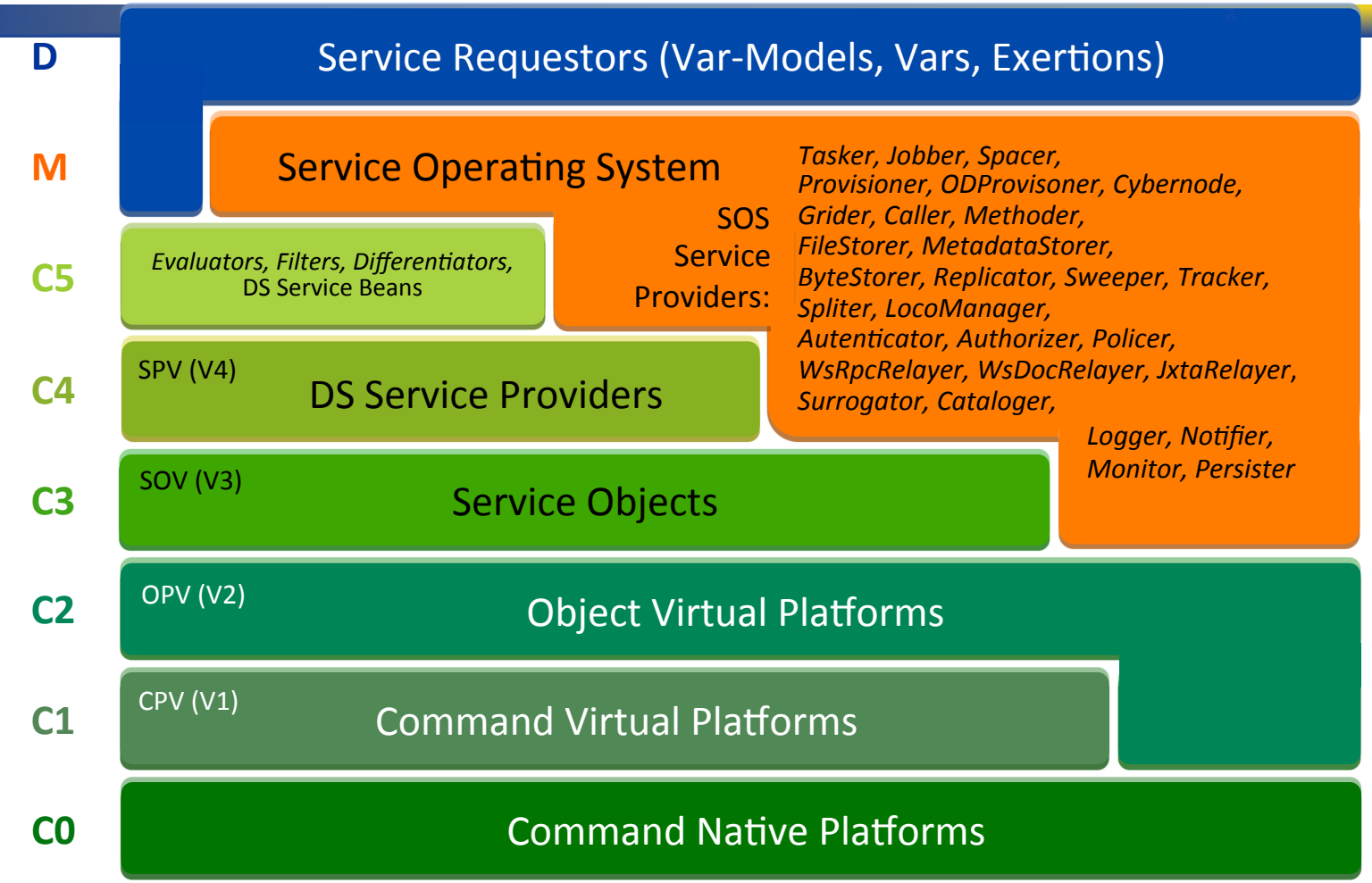


SORCER Terminology

- ***Service Provider*** : a remote object accepting ***Exertions*** from service requestors and performs calculations. Can provide one or more services.
- The ***grid***: 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.



SORCER SO Platform

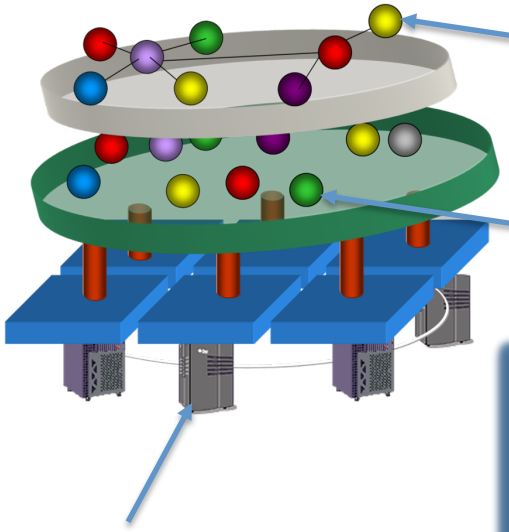


Service requestors (exertions) – commands of the SO processor (C0-C5)

SO program – an exertion executed by the SOS shell

SV – service virtualization, PV – platform virtualization

From Computing to Metacomputing (Exertion Oriented Programming)



Metaplatform

Provider

Platform

Exertion Oriented Programming

SORCER Meta-OS

Metaprocessor (Services)

Service Context

Service Management

Service Implementation

Programming Env

OS

Processor

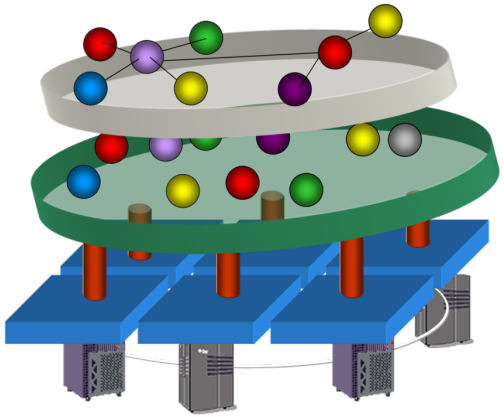
SORCER Model & Variables

- ***Models –*** consists of ***Variables, Filters*** and ***Evaluators***.
  - Current available models – ResponseModel(including sensitivities) , ParametricModel,, OptimizationModel
- ***Variables (Var) -*** 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.
- ***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
- ***Filters -*** are used to map the results of *Evaluators* to *Variable* Values. Think unix shell piping. Filters can be concatenated *n* times.
  - Current Filter Types – BasicFileFilter, ContextFilter, FileFilter, GrepFilter, ListFilter, MapFilter, ObjetFilter, PatternFilter, TextFilter

All are Objects

VFEEx Paradigm (Variable Based Computing)
Value-Filter-Evaluator-Exertion

SORCER



Variable

Value

Filters

Evaluators

Exertion (collaboration)

Data Context

Control Context

Signatures

Metaplatfom

Metaprograms

SORCER Meta-OS

Metaprocessor

Provider (computation)

Service Context

Service Management

Service Implementation

Platform

Programs

OS

Processor

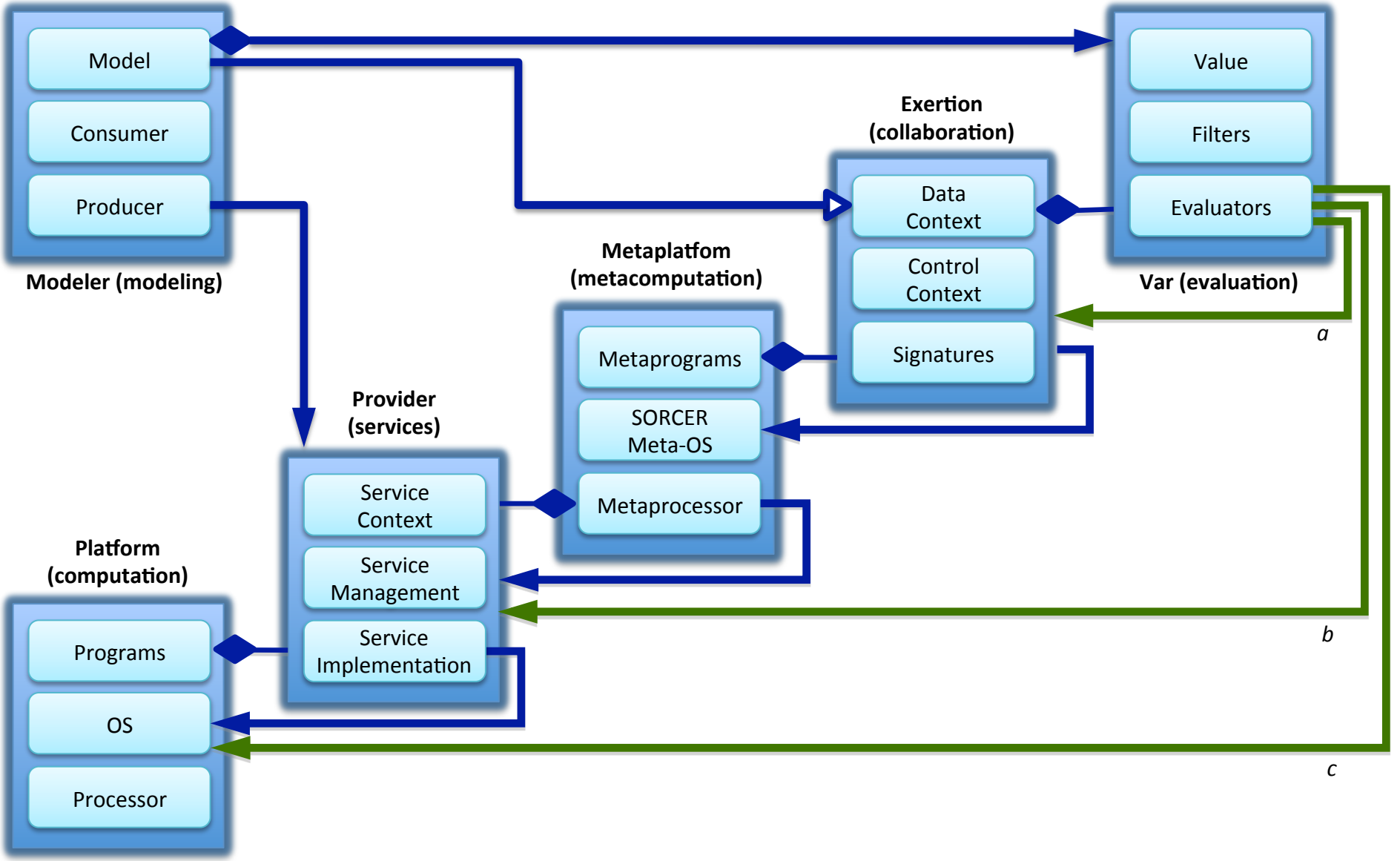
a

b

c

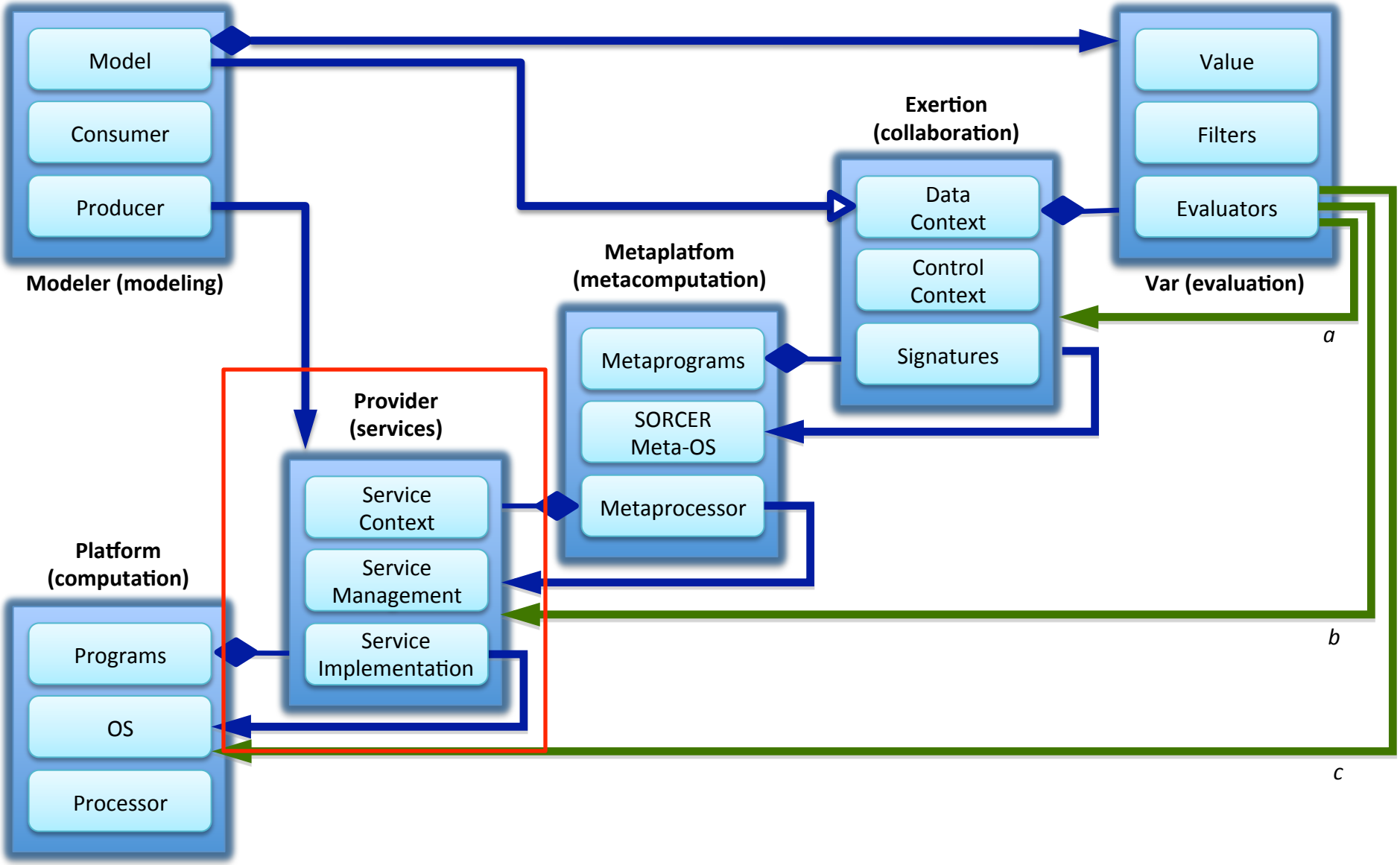
Model Based Computing

SORCER



Model Based Computing

SORCER



Service Providers

- Wrap existing applications with java to expose one or more functionalities(services) of the application to the SORCER Environment

```
Public class xxxProvider extends
ServiceProvider implements
xxxRemoteInterface, SorcerConstants{ }
```

- Tight integration with c, c++, fortran etc.. using jna, jni and swig. Allows interaction at the api level not at the file level. More efficient and robust.

Robustness & Reuse
Your Computational Environment is only as good as your Wrapping!

Service Provider Implementation Class

(engineering.provider.astros.AstrosProviderImpl)

```
// Astros Provider Impl Class
public class AstrosProviderImpl extends ServiceProvider
   implements AstrosRemoteInterface,
   AstrosBoundaryConditionRemote, SorcerConstants {
…
…
..
}
```

Astros RemoteInterface

(engineering.provider.astros.AstrosRemoteInterface)

```
package engineering.provider.astros;
import sorcer.core.*;
import sorcer.util.*;
import java.rmi.*;
/**
 * Generic Remote Interface for Astros Services
 * providers.
 * @author       R. M. Kolonay
 * @version      %I%, %G%
 * @since        JDK1.4
 */
public interface AstrosRemoteInterface extends
   AstrosInterface, Remote{
}
```

AstrosInterface

(engineering.provider.astros.AstrosInterface)

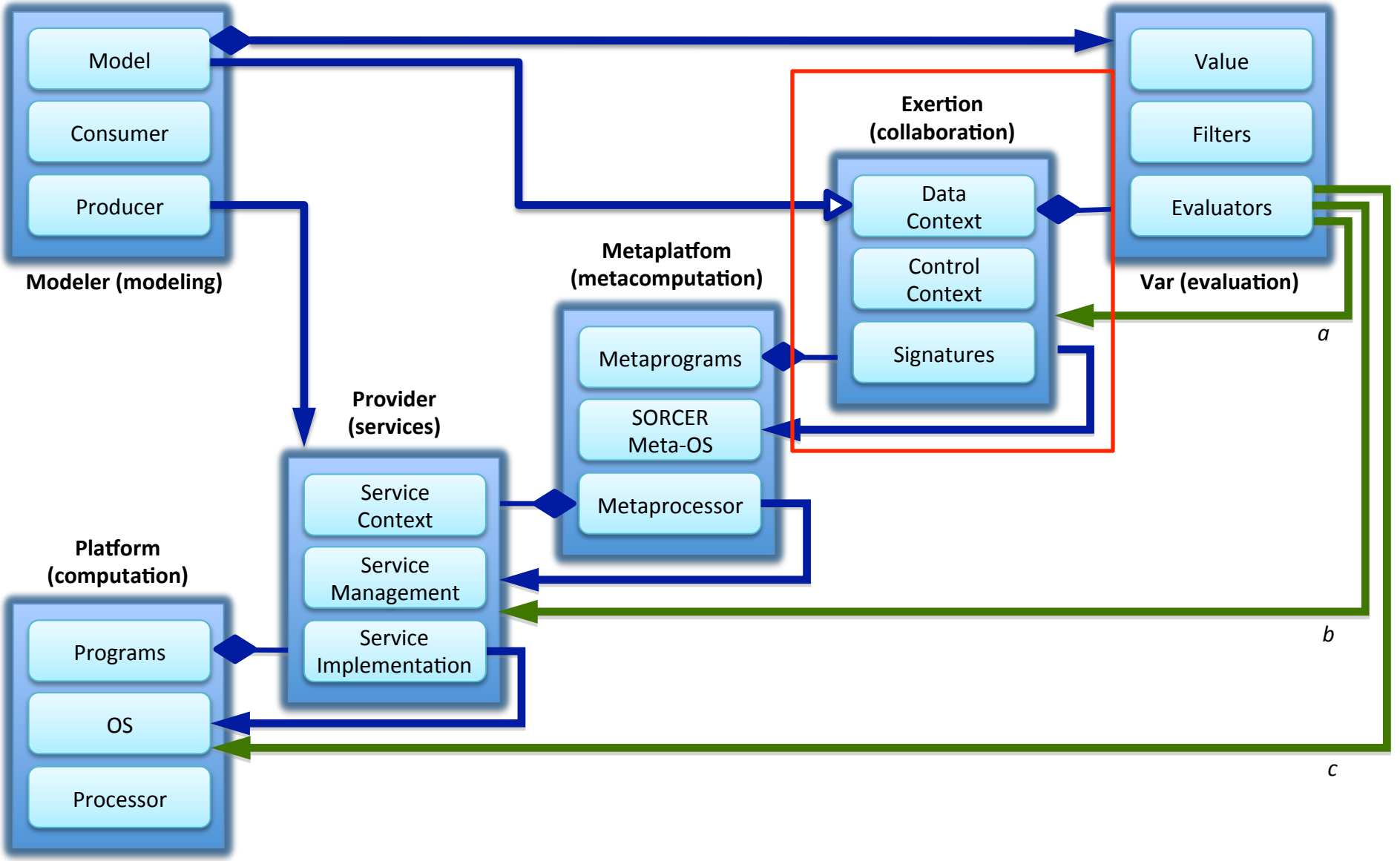

```
package engineering.provider.astros;

import java.rmi.RemoteException;
import sorcer.core.context.ServiceContext;
import sorcer.service.Context;

/**
 * Generic Interface for Astros Services
 * providers.
 * @author       R. M. Kolonay
 * @version      %I%, %G%
 * @since        JDK1.4
 */
public interface AstrosInterface {

    public Context executeAstros(Context context)
        throws RemoteException;

    public Context computeLiftPerUnitSpan(Context context)
    throws RemoteException;

    public Context computeFlutterDamping(Context context)
    throws RemoteException;
}
```

Three Services made available by this Interface

- executeAstros
- computeLiftPerUnitSpan
- computeFlutterDamping

Model Based Computing

SORCER

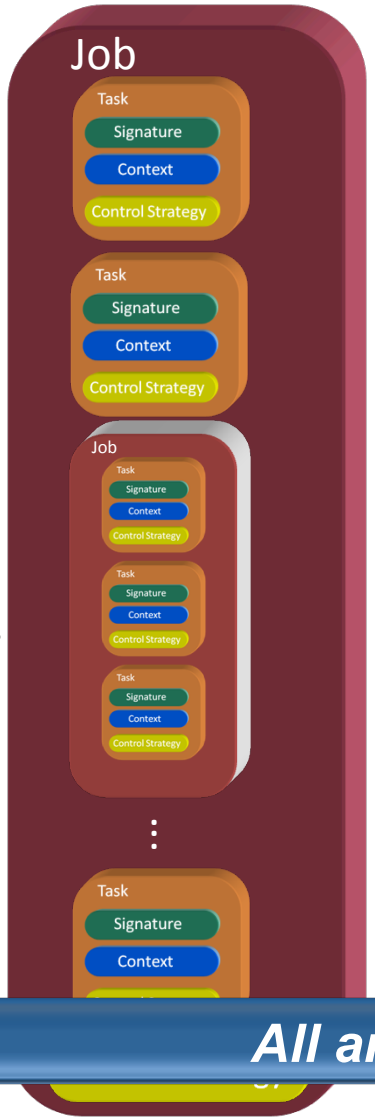


Exertion

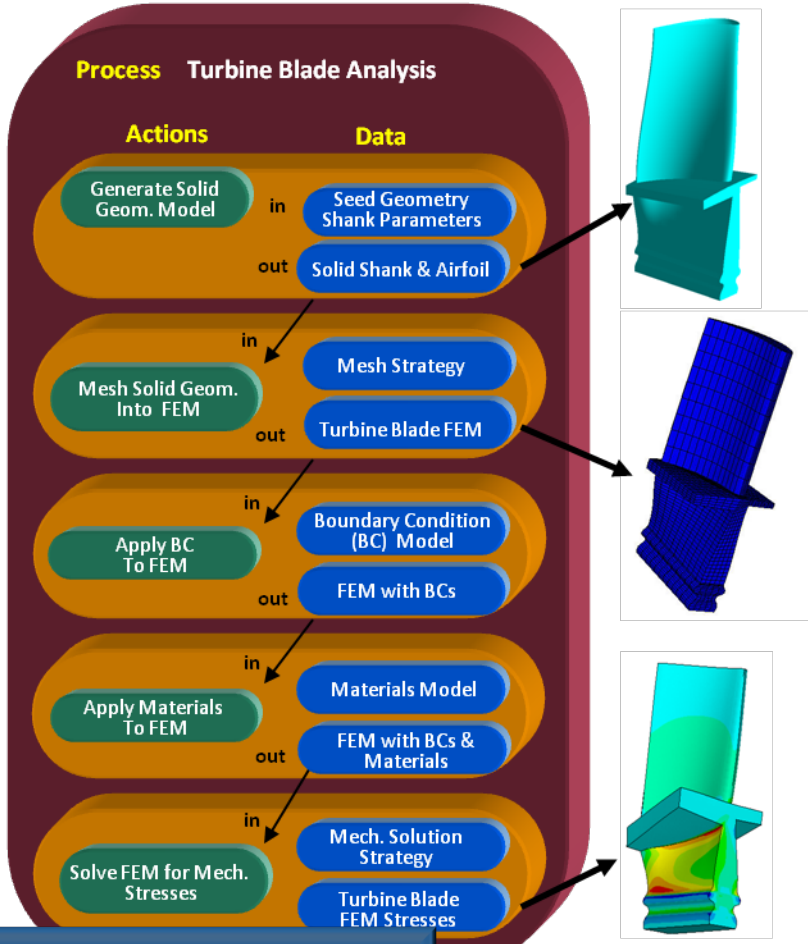
Elementary Exertion

Signature – identifies the Service
Context – contains input & output data/objects
Control Strategy – Guidance for Service

Composite Exertion



Engineering Example



All are Objects

Astros Exertion (Task)

(engineering.requestor.astros. AstrosRequestor)


```
// Create the URLs for the Astros input files
String urlBase = getWebsterUrl();
String inputDirName = getProperty("requestor.astros.input.path");
String astrosMainInputFileName = getProperty("requestor.inputfile");
String astrosbdf1InputFileName = getProperty("requestor.bdf1");
URL astrosMainInputURL = new URL(urlBase+"/"+inputDirName+"/"+astrosMainInputFileName);
URL astrosBulkData1URL = new URL(urlBase+"/"+inputDirName+"/"+astrosbdf1InputFileName);
```


```
// Create the Context for the task
Context context = new ServiceContext("AstrosContext","AstrosContext");
   Contexts.putInValue(context, IN_VALUE + CPS + "ASTROS/MAIN/INPUT", astrosMainInputURL, mainInputDT);
Contexts.putInValue(context, IN_VALUE + CPS + "ASTROS/BDF1/INPUT", astrosBulkData1URL, bdfInputDT);

// echo Context to System.out
logger.info("Astros Input Context: "+context);
```

```
// construct Method and then Task
   Signature methodEN = new ServiceSignature("executeAstros",   engineering.provider.astros.AstrosRemoteInterface.class,
                                "ENGINEERING-Astros-RMac");
```

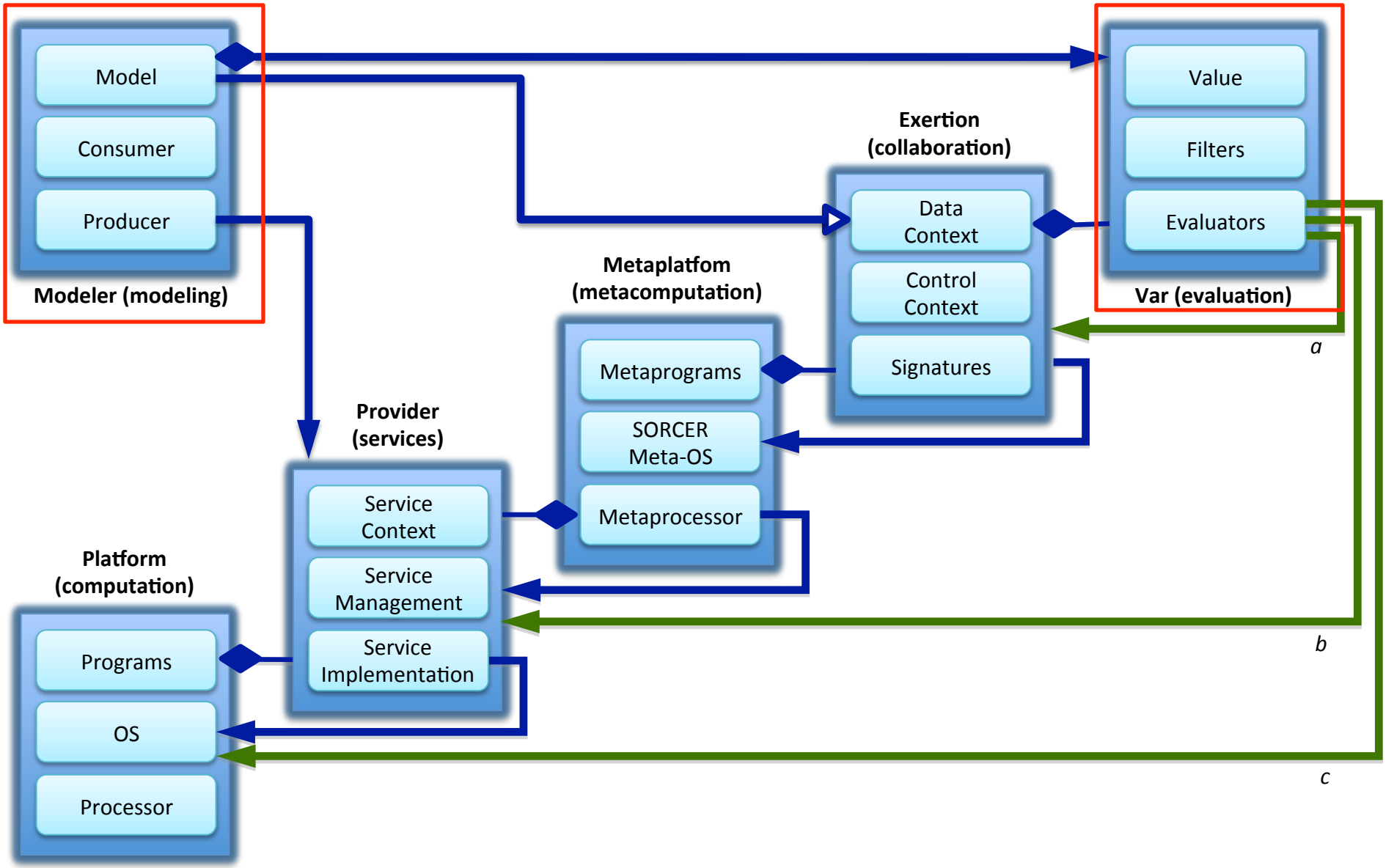
```
   ServiceTask taskEN = new ServiceTask("run Astros", "Task to run Astros", methodEN);

   // Put the context into the task
   taskEN.setContext(context);
```

```
   // Execute the task
taskEN = (ServiceTask)taskEN.exert(null);
// echo the returned task to the logger
 logger.info("Returned Task after exert: "+taskEN);
 context = taskEN.getContext();
```

"exert" – execute the task on the network

Model Based Computing

SORCER



SORCER Model & Variables

- ***Models –*** consists of ***Variables, Filters*** and ***Evaluators***.
  - 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.
- ***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
- ***Filters -*** are used to map the results of *Evaluators* to *Variable* Values. Think unix shell piping. Filters can be concatenated *n* times.
  - Current Filter Types – BasicFileFilter, ContextFilter, FileFilter, GrepFilter, ListFilter, MapFilter, ObjetFilter, PatternFilter, TextFilter

All are Objects

SORCER Variables

SORCER Variables are distinguished by four attributes: ***type, kind, valueType,*** and ***mathType***

SORCER Variable ***type*** (only one)

- DESIGN (DEFAULT)
- RESPONSE

SORCER Variable ***kind*** (one or more combinations)

- LINKED
- PARAMETER
- BOUNDED
- RANDOM
- CONSTRAINT
- OBJECTIVE
- INDEPENDENT

SORCER Variable ***ValueType*** (only one)

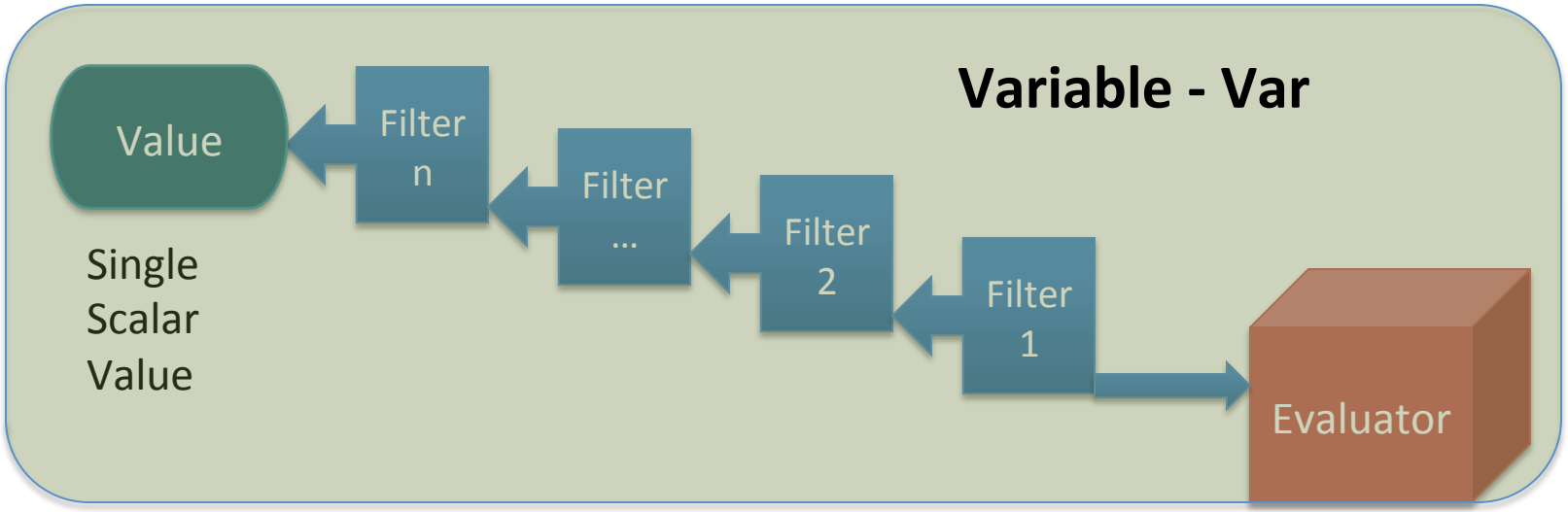
- INTEGER
- LONG
- DOUBLE (DEFAULT)
- FLOAT
- STRING
- OBJECT
- UNKNOWN
- UNDEFINED

SORCER Variable ***mathType*** (one or more combinations)

- CONTINUOUS (DEFAULT)
- DISCRETE
- DISCRETE_WITH_ORDER
- DISCRETE_WITH_MATH
- DISCRETE_NO_ORDER
- PROBLEM_PARAMETER
- REAL

Example: skinThickness: Type:DESIGN, Kinds:BOUNDED,RANDOM, ValueType:DOUBLE, MathType: REAL, CONTINUOUS

Variable - Value, Filters, Evaluators



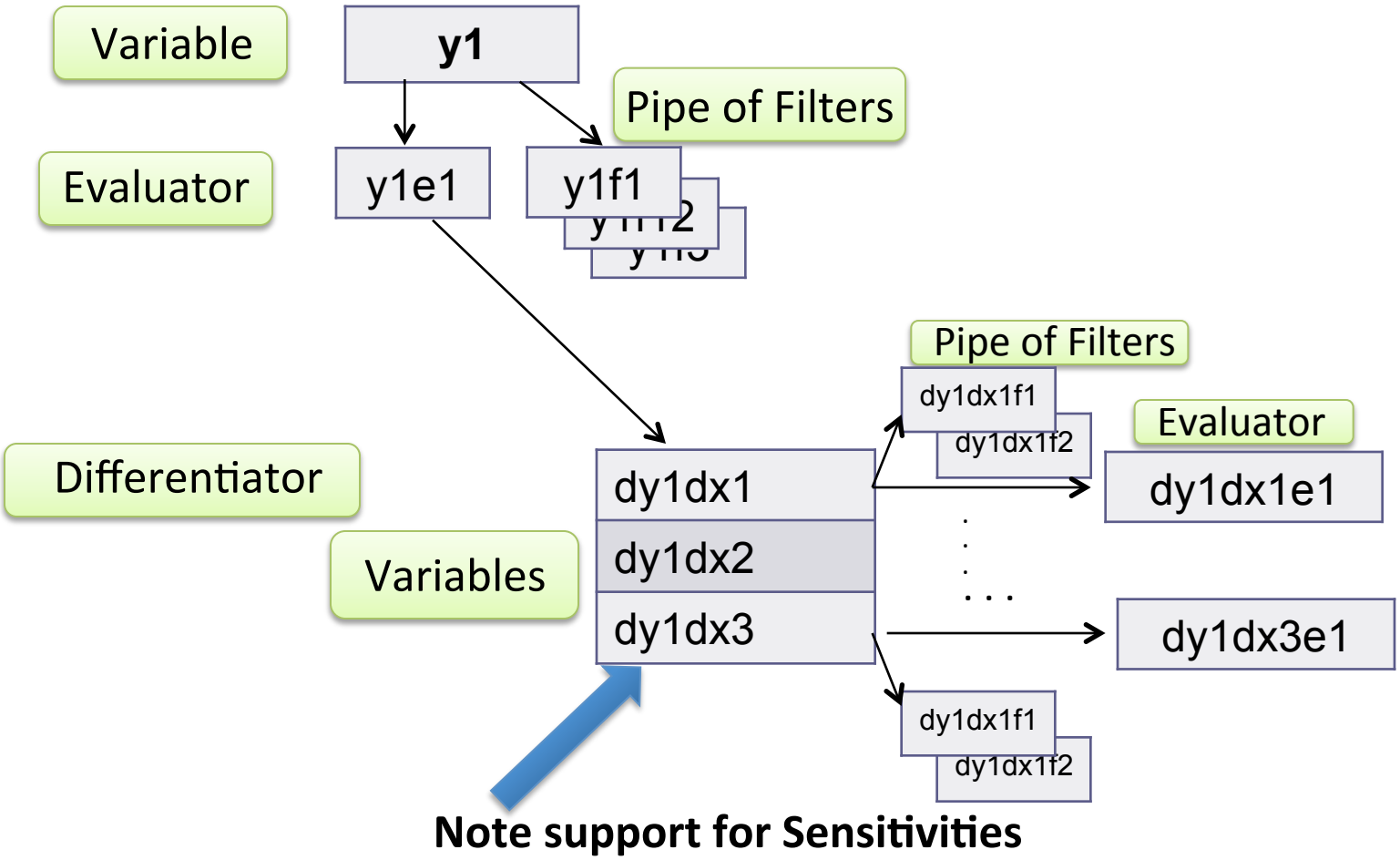
Computational Entity. Creating Large Amounts of Data

Variable, Evaluator, Filter (VEF) Design Patterns

- Lazy variable evaluation
  - getValue vs. evaluate (**demand driven calculations**)
- Evaluator – Decorator pattern
  - Evaluator: getValue/evaluate implements `EvaluationManagement`
  - Outer evaluator calls on its inner one
  - Evaluator is observable and observer (**functional programming**)
- Observer-Observable
  - setChanged
  - notify/update

Basic Variable Structure

$$y_1(x_1, x_2, x_3)$$

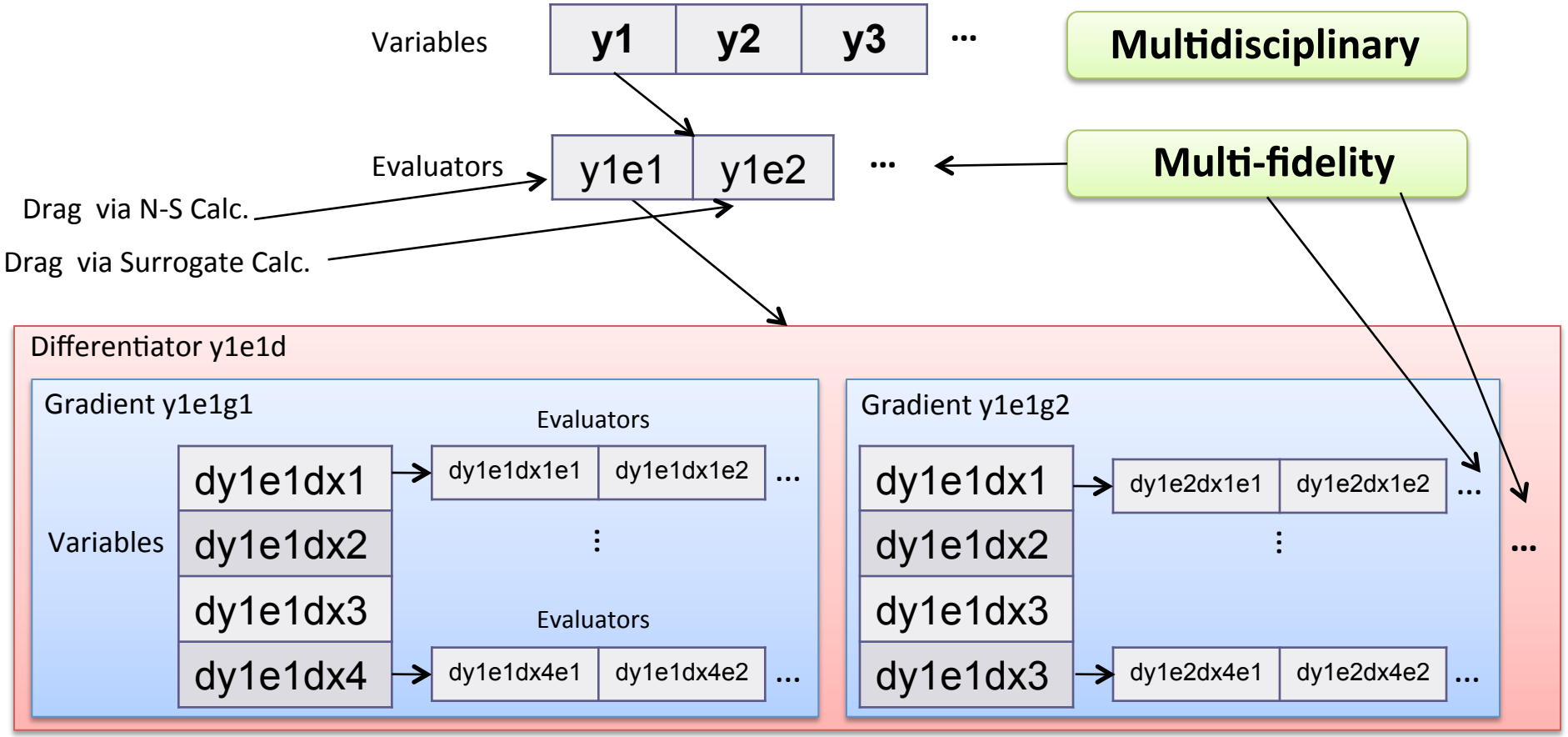


Advanced Variable Structure

SORCER

Functions of Functions

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

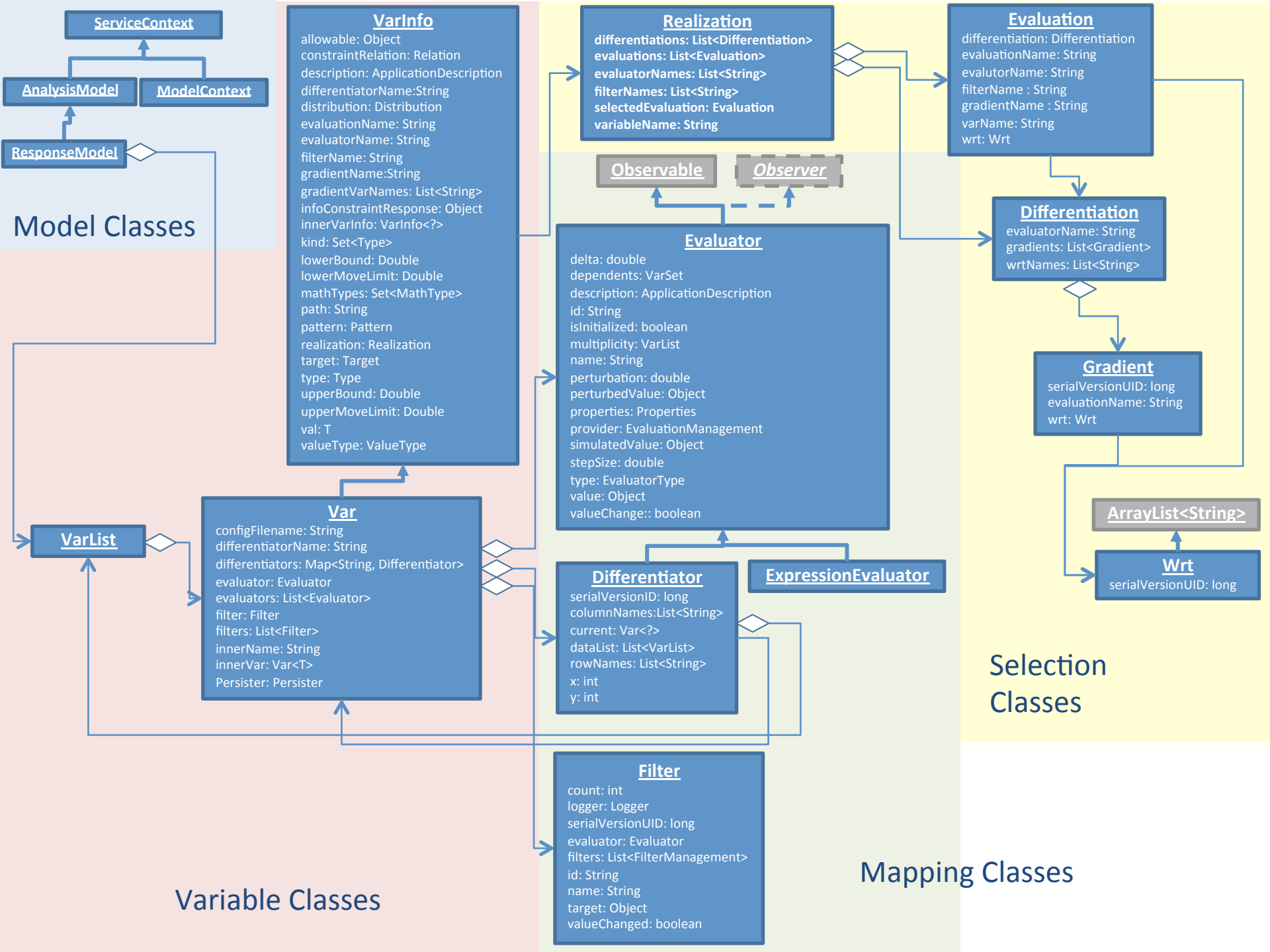


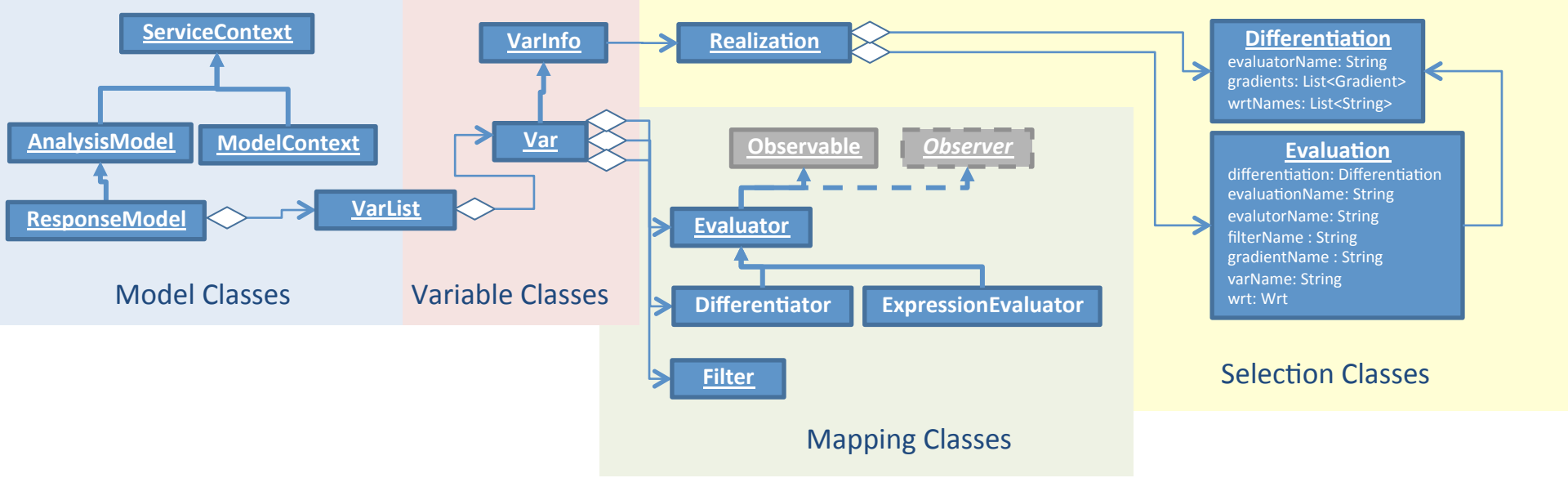
Evaluators

- ***Evaluators –*** are used to determine the value of variables and their partial derivatives(chain rule works) with respect to their dependencies (Variables). Evaluators do not necessarily produce a scalar value. The result can be an entire database or file. (for a given set of input an application creates many outputs)
  - Current Evaluator Types – ModelEvaluator, ExertionEvaluator, ExpressionEvaluator, GroovyEvaluator, JepEvaluator, MethodEvaluator, SOAEvaluator, FDEvaluator

Filters

- ***Filters -*** are used to map the results of *Evaluators* to *Variable* Values. Think unix shell piping. Filters take the output of Evaluators and reduce it to a single scalar value necessary to define a *Variable* value. Filters can be concatenated *n* times.
  - Current Filter Types – BasicFileFilter, ContextFilter, FileFilter, GrepFilter, ListFilter, MapFilter, ObjetFilter, PatternFilter, TextFilter


ServiceContext
AnalysisModel
ModelContext
ResponseModel
Model Classes
VarInfo
allowable: Object
constraintRelation: Relation
description: ApplicationDescription
differentiatorName:String
distribution: Distribution
evaluationName: String
evaluatorName: String
filterName: String
gradientName:String
gradientVarNames: List<String>
infoConstraintResponse: Object
innerVarInfo: VarInfo<?>
kind: Set<Type>
lowerBound: Double
lowerMoveLimit: Double
mathTypes: Set<MathType>
path: String
pattern: Pattern
realization: Realization
target: Target
type: Type
upperBound: Double
upperMoveLimit: Double
val: T
valueType: ValueType
Realization
differentiations: List<Differentiation>
evaluations: List<Evaluation>
evaluatorNames: List<String>
filterNames: List<String>
selectedEvaluation: Evaluation
variableName: String
Evaluation
differentiation: Differentiation
evaluationName: String
evalutorName: String
filterName : String
gradientName : String
varName: String
wrt: Wrt
Observable
Observer
Evaluator
delta: double
dependents: VarSet
description: ApplicationDescription
id: String
isInitialized: boolean
multiplicity: VarList
name: String
perturbation: double
perturbedValue: Object
properties: Properties
provider: EvaluationManagement
simulatedValue: Object
stepSize: double
type: EvaluatorType
value: Object
valueChange:: boolean
Differentiation
evaluatorName: String
gradients: List<Gradient>
wrtNames: List<String>
Gradient
serialVersionUID: long
evaluationName: String
wrt: Wrt
ArrayList<String>
Wrt
serialVersionUID: long
VarList
Var
configFilename: String
differentiatorName: String
differentiators: Map<String, Differentiator>
evaluator: Evaluator
evaluators: List<Evaluator>
filter: Filter
filters: List<Filter>
innerName: String
innerVar: Var<T>
Persister: Persister
Differentiator
serialVersionID: long
columnNames:List<String>
current: Var<?>
dataList: List<VarList>
rowNames: List<String>
x: int
y: int
ExpressionEvaluator
Filter
count: int
logger: Logger
serialVersionUID: long
evaluator: Evaluator
filters: List<FilterManagement>
id: String
name: String
target: Object
valueChanged: boolean
Selection Classes
Variable Classes
Mapping Classes



Mapping Classes

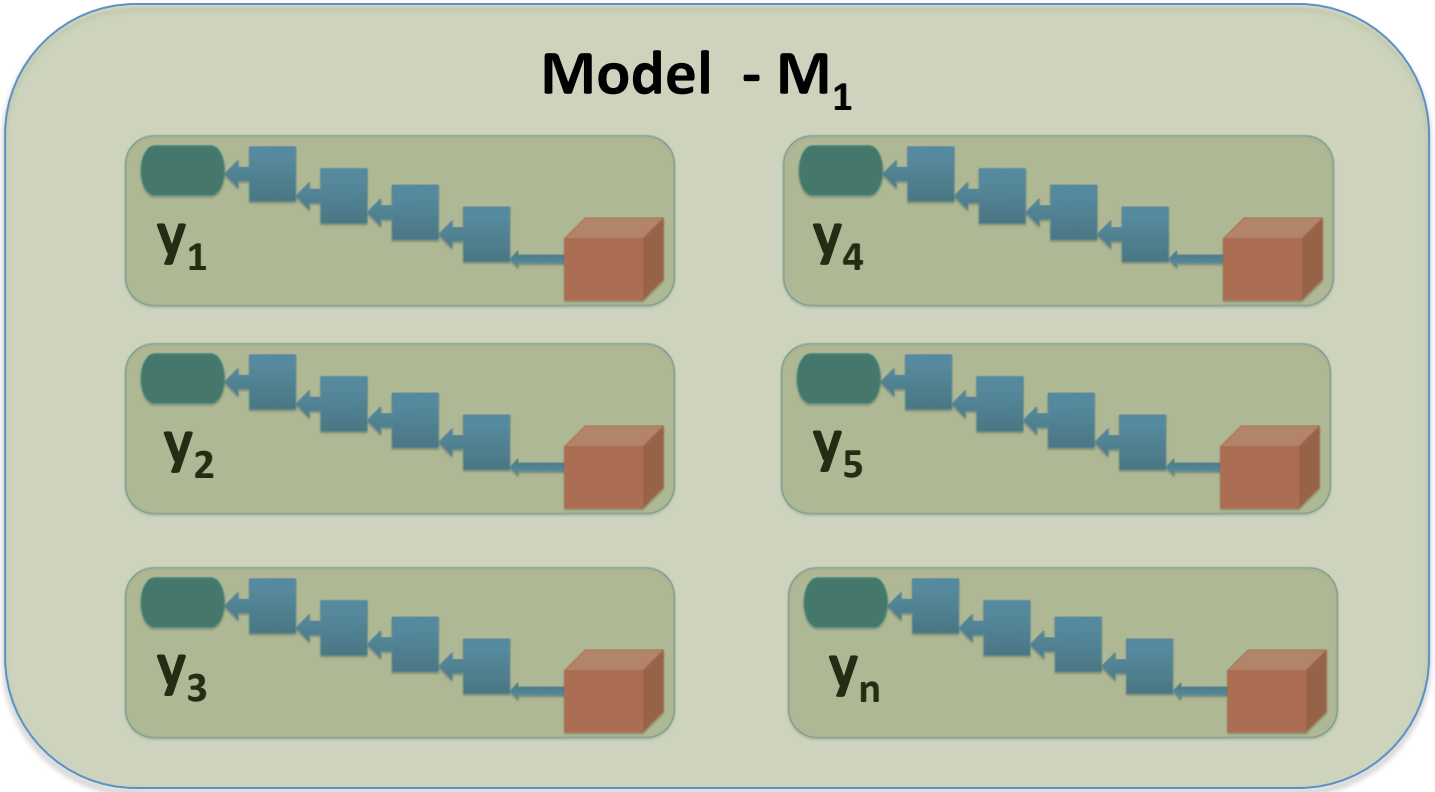
SORCER Models

SORCER Models support

- Analysis – (ResponsModel & ParametricModel)
  - multidisciplinary response analysis
  - multi-fidelity response analysis
  - multi-fidelity response sensitivities

- Design Space Exploration (OptimizationModel)
  - Optimization
  - Multidisciplinary Optimization

Model is a Collection of Variables



Model Construction

- Constructing a Model Consists of Two Steps
  - Model Definition (Skeleton)
    - Defines the model Variables (design, response, 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)

Model Construction Example
Rosen-Suzuki Functions Definition

Independent Varibles - $\{x_1, x_2, x_3, x_4\}$

Functions

$$f(x_1, x_2, x_3, x_4) = x_1^2 - 5x_1 + x_2^2 - 5x_2 + 2x_3^2 - 21x_3 + x_4^2 + 7x_4 + 50$$

$$g_1(x_1, x_2, x_3, x_4) = x_1^2 + x_1 + x_2^2 - x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0$$

$$g_2(x_1, x_2, x_3, x_4) = x_1^2 - x_1 + 2x_2^2 + x_3^2 + 2x_4^2 - x_4 - 10.0$$

$$g_3(x_1, x_2, x_3, x_4) = 2x_1^2 + 2x_1 + x_2^2 - x_2 + x_3^2 - x_4 - 5.0$$

Model Definition
(using functional programming operators)

`sorcer.rs.ex7.model.RsResponseModelBuilder` examples/ex7

Model Definition (using functional programming operators)

```
// define the model
model = responseModel("Rosen-Suzuki Response Model",
                designVars(vars(loop("i", 1,4),"x$i$")),
                responseVar("f",
                                realization(
                                                evaluation("FExacte", "fe",null,null))),
                responseVars(loop("i", 1, 3),"g$i$",
                                realization(
                                                evaluation("g$i$Exacte", "g$i$e",null,null)))
        );
```

The above creates
ResponseModel: “Rosen-Suzuki Response Model”, which contains -
Vars: with names: “x1”,”x2”,”x3”,”x4”, type: DESIGN
Var: with name: “f”, type: RESPONSE, evaluationName:”Fexacte” , evaluatorName:”fe”
Vars: with names: “g1”,”g2”,”g3”, type: RESPONSE, evaluationName:”g1Exacte”, evaluatorName:”g1e”
evaluationName:”g2Exacte”, evaluatorName:”g2e”
evaluationName:”g3Exacte”, evaluatorName:”g3e”

Rosen-Suzuki Functions

`sorcer.rs.ex7.model.RsResponseModelBuilder` Examples/ex7

Model Configuration (using operators and explicit programming)

```
fe = evaluator("fe","x1^2-5.0*x1+x2^2-5.0*x2+2.0*x3^2-21.0*x3+x4^2+7.0*x4+50.0");
fe.addArgs(model.getDesignVars("x1", "x2", "x3", "x4"));
model.setResponseEvaluator("f", fe);
// an alternative approach would be to do it declaratively
//var(model,"f","fe",evaluator("fe","x1^2-5.0*x1+x2^2-5.0*x2+2.0*x3^2-21.0*x3+x4^2+7.0*x4+50.0"),args("x1", "x2", "x3", "x4"));

Evaluator g1e = evaluator("g1e", "x1^2+x1+x2^2-x2+x3^2+x3+x4^2-x4-8.0");
g1e.addArgs(model.getDesignVars("x1", "x2", "x3", "x4"));
model.setResponseEvaluator("g1", g1e);

Evaluator g2e = evaluator("g2e", "x1^2-x1+2.0*x2^2+x3^2+2.0*x4^2-x4-10.0");
g2e.addArgs(model.getDesignVars("x1", "x2", "x3", "x4"));
model.setResponseEvaluator("g2", g2e);

Evaluator g3e = evaluator("g3e", "2.0*x1^2+2.0*x1+x2^2-x2+x3^2-x4-5.0");
g3e.addArgs(model.getDesignVars("x1", "x2", "x3", "x4"));
model.setResponseEvaluator("g3", g3e);
```

Steps for Configuration: 1. Create Evaluator, 2. add Arguments/Dependencies to Evaluator, 3. Create Filter(s) if required, 4. Associate Evaluator with Var

Configuring a Var

In SORCER Vars have ***type*** either **DESIGN** (default) or **RESONSE. Depending on the *type,*** the configuration process is different.

type: **DESIGN –** No ***Evaluator*** is specified, a ***Filter*** may or may not be specified, ***kind***, ***valueType***, and, ***mathType*** may or may not be specified (defaults will occur).

In the Roszen-Suzuki Example the DESIGN Vars are “x1”, “x2”, “x3”, and “x4”. They do not require filters and hence no further configuration.

Configuring a Var in the Model

type: **RESPONSE –** ***Evaluator*** must be specified, a ***Filter*** (default **null)** may or may not be specified, ***Differentiator*** (default **null**) must be specified if derivatives are to be computed, ***evaluationName*** may or may not be specified – if it is specified it must match an ***evaluationName*** given in the model it is utilized within. ***kind***, ***valueType***, and, ***mathType*** may or may not be specified (defaults will occur).

Note: the ***evaluatorName*** given for the ***Evaluator*** must match a ***name*** given in the model it is utilized within.

In the Roszen-Suzuki Example for the **RESPONSE** ***Var*** "g1" an ***Evaluator***
g1e = evaluator("g1e", "x1^2+x1+x2^2-x2+x3^2+x3+x4^2-x4-8.0"); is created.
Note: that the ***evaluatorName*** *"g1e"* matches the ***evaluatorName*** given in the model definition.

The statement:
model.setResponseEvaluator("g1", g1e);
Actually associates the ***Evaluator g1e*** with the ***Var*** "g1" in the model. Note: No ***Filter*** or ***Differentiator*** is needed in this example. (they are set to ***null*** by default).

```
// define the model
model = responseModel("Rosen-Suzuki Response Model",
        designVars(vars(loop("i", 1,4),"x$i$")),
        responseVar("f",
                realization(
                        evaluation("FExacte", "fe",null,null))),
        responseVars(loop("i", 1, 3),"g$i$",
                realization(
                        evaluation("g$i$Exacte", "g$i$e",null,null)))
);
```

This could have been achieved with the following two statements
Var g1Var = Model.getVar("g1");
g1Var.setEvaluator(g1e);

Using the Model

- Models can be used in two ways
  - Intra – instantiating the class and using the available methods or “querying” the model using a context (the ***ModelContext***).
  - Inter – publishing the model as a service and “querying” the model using a context (the ***ModelContext***).

Rosen-Suzuki Functions

Model Use (using the methods in an instantiated class)

```
// you can set the value of the independent variables
rm.setDesignVarValues(set("x1", 2.0), set("x2", 3.0), set("x3", 4.0), set("x4", 200.0));

// get the values of the responses for the new x values
rm.getRepsonses();
Data Table [x1,x2,x3,x4,  f, g1,g2,g3]
           [2.0, 3.0, 4.0, 5.0,  46.,  44.0, 71.0, 24.0]

// additional examples of using the api
Var x4 = rm.getDesignVar("x4");
logger.info("\n\nx4 value: " + x4.getValue());
x4 value: 5.0

// examine the responseVar f
Var f = rm.getResponseVar("f");
ExpressionEvaluator ee = (ExpressionEvaluator) f.getEvaluator();
logger.info("\n\nexpression:\n" + ee.getName() + "/" + ee.getExpression());
fe/x1^2-5.0*x1+x2^2-5.0*x2+2.0*x3^2-21.0*x3+x4^2+7.0*x4+50.0
logger.info("\n\nresponse:\n" + ee.getName() + "/" + f.getValue());
fe/46.0
```

Rosen-Suzuki Functions

Model Use (instantiating the class & query (intra), publish the model and query the service (inter))

sorcer.rs.ex7.requestor.RsResponseModelRequestor

```
// Creating the Model "query" context
ModelContext modelContext = new ModelContext("Rosen-Suzuki");
VarInfoList designInfo = varsInfo(varInfo("x1", 2.0), varInfo("x2", 3.0),
    varInfo("x3", 4.0), varInfo("x4", 5.0));
modelContext.setDesignVarsInfo(designInfo);
modelContext.setResponsesInfo(null);

if (isIntra) {
model = RsResponseModelBuilder.getModel();
ModelContext outContext = (ModelContext)model.evaluate(modelContext);
logger.info(">>>>>>>>>>>>> query results: " +
    outContext.getResponsesInfo().getValueMap());
}
If (isInter){
Task responses = task("responses", sig("evaluate", ResponseModeling.class),
    modelContext);
Exertion out = exert(responses);
logger.info(">>>>>>>>>>>>> exceptions: " + out.getExceptions());
logger.info(">>>>>>>>>>>>> query results: " +
    ((ModelContext)out.getContext()).getResponsesInfo().getValueMap());
}
```

ModelContext

- The ***ModelContext*** is used to query the model, update the model or configure the model.
- If you want the values for variables or their derivatives. You use the ***ModelContext*** methods to set the appropriate path with a list of variables you desire values for. For example if you want a select set of response values from the model you would use
  - ***modelContext.setResponsesInfo(VarInfoList)***
  - where ***VarInfoList*** is a list of ***VarInfo*** variables that you desire values for. If you set ***VarInfoList = null***. The model will return the values for all ***ResponseVars*** in the model. This was demonstrated in the previous slide.

Building and Running Ex7

- There are seven different Rosen-Suzuki models in ex7. They can be found in ex7/model. They differ by the fact that some have sensitivities, some demonstrate functions of functions, some have multi-fidelity etc…
- The simplest model is ***RsResponseModelBuilder.java***. The model definition and configuration was shown earlier. Here we will discuss the publishing of the model and a requestor to call the model.
- As mentioned earlier the model can be run in two ways "***inter***" or "***intra***". "***inter***" requires the model to be "published" on the network while "***intra***" runs everything in a single JVM. The class ex7/requestor/**RsResponseModelRequestor** is a requestor that is used to query the model. This requestor is executed by running the xml file ex7/bin/***rs-response-model-prv.run.xml***.

Building and Running Ex7

In the xml file that executes the requestor (ex7/bin/***rs-response-model-req-run.xml***)
Near the bottom of the file there appears:

<target name="run.rs-response-model-requestor">
.
.
.
<arg value="inter" />

To switch between running "***inter***" versus "***intra***" one only needs to change the <arg value="inter"/> line to the desired state.

Running "***intra***"

If "intra" is selected then all that is necessary is for i-Grid to be compiled and for ex7 be compiled. If you wish you may modify the requestor to "query" the model for different information or for a different set of design variable values.

Once compiled it can be run by executing the following xml file.

run.rs-response-model-requestor.xml

Executing ex7

- The console should display the following.

Feb 2, 2011 11:38:27 AM sorcer.rs.ex7.requestor.RsResponseModelRequestor getResponses

[java] INFO: >>>>>>>>>>>>>> query results: {f=46.0, g2=71.0, g1=44.0, g3=24.0}

- To run the example as "***inter***" you must edit the ***run.rs-response-model-requestor.xml*** and set the <arg value="inter" />
- Next the model now has to be published on the network. This is done by executing the scirpt ex7/bin/***rs-response-model-prv.run.xml***. This should result in the model showing up in your sorcer browser.
- Now you can run the ***rs-response-model-requestor.xml*** (assuming iGrid is up and running as well) and should yield the same result as that found when running "inter"

[java] INFO: >>>>>>>>>>>>>> query results: {f=46.0, g2=71.0, g1=44.0, g3=24.0}

Rosen-Suzuki Functions - Ex7

Adding Sensitivities to the Vars & Model

Ex7
RsSensitivityModelBuilder.java

The ***RsSensitivityModelBuilder*** class defines and configures the Rosen-Suzuki Model for the responses and their respective sensitivities.

Model Definition – Note definition identical as RsResponseModelBuilder except for the addition of "***differentiation***" as an additional argument to ***evaluation***. "***differentiation***" has arguments "***wrt***" and "***gradient***".

```
Int designVarCount = 4
ResponseModel rm = responseModel("Rosen-Suzuki Response Model",
        designVars(vars(loop("i", 1, designVarCount),"x$i$", 0.0, -100.0, 100.0)),
        responseVar("f",
            realization(
                evaluation("FExacte", "fe",
                    differentiation(wrt(names(loop("i", 1, designVarCount), "x$i$")), gradient("FExacteg1"))))),
        responseVars(loop("i", 1, 3),"g$i$",
            realization(
                evaluation("g$i$Exacte", "g$i$e",
                    differentiation(wrt(names(loop("k", 1, designVarCount), "x$k$")), gradient("g$i$Exacteg1"))))));
```

Ex7
RsSensitivityModelBuilder.java

Model Configuration– The configuration of response variables are the same. There is now the additional configuration of the derivative evaluators and variables. Below are the evaluators for the calculation of the the derivatives of f wrt x_i

```
Evaluator dfedx1e1 = evaluator("dfedx1e1", "2.0*x1-5.0",args(model.getDesignVars("x1")));
Evaluator dfedx2e1 = evaluator("dfedx2e1", "2.0*x2-5.0",args(model.getDesignVars("x2")));
Evaluator dfedx3e1 = evaluator("dfedx3e1", "4.0*x3-21.0",args(model.getDesignVars("x3")));
Evaluator dfedx4e1 = evaluator("dfedx4e1", "2.0*x4+7.0",args(model.getDesignVars("x4")));
List<Evaluator> feg1 = list(dfedx1e1, dfedx2e1, dfedx3e1, dfedx4e1);
model.setGradientEvaluators("f", "fe", "feg1",  feg1);
```

Note: The setGradientEvaluators methods takes a List<Evaluators> as shown below

setGradientEvaluators(String variableName,String evaluatorName, String gradientName,List<Evaluator> evaluators)

An alternative is to pass a VarList in instead of the List<Evaluator>

setGradientEvaluators(String variableName,String evaluatorName, String gradientName, VarList)

Ex7 RsSensitivityModelBuilder.java

When a List<Evaluator> is passed in, internally Vars are created for each Evaluator. This is due to the fact that Vars are actually used to perform the senstivities. Below are examples of using the methods on Var to obtain the sensitivities.

```
Var f = sm.getResponseVar("f");
logger.info("\n current function value:\n" + f.getValue());
logger.info("---------------- partial derivative f wrt x1 fe|feg1: " + f.getPartialDerivative("feg1", "x1" ));
logger.info("---------------- partial derivative f wrt x2 fe|feg1: " + f.getPartialDerivative("feg1", "x2" ));
logger.info("---------------- partial derivative f wrt x3 fe|feg1: " + f.getPartialDerivative("feg1", "x3" ));
logger.info("---------------- partial derivative f wrt x4 fe|feg1: " + f.getPartialDerivative("feg1", "x4"));
logger.info("*-*-*-*-*-*-*-*-* partial gradient   f for   fe|feg1: " + f.getPartialDerivativeTable( "feg1"));
logger.info("*+*+*+*+*+*+*+*+* total gradient     f for   fe|feg1: " + f.getTotalDerivativeTable( "feg1"));

Var g1 = sm.getResponseVar("g1");
Var g2 = sm.getResponseVar("g2");
Var g3 = sm.getResponseVar("g3");
logger.info("*+*+*+*+*+*+*+*+* total gradient     g1 for   g1e|g1eg1: " + g1.getTotalDerivativeTable( "g1eg1"));
logger.info("*+*+*+*+*+*+*+*+* total gradient     g2 for   g2e|g2eg1: " + g2.getTotalDerivativeTable( "g2eg1"));
logger.info("*+*+*+*+*+*+*+*+* total gradient     g3 for   g3e|g3eg1: " + g3.getTotalDerivativeTable( "g3eg1"));
```

The next slide shows the console results for the above code.

Ex7
RsSensitivityModelBuilder.java

Results from executing rs-sensitivity-model-req-run.xml

[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: ---------- partial derivative f wrt x1 fe|feg1: -1.0
[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: ---------- partial derivative f wrt x2 fe|feg1: 1.0
[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: ---------- partial derivative f wrt x3 fe|feg1: -5.0
[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: ---------- partial derivative f wrt x4 fe|feg1: 17.0
[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: *-*-*-*-*-*-*-*-*-* partial gradient f for fe|feg1:
[java] f:fe[x1, x2, x3, x4]
[java] feg1[[D@2087c268]
[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: *+*+*+*+*+*+*+*+*+* total gradient f for fe|feg1:
[java] f:fe[x1, x2, x3, x4]
[java] feg1[-1.0, 1.0, -5.0, 17.0] [java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: *+*+*+*+*+*+*+*+*+* total gradient g1 for g1e|g1eg1:
[java] g1:g1e[x1, x2, x3, x4]
[java] g1eg1[5.0, 5.0, 9.0, 9.0]
[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: *+*+*+*+*+*+*+*+*+* total gradient g2 for g2e|g2eg1:
[java] g2:g2e[x1, x2, x3, x4]
[java] g2eg1[3.0, 12.0, 8.0, 19.0]

[java] Feb 2, 2011 6:21:32 PM sorcer.rs.ex7.model.RsSensitivityModelBuilder doSensitivityAnalysis
[java] INFO: *+*+*+*+*+*+*+*+*+* total gradient xxx g3 for g3e|g3eg1:
[java] g3:g3e[x1, x2, x3, x4]
[java] g3eg1[10.0, 5.0, 8.0, -1.0]

Optimization Example
Rosen-Suzuki Function

The following example can be found in iGrid-10

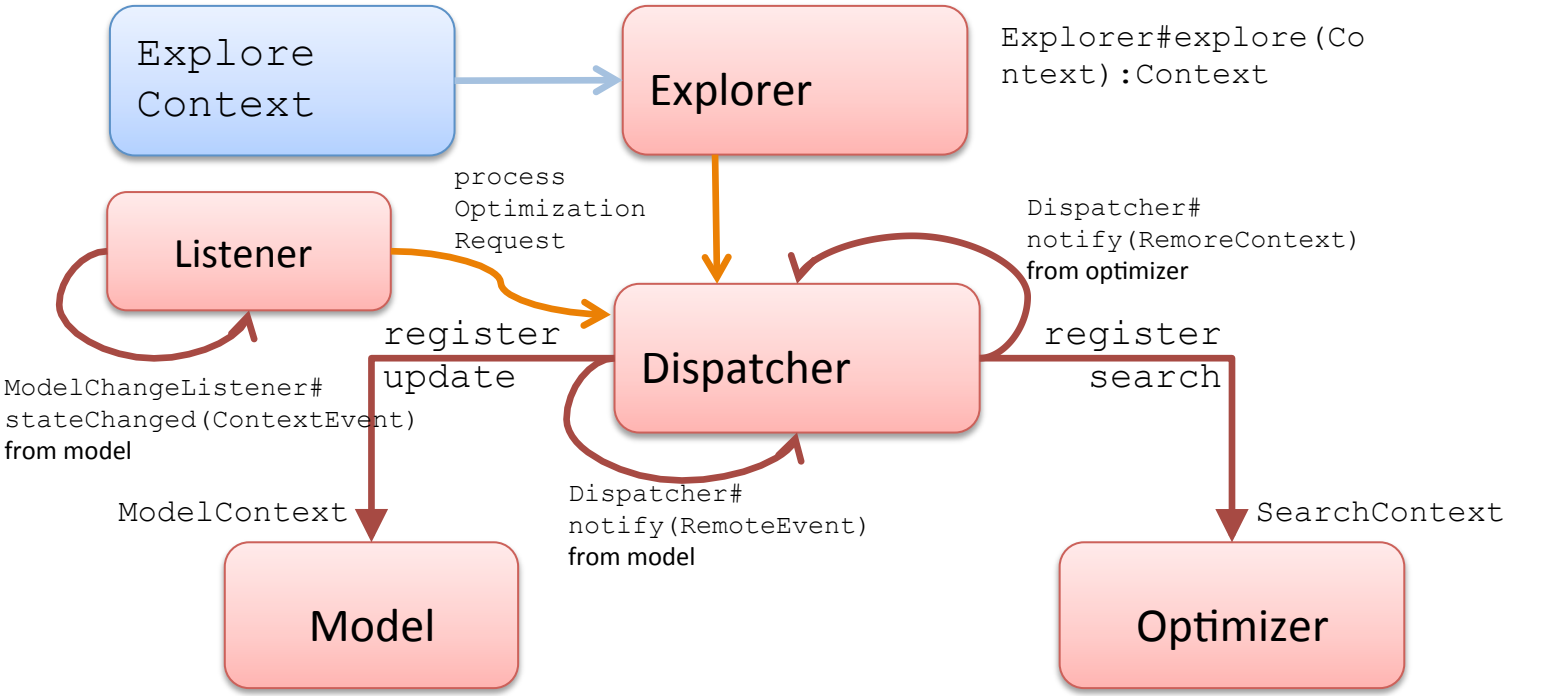
-**Exact problem optimization Example**
- **Model**
  - iGrid-10/examples/ex8/src/sorcer/model/rs/RosenSuzukiModelCreator.java
- **Dispatcher**
  -iGrid-10/examples/ex10a/src/sorcer/explorer/rs/RosenSuzukiDispatcher.java
- **Optimizer**
  -iGrid-10/examples/ex9/src/sorcer/opti/rs/RosenSuzukiOptimizer.java

-**Exact & SOA problem optimization Example**
- **Model**
  - iGrid-10/examples/ex8/src/sorcer/model/rs/RosenSuzukiMultiFidelityModelCreator.java
- **Dispatcher**
  -iGrid-10/examples/**ex10b**/src/sorcer/explorer/rs/RosenSuzukiDispatcher.java
- **Optimizer**
  -iGrid-10/examples/ex9/src/sorcer/opti/rs/RosenSuzukiOptimizer.java

Event-driven Exploration: Dispatcher, Model, Optimizer

Intra- or Inter-processor Exploration

Outer loop: Dispatcher
Inner loop: Subdispatcher
Model loop: ModelListener




```
ExplorerDispatcher#dispatch():void
ExplorerDispatcher#initializeSearchContext(SearchContext):SearchContext
ExplorerDispatcher#updateModelContext(SearchContext):ModelContext
ExplorerDispatcher#updateSearchContext(ModelContext):SearchContext
ExploreDispatcher#processModelRequest(ContextEvent):void
ExploreDispatcher#processOptimizerRequest(ContextEvent):void
ModelChangeListener#processOptimizerRequest(ModelContext):void (uses EvaluationManagement)
```

Rosen-Suzuki Function Optimization Definition

Minimize

$$f = x_1^2 - 5x_1 + x_2^2 - 5x_2 + 2x_3^2 - 21x_3 + x_4^2 + 7x_4 + 50$$

Subject to:

$$g_1 = x_1^2 + x_1 + x_2^2 - x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0 \le 0.0$$

$$g_2 = x_1^2 - x_1 + 2x_2^2 + x_3^2 + 2x_4^2 - x_4 - 10.0 \le 0.0$$

$$g_3 = 2x_1^2 + 2x_1 + x_2^2 - x_2 + x_3^2 - x_4 - 5.0 \le 0.0$$

$$-100.0 \le x_1 \le 100.0$$

$$-100.0 \le x_2 \le 100.0$$

$$-100.0 \le x_3 \le 100.0$$

$$-100.0 \le x_4 \le 100.0$$

Objective Function, Constraints and Sensitivities

Objective Sensitivities:

$$f = x_1^2 - 5x_1 + x_2^2 - 5x_2 + 2x_3^2 - 21x_3 + x_4^2 + 7x_4 + 50$$

$$\frac{\partial f}{\partial x_1} = 2x_1 - 5, \quad \frac{\partial f}{\partial x_2} = 2x_2 - 5, \quad \frac{\partial f}{\partial x_3} = 4x_3 - 21, \quad \frac{\partial f}{\partial x_4} = 2x_4 + 7$$

Constraints Sensitivities:

$$g_1 = x_1^2 + x_1 + x_2^2 - x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0 \le 0.0$$

$$\frac{\partial g_1}{\partial x_1} = 2x_1 + 1.0, \quad \frac{\partial g_1}{\partial x_2} = 2x_2 - 1.0, \quad \frac{\partial g_1}{\partial x_3} = 2x_3 + 1.0, \quad \frac{\partial g_1}{\partial x_4} = 2x_4 - 1.0$$

$$g_2 = x_1^2 - x_1 + 2x_2^2 + x_3^2 + 2x_4^2 - x_4 - 10.0 \le 0.0$$

$$\frac{\partial g_2}{\partial x_1} = 2x_1 - 1.0, \quad \frac{\partial g_2}{\partial x_2} = 4x_2, \quad \frac{\partial g_2}{\partial x_3} = 2x_3, \quad \frac{\partial g_2}{\partial x_4} = 4x_4 - 1.0$$

$$g_3 = 2x_1^2 + 2x_1 + x_2^2 - x_2 + x_3^2 - x_4 - 5.0 \le 0.0$$

$$\frac{\partial g_3}{\partial x_1} = 4x_1 + 2.0, \quad \frac{\partial g_3}{\partial x_2} = 2x_2 - 1.0, \quad \frac{\partial g_3}{\partial x_3} = 2x_3, \quad \frac{\partial g_3}{\partial x_4} = -1.0$$

Rosen-Suzuki Function Optimization

Optimization Model Definition

```
OptimizationModel om = optimizationModel("DesignExploration Model",
                designVars(vars(loop(4), "x", 0.0,-100., 100.)),
                responseVars("f"),
                responseVars("g",3),
                objectiveVars(var("fo", "f", Target.min )),
                constraintVars(var("g1c", "g1", Relation.lt, 0.0),
                               var("g2c", "g2", Relation.lt, 0.0),
                               var("g3c","g3", Relation.lt, 0.0)))
);
```

Exact Function Optimization

Rosen-Suzuki Function Optimization

Optimization Model Configuration

Sensitivity Evaluators for the Objective

$$\frac{\partial f}{\partial x_1} = 2x_1 - 5, \quad \frac{\partial f}{\partial x_2} = 2x_2 - 5, \quad \frac{\partial f}{\partial x_3} = 4x_3 - 21, \quad \frac{\partial f}{\partial x_4} = 2x_4 + 7$$

Evaluator dfedx1e1 = *evaluator("dfedx1e1", "2.0*x1-5.0",args(om.getDesignVars("x1")));*

Evaluator dfedx2e1 = *evaluator("dfedx2e1", "2.0*x2-5.0",args(om.getDesignVars("x2")));*

Evaluator dfedx3e1 = *evaluator("dfedx3e1",*
*"4.0*x3-21.0",args(om.getDesignVars("x3")));*

Evaluator dfedx4e1 = *evaluator("dfedx4e1", "2.0*x4+7.0",args(om.getDesignVars("x4")));*

List<Evaluator> feg1 = *list(dfedx1e1, dfedx2e1, dfedx3e1, dfedx4e1);*
om.setGradientEvaluators("f", "fe", "feg1", feg1);

Exact Function Optimization

Rosen-Suzuki Function Optimization

Optimization Model Configuration

Sensitivity Evaluators for the Constraints

$$\frac{\partial g_1}{\partial x_1} = 2x_1 + 1.0, \quad \frac{\partial g_1}{\partial x_2} = 2x_2 - 1.0, \quad \frac{\partial g_1}{\partial x_3} = 2x_3 + 1.0, \quad \frac{\partial g_1}{\partial x_4} = 2x_4 - 1.0$$

```
Evaluator dg1edx1e1 = evaluator("dg1edx1e1", "2.0*x1+1",args(om.getDesignVars("x1")));

Evaluator dg1edx2e1 = evaluator("dg1edx2e1", "2.0*x2-1.0",args(om.getDesignVars("x2")));

Evaluator dg1edx3e1 = evaluator("dg1edx3e1", "2.0*x3+1.0",args(om.getDesignVars("x3")));

Evaluator dg1edx4e1 = evaluator("dg1edx4e1", "2.0*x4-1.0",args(om.getDesignVars("x4")));

List<Evaluator> g1eg1 = list(dg1edx1e1, dg1edx2e1, dg1edx3e1, dg1edx4e1);
om.setGradientEvaluators("g1", "g1e", "g1eg1", g1eg1);
```

Exact Function Optimization

Rosen-Suzuki Function Optimization

Optimization Model Configuration

Sensitivity Evaluators for the Constraints

$$\frac{\partial g_2}{\partial x_1} = 2x_1 - 1.0, \quad \frac{\partial g_2}{\partial x_2} = 4x_2, \quad \frac{\partial g_2}{\partial x_3} = 2x_3, \quad \frac{\partial g_2}{\partial x_4} = 4x_4 - 1.0$$

```
Evaluator dg2edx1e1 = evaluator("dg2edx1e1", "2.0*x1-1",args(om.getDesignVars("x1")));

Evaluator dg2edx2e1 = evaluator("dg2edx2e1", "4.0*x2",args(om.getDesignVars("x2")));

Evaluator dg2edx3e1 = evaluator("dg2edx3e1", "2.0*x3",args(om.getDesignVars("x3")));

Evaluator dg2edx4e1 = evaluator("dg2edx4e1", "4.0*x4-1.0",args(om.getDesignVars("x4")));

List<Evaluator> g2eg1 = list(dg2edx1e1, dg2edx2e1, dg2edx3e1, dg2edx4e1);
om.setGradientEvaluators("g2", "g2e", "g2eg1", g2eg1);
```

Exact Function Optimization

Rosen-Suzuki Function Optimization

Optimization Model Configuration

Sensitivity Evaluators for the Constraints

$$\frac{\partial g_3}{\partial x_1} = 4x_1 + 2.0, \quad \frac{\partial g_3}{\partial x_2} = 2x_2 - 1.0, \quad \frac{\partial g_3}{\partial x_3} = 2x_3, \quad \frac{\partial g_3}{\partial x_4} = -1.0$$

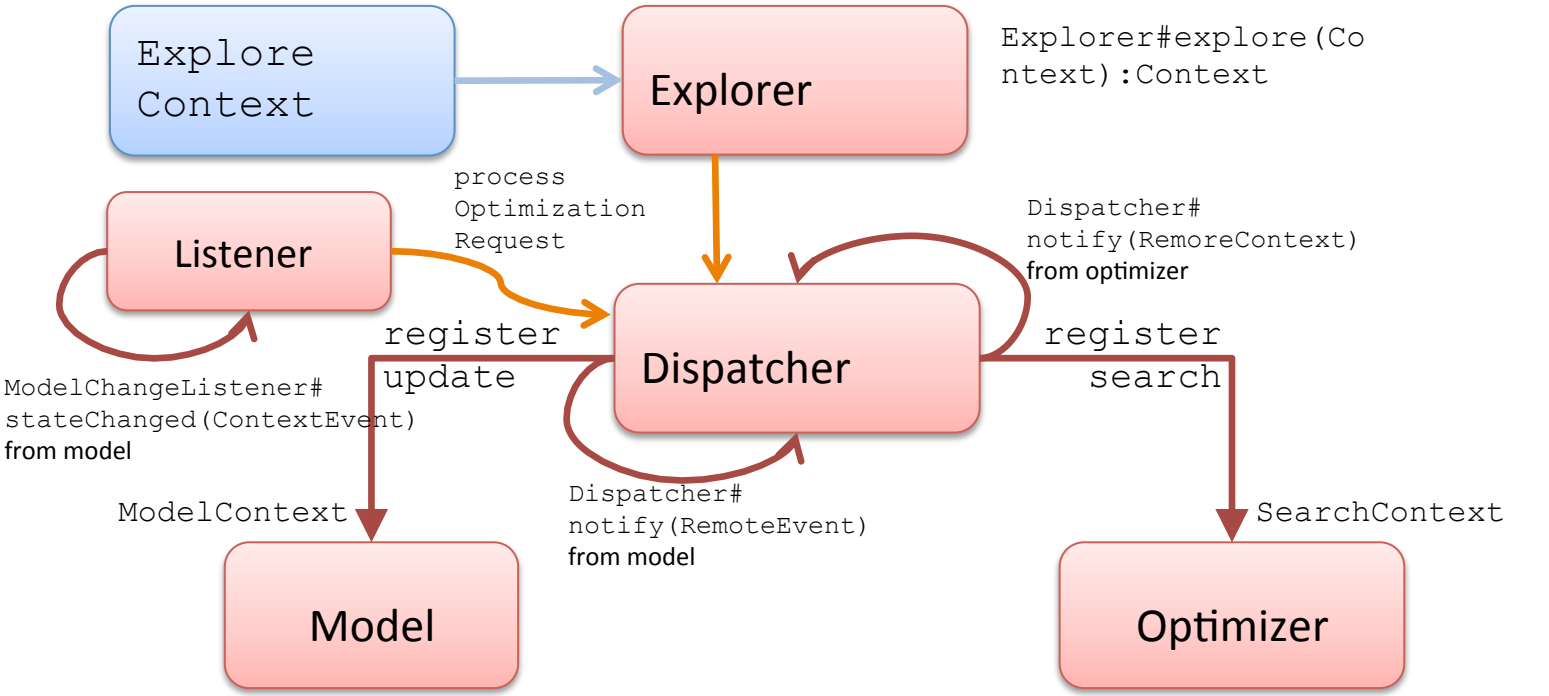
```
Evaluator dg3edx1e1 = evaluator("dg3edx1e1", "4.0*x1+2.0",args(om.getDesignVars("x1")));

Evaluator dg3edx2e1 = evaluator("dg3edx2e1", "2.0*x2-1.0",args(om.getDesignVars("x2")));

Evaluator dg3edx3e1 = evaluator("dg3edx3e1", "2.0*x3",args(om.getDesignVars("x3")));

Evaluator dg3edx4e1 = evaluator("dg3edx4e1", "-1.0");

List<Evaluator> g3eg1 = list(dg3edx1e1, dg3edx2e1, dg3edx3e1, dg3edx4e1);
om.setGradientEvaluators("g3", "g3e", "g3eg1", g3eg1);
```

Exact Function Optimization

Event-driven Exploration: Dispatcher, Model, Optimizer

Intra- or Inter-processor Exploration

Outer loop: Dispatcher
Inner loop: Subdispatcher
Model loop: ModelListener




```
ExplorerDispatcher#dispatch():void
ExplorerDispatcher#initializeSearchContext(SearchContext):SearchContext
ExplorerDispatcher#updateModelContext(SearchContext):ModelContext
ExplorerDispatcher#updateSearchContext(ModelContext):SearchContext
ExploreDispatcher#processModelRequest(ContextEvent):void
ExploreDispatcher#processOptimizerRequest(ContextEvent):void
ModelChangeListener#processOptimizerRequest(ModelContext):void (uses EvaluationManagement)
```

Explorer & ExplorerDispatcher

- Explorer instantiates an ExplorerDispatcher and a ModelManager based on what is in the ExplorerContext
- The dispatcher coordinates/processes events between the Model and the Optimizer. The processing of the events is done by two methods in the ExplorerDispatcher
  - **ExploreDispatcher#processModelRequest(ContextEvent):void**
    - The contextEvent contains a ModelContext. It calls the following methods
    - ExplorerDispatcher#updateSearchContext(ModelContext):SearchContext
    - optimizer.search(searchContext)
  - **ExploreDispatcher#processOptimizerRequest(ContextEvent):void**
    - The contextEvent contains a SearchContext. It calls the following methods
    - ExplorerDispatcher#updateModelContext(SearchContext):ModelContext
    - model.notifyEvaluation(modelContext);

Explorer Context (Data)

Explorer Context

VariableInfo

- Design Variable Info
- Objectives Info
- Objectives Gradient Info
- Constraints Info
- Constraints Gradient Info

Optimization

- Optimization State
- Optimization Strategy
- Optimization Signature

Dispatcher

- Dispatcher Signature
- Dispatcher Strategy

Model

- Model Signature
- ModelManager Signature

Rosen-Suzuki Function Optimization

Exact Function Optimization

Explorer Context

```
ExploreContext exploreContext = new ExploreContext("Rosen-Suzuki");
VarInfoList designInfo = varsInfo(varInfo("x1", 1.0), varInfo("x2", 1.0), varInfo("x3", 1.0), varInfo("x4", 1.0));
exploreContext.setDesignVarsInfo(designInfo);

exploreContext.setObjectivesInfo(null);
exploreContext.setObjectivesGradientInfo(null);

exploreContext.setConstraintsInfo(null);
exploreContext.setConstraintsGradientInfo(null);

ConminStrategy cmnStrategy=new ConminStrategy(new File(System.getProperty("conmin.strategy.file")));
exploreContext.setOptimizerStrategy(cmnStrategy);
//exploreContext.setOptimizerState(cmnState); // state initialized in Dispatcher after first Model update


// specify the Model
exploreContext.setModelSignature(signature("createModel", RosenSuzukiModelCreator.class, Type.INTRAPROCESS));

//specify the Optimizer
exploreContext.setOptimizerSignature(signature(null, ConminOptimizerJNA.class, Type.INTRAPROCESS));

// specify the explorer dispatched
exploreContext.setDispatcherSignature(signature(null, RosenSuzukiDispatcher.class, Type.INTRAPROCESS));
```

Rosen-Suzuki Function Optimization

Explorer Execution

Exact Function Optimization

```
outContext = (ExploreContext)explorer.explore(exploreContext);

********* CONMIN STATE **************

CONMIN Iteration # =  29

Objective Fuction fo = 6.00260795945281

Design Variable Values
   x1 = 2.5802964087086235E-4 x2 = 0.9995594642481355 x3 = 2.000313835134211 x4 = -0.9986692050113675

Constraint Values
   g1c = -0.002603585246998996 g2c = -1.0074147118087602 g3c = 4.948009193483927E-7
T
Termination Criterion
    ABS(OBJ(I)-OBJ(I-1))   LESS THAN DABFUN = 5.0E-5 FOR 3 ITERATIONS

Evaluation Statistics
    Number of Objective Evaluations = 88
    Number of Constraint Evaluations = 88
    Number of Objective Gradient Evaluations = 29
     Number of Constraint Gradient Evaluations = 29
```

Analytic Optimum Results
fo = 6.0
X1=0.0, x2=1.0, x3=2.0, x4=-1.0
g1c=0.0, g2c=-1.0, g3c=0.0

Rosen-Suzuki Functions Optimization Example

- How to run the Optimization Example – ex10a
- Building all the necessary classes
  - in ex10a execute the rs-all-build.xml. This will compile the needed classes and build the needed jars – the conmin provider (ConminOptimizerJNA, ..) using the conmin-prv-build.xml , example8, using the rs-model-build.xml which compiles the RosenSuzukiModelBuilder class (the model), and finally the exp rs-explorer-build.xml in the ex10a folder. Rs-explorer-build.xml compiles the RosenSuzukiDispatcher and the RosenSuzukiExplorerRequestor.

- Executing ex10
  - First there are two ways to execute the requestor, “intra” or “inter”. In the 10a bin folder there is a file named – rs-explorer-req-run.xml. This is the requestor xml file. If you open this file and go to the bottom you should see a

```
<target name="run.requestor">
.
.
.
<arg value="inter" />
```

The last <arg value> will be either “inter” or “intra” .

  - If it is “intra” then no services need to be published. I-Grid does not need to be running, and Webster does not need to be running. “Intra” causes all classes to be instantiated “local”, in the same virtual machine and run there.

- You can now run the example by executing rs-explorer-req-run.xml. This should produce the following result in the console

```
Context name: ********* CONMIN STATE **************

   CONMIN Iteration # =  29

   Objective Function fo = 6.002607805900986

   Design Variable Values
x1 = 2.5802964087086235E-4 x2 = 0.9995594642481355 x3 = 2.000313835134211 x4 = -0.9986692050113675
   Design Variable Values iter-1
    x1 = 3.198365004081394E-4 x2 = 0.9998803365384545 x3 = 2.000177879135422 x4 = -0.9987682233679354

Constraint Values
    g1c = -0.002603585246998996 g2c = -1.0074147118087602 g3c = 4.948009193483927E-7

Note: Objective Function and Constraint Values may be for iter-1, not the current Design Variale Values
Termination Criterion
                ABS(OBJ(I)-OBJ(I-1))   LESS THAN DABFUN = 5.0E-5 FOR 3 ITERATIONS

Evaluation Statistics
    Number of Objective Evaluations = 88
    Number of Constraint Evaluations = 88
    Number of Objective Gradient Evaluations = 29
    Number of Constraint Gradient Evaluations = 29

*************************************************************
```

- You can also go into the 10a/logs folder and open the log file and see the same result.

- The second way to run 10a is “inter”. This is done by changing <arg value=”inter”> in rs-explorer-req-run.xml .
- This now runs the problem in a distributed mode. That is it uses services on the network. It uses 3 services, Rosen-Suzuki Model, Rosen-Suzuki Optimizer (this is conmin), and Rosen-Suzuki Explorer.
- For this case you must start Webster (webster-run-xml) and iGrid-min-boot.xml. Next you need to start the 3 above mentioned services. They can be started with the single xml file in the ex10a/bin folder called all-rs-opti-prv-boot.xml. This xml file publishes the services as beans. If this executes properly you should see the following three services in the sorcer browser. Rosen-Suzuki Model, Rosen-Suzuki Optimizer, and Rosen-Suzuki Explorer. You can also publish each of these individually if you like. With the following commands. For the model – ex8/bin/rs-model-prv.run.xml, for the optimization provider (conmin), ex9/bin/crs-opti-prv.run.xml, and finally for the explorer, ex10a/bin/rs-explorer-prv.run.xml.
- You are now ready to run the rs-explorer-req-run.xml in “inter” mode. If this runs properly you should get the same result as you got when running “intra”

- Multi-Fidelity for the Responses and Sensitivities

Objective Function Multi-Fidelity

Two Evaluators for F. Each evaluator represents a different fidelity for the calculation of F (multi - fidelity)

$$F_{e1} = x_1^2 - 5x_1 + x_2^2 - 5x_2 + 2x_3^2 - 21x_3 + x_4^2 + 7x_4 + 5$$

$$F_{e2} = f_1 + f_3 + f_4 - 21x_3 + xl_1 + p_1$$

Where

$$xl_1 = 7x_4$$

$$f_1 = x_1^2 + x_2^2$$

$$f_2 = x_1 + x_2$$

$$f_3 = -5f_2$$

$$f_4 = 2x_3^2 + x_4^2$$

$$p_1 = 50.$$

Constraint Functions Multi-Fidelity

Two Evaluators for G1

$$G_{1e1} = x_1^2 + x_1 + x_2^2 - x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0 \le 0.0$$

$$G_{1e2} = f_1 + f_2 - 2x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0 \le 0.0$$

Single Evaluators for G2 & G3

$$G_{2e1} = x_1^2 - x_1 + 2x_2^2 + x_3^2 + 2x_4^2 - x_4 - 10.0 \le 0.0$$

$$G_{3e1} = 2x_1^2 + 2x_1 + x_2^2 - x_2 + x_3^2 - x_4 - 5.0 \le 0.0$$

Objective Sensitivities with Multi-Fidelity

$$F_{e1} = x_1^2 - 5x_1 + x_2^2 - 5x_2 + 2x_3^2 - 21x_3 + x_4^2 + 7x_4 + 50$$

Analytic

$$\frac{\partial F_{e1g1}}{\partial x_1} = 2x_1 - 5, \quad \frac{\partial F_{e1g1}}{\partial x_2} = 2x_2 - 5, \quad \frac{\partial F_{e1g1}}{\partial x_3} = 4x_3 - 21, \quad \frac{\partial F_{e1g1}}{\partial x_4} = 2x_4 + 7$$

Finite Difference

$$\frac{\partial F_{e1g2}}{\partial x_1} = F_{e1}(x_1 + \Delta x_1) - F_{e1}(x_1)/\Delta x_1 : \text{ Note : SORCER has a FD Evaluator}$$

$$\frac{\partial F_{e1g2}}{\partial x_2} = F_{e1}(x_2 + \Delta x_2) - F_{e1}(x_2)/\Delta x_2$$

$$\frac{\partial F_{e1g2}}{\partial x_3} = F_{e1}(x_3 + \Delta x_3) - F_{e1}(x_3)/\Delta x_3$$

$$\frac{\partial F_{e1g2}}{\partial x_4} = F_{e1}(x_4 + \Delta x_4) - F_{e1}(x_4)/\Delta x_4$$

Objective Sensitivities with Multi-Fidelity

Multi-Fidelity Sensitivities for the Multi-Fidelity Objective

$$F_{e2} = f_1 + f_3 + f_4 - 21x_3 + xl_1 + p_1$$

$$\frac{\partial F_{e2g1}}{\partial f_1} = 1.0, \quad \frac{\partial F_{e2g1}}{\partial f_3} = 1.0, \quad \frac{\partial F_{e2g1}}{\partial f_4} = 1.0, \quad \frac{\partial F_{e2g1}}{\partial x_3} = -21.0, \quad \frac{\partial F_{e2g1}}{\partial xl_1} = 1.0, \quad \frac{\partial F_{e2g1}}{\partial p} = 0.0$$

$$\frac{\partial F_{e2g2}}{\partial f_i} = (F_{e2}(f_i + \Delta f_i) - F_{e2}(f_i))/\Delta f_i, \quad \frac{\partial F_{e2g2}}{\partial x_3} = (F_{e2}(x_3 + \Delta x_3) - F_{e2}(x_3))/\Delta x_3,$$

$$\frac{\partial F_{e2g2}}{\partial xl_1} = (F_{e2}(xl_1 + \Delta xl_1) - F_{e2}(xl_1))/\Delta xl_1$$

$$xl_1 = 7x_4, \quad \frac{\partial xl_1}{\partial x_4} = 7.0$$

$$f_1 = x_1^2 + x_2^2, \frac{\partial f_1}{\partial x_1} = 2x_1, \quad \frac{\partial f_1}{\partial x_2} = 2x_2$$

$$f_2 = x_1 + x_2, \quad \frac{\partial f_2}{\partial x_1} = 1.0 \ , \quad \frac{\partial f_2}{\partial x_2} = 1.0$$

$$f_3 = -5f_2, \ \frac{\partial f_3}{\partial f_2} = -5.0$$

$$f_4 = 2x_3^2 + x_4^2, \quad \frac{\partial f_{4g1}}{\partial x_3} = 4x_3, \quad \frac{\partial f_{4g1}}{\partial x_4} = 2x_4, \quad \frac{\partial f_{4g2}}{\partial x_i} = (f_4(x_i + \Delta x_i) - f_4(x_i))/\Delta x_i$$

$$p_1 = 50.$$

Multi-Fidelity Constraints & Multi-Fidelity Sensitivities

$$G_{e11} = x_1^2 + x_1 + x_2^2 - x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0 \le 0.0$$

Analytic

$$\frac{\partial G_{1e1g1}}{\partial x_1} = 2x_1 + 1.0, \quad \frac{\partial G_{1e1g1}}{\partial x_2} = 2x_2 - 1.0, \quad \frac{\partial G_{1e1g1}}{\partial x_3} = 2x_3 + 1.0, \quad \frac{\partial G_{1e1g1}}{\partial x_4} = 2x_4 - 1.0$$

Finite Difference

$$\frac{\partial G_{1e1g2}}{\partial x_1} = (G_{1e1}(x_i + \Delta x_i) - G_{1e1}(x_i))/\Delta x_i$$

$$G_{1e2} = f_1 + f_2 - 2x_2 + x_3^2 + x_3 + x_4^2 - x_4 - 8.0 \le 0.0$$

Analytic

$$\frac{\partial G_{1e2g1}}{\partial f_1} = 1.0, \quad \frac{\partial G_{1e2g1}}{\partial f_2} = 1.0, \quad \frac{\partial G_{1e2g1}}{\partial x_2} = -2.0, \quad \frac{\partial G_{1e2g1}}{\partial x_3} = 2x_3 + 1.0, \quad \frac{\partial G_{1e2g1}}{\partial x_4} = 2x_2 - 1.0$$

Finite Difference

$$\frac{\partial G_{1e2g2}}{\partial f_i} = (G_{1e2g2}(f_i + \Delta f_i) - G_{1e2g2}(f_i))/\Delta f_i, \quad \frac{\partial G_{1e2g2}}{x_i} = (G_{1e2g2}(x_i + \Delta x_i) - G_{1e2g2}(x_i))/\Delta x_i$$

Constraint Sensitivities

$$G_2 = x_1^2 - x_1 + 2x_2^2 + x_3^2 + 2x_4^2 - x_4 - 10.0 \le 0.0$$

$$\frac{\partial G_2}{\partial x_1} = 2x_1 - 1.0, \quad \frac{\partial G_2}{\partial x_2} = 4x_2, \quad \frac{\partial G_2}{\partial x_3} = 2x_3, \quad \frac{\partial G_2}{\partial x_4} = 4x_4 - 1.0$$

$$G_3 = 2x_1^2 + 2x_1 + x_2^2 - x_2 + x_3^2 - x_4 - 5.0 \le 0.0$$

$$\frac{\partial G_3}{\partial x_1} = 4x_1 + 2.0, \quad \frac{\partial G_3}{\partial x_2} = 2x_2 - 1.0, \quad \frac{\partial G_3}{\partial x_3} = 2x_3, \quad \frac{\partial G_3}{\partial x_4} = -1.0$$

Model Definition with Multi-Fidelity Responses and Sensitivities


```
ResponseModel sm = responseModel("Sensitivity Analysis",
designVars("x", designVarCount),
linkedVars("xl1"),
parameterVars(var("p1", 50.0)),
responseVars("f",
    realization(
       evaluation("fe1"), evaluation("fe2"),
          differentiation("fe1", wrt(names(loop(1, 4), "x")), gradient("fe1g1"), gradient("fe1g2") ),
          differentiation("fe2", wrt("f1", "f3", "f4", "x3", "xl1"), gradient("fe2g1"),
gradient("fe2g2"))),
responseVars("f",4),
     realization("f1", differentiation(wrt("x1", "x2"))),
     realization("f2", differentiation(wrt("x1", "x2"))),
     realization("f3", differentiation(wrt("f2"))),
     realization("f4", evaluation("f4e"),
         differentiation("f4e", wrt("x3", "x4"), gradient("f4eg1"), gradient("f4eg2"))),
responseVars("g", 3),
      realization("g1", evaluation("g1e1"), evaluation("g1e2"),
          differentiation(wrt(names(loop(4), "x")), gradient("g1e1g1"), gradient("g1e1g2")),
         differentiation(wrt("f1", "f2", "x2", "x3", "x4"), gradient("g1e2g1"), gradient("g1e2g2"))),
realization("g2", differentiation(wrt(names(loop(4), "x")))),
realization("g3", differentiation(wrt("x1", "x2", "x3", "x4"))));
```

NOTE: With Multi-Fidelity the realizations MUST be Defined in the Model

Configure the Response Variables


```
var(model, "xl1", evaluator("xl1e", "7*x4"), args("x4"));
// first fidelity for f:fe1
var(model, "f", "fe1", evaluator("fe1", "x1^2-5.0*x1+x2^2-5.0*x2+2.0*x3^2-21.0*x3+x4^2+7.0*x4+50.0"),
        args("x1", "x2", "x3", "x4"));
First fidelity for g1:g1e1
var(model, "g1", evaluator("g1e1", "x1^2+x1+x2^2-x2+x3^2+x3+x4^2-x4-8.0"),
        args("x1", "x2", "x3", "x4"));
var(model, "g2", evaluator("g2e", "x1^2-x1+2.0*x2^2+x3^2+2.0*x4^2-x4-10.0"),
        args("x1", "x2", "x3", "x4"));
var(model, "g3", evaluator("g3e", "2.0*x1^2+2.0*x1+x2^2-x2+x3^2-x4-5.0"),
        args("x1", "x2", "x3", "x4"));
// second fidelity for f:fe2
var(model, "f", "fe2", evaluator("fe2", "f1+f3+f4-21.0*x3+xl1+p1+.01"),args("x3", "f1","f3","f4", "xl1", "p1"));
var(model, "f1", evaluator("f1e", "x1^2+x2^2"), args("x1","x2"));
var(model, "f2", evaluator("f2e","x1+x2"), args("x1","x2"));
var(model, "f3", evaluator("f3e", "-5.0*f2"), args("f2"));
var(model, "f4", evaluator("f4e", "2.0*x3^2+x4^2"), args("x3","x4"));
// second fidelity for g1:g1e2
var(model, "g1", evaluator("g1e2", "f1+f2-2.0*x2+x3^2+x3+x4^2-x4-8.0"), args("f1", "f2”,"x2","x3","x4"));
```

Configure the sensitivities for the Response Model


```
// f:fe1
// design vars: x1, x2, x3, x4
// response vars: f
// response f:fe1="x1^2-5.0*x1+x2^2-5.0*x2+2.0*x3^2-21.0*x3+x4^2+7.0*x4+50.0"
//dfdxi:dfe1dxie1
// partial derivative dfe1dx1e1 = "2.0*x1-5.0"
// partial derivative dfe1dx2e1 = "2.0*x2-5.0"
// partial derivative dfe1dx3e1 = "4.0*x3-21.0"
// partial derivative dfe1dx4e1 = "2.0*x4+7.0"
Evaluator dfe1dx1e1 = expression("dfe1dx1e1", "2.0*x1-5.0",args(sm.getDesignVar("x1")));
Evaluator dfe1dx2e1 = expression("dfe1dx2e1", "2.0*x2-5.0",args(sm.getDesignVar("x2")));
Evaluator dfe1dx3e1 = expression("dfe1dx3e1", "4.0*x3-21.0",args(sm.getDesignVar("x3")));
Evaluator dfe1dx4e1 = expression("dfe1dx4e1", "2.0*x4+7.0",args(sm.getDesignVar("x4")));
List<Evaluator> fe1g1 = list(dfe1dx1e1, dfe1dx2e1, dfe1dx3e1, dfe1dx4e1);
sm.setGradientEvaluators("f", "fe1", "fe1g1",  fe1g1);



//dfdxi:dfe1dxie2
// partial derivative dfe1dx1e2 = (fe1(x1+deltax1) -fe1(x1))/deltax1
// partial derivative dfe1dx2e2 = (fe1(x2+deltax2) -fe1(x1))/deltax2
// partial derivative dfe1dx3e2 = (fe1(x3+deltax3) -fe1(x1))/deltax3
// partial derivative dfe1dx4e2 = (fe1(x4+deltax4) -fe1(x1))/deltax4

Evaluator dfe1dx1e2 = new FiniteDifferenceEvaluator("dfe1dx1e2", sm.getResponseVar("f").getEvaluator("fe1"),0.1, "x1");
Evaluator dfe1dx2e2 = new FiniteDifferenceEvaluator("dfe1dx2e2", sm.getResponseVar("f").getEvaluator("fe1"),0.1, "x2");
Evaluator dfe1dx3e2 = new FiniteDifferenceEvaluator("dfe1dx3e2", sm.getResponseVar("f").getEvaluator("fe1"),0.1, "x3");
Evaluator dfe1dx4e2 = new FiniteDifferenceEvaluator("dfe1dx4e2", sm.getResponseVar("f").getEvaluator("fe1"),0.1, "x4");
List<Evaluator> fe1g2 = list(dfe1dx1e2, dfe1dx2e2, dfe1dx3e2, dfe1dx4e2);
sm.setGradientEvaluators("f", "fe1", "fe1g2",  fe1g2);

// link variable
Evaluator dfe1dxl1e = expression("dfe1dxl1e", "7",args(sm.getDesignVar("x4")));
sm.setGradientEvaluators("xl1",  "xl1e", "xl1eg", list(dfe1dxl1e));
```

Fe2 with with Multi-Fidelity Sensitivities


```
// f:fe2
// design vars: x1, x2, x3, x4
// linked vars: xl1="7.0*x4"
// parameters: p1=50.
// response vars: f, f1, f2, f3, f4,
// response f:fe2="f1+f3+f4-21.0*x3+xl1+p1"
// dfdxi:dfe2dxie1
// partial derivative dfe2df1e1="1.0"
// partial derivative dfe2df3e1="1.0"
// partial derivative dfe2df4e1="1.0"
// partial derivative dfe2dx3e1="-21.0"
// partial derivative dfe2dxl1e1="1.0"
Evaluator dfe2df1e1 = expression("dfe2df1e1", "1.0");
Evaluator dfe2df3e1 = expression("dfe2df3e1", "1.0");
Evaluator dfe2df4e1 = expression("dfe2df4e1", "1.0");
Evaluator dfe2dx3e1 = expression("dfe2dx3e1", "-21.0");
Evaluator dfe2dxl1e1 = expression("dfe2dxl1e1", "1.0");
Evaluator dfe2dp1e1 = expression("dfe2dp1e1", "0.0");
List<Evaluator> fe2g1 = list(dfe2df1e1, dfe2df3e1, dfe2df4e1, dfe2dx3e1, dfe2dxl1e1, dfe2dp1e1);
sm.setGradientEvaluators("f", "fe2", "fe2g1",  fe2g1);
// dfdxi:dfe2dxie2
// partial derivative dfe2df1e2 = (fe2(f1+deltaf1) -fe2(f1))/deltaf1
// partial derivative dfe2df3e2 = (fe2(f3+deltaf3) -fe2(f3))/deltaf3
// partial derivative dfe2df4e2 = (fe2(f4+deltaf4) -fe2(f4))/deltaf4
// partial derivative dfe2dx3e2 = (fe2(x3+deltax3) -fe2(x3))/deltax3
// partial derivative dfe2dxl1e2 = (fe2(xl1+deltaxl1) -fe2(x1l))/deltaxl1
Evaluator dfe2df1e2 = new FiniteDifferenceEvaluator("dfe2df1e2", sm.getResponseVar("f").getEvaluator("fe2"), 0.1, "f1");
Evaluator dfe2df3e2 = new FiniteDifferenceEvaluator("dfe2df3e2", sm.getResponseVar("f").getEvaluator("fe2"), 0.1, "f3");
Evaluator dfe2df4e2 = new FiniteDifferenceEvaluator("dfe2df4e2", sm.getResponseVar("f").getEvaluator("fe2"), 0.1, "f4");
Evaluator dfe2dx3e2 = new FiniteDifferenceEvaluator("dfe2dx3e2", sm.getResponseVar("f").getEvaluator("fe2"), 0.1, "x3");
Evaluator dfe2dxl1e2 = new FiniteDifferenceEvaluator("dfe2dxl1e2", sm.getResponseVar("f").getEvaluator("fe2"), 0.1, "xl1");
Evaluator dfe2dp1e2 = expression("dfe2dp1e2", "0.0");
List<Evaluator> fe2g2 = list(dfe2df1e2, dfe2df3e2, dfe2df4e2, dfe2dx3e2, dfe2dxl1e2,dfe2dp1e2 );
sm.setGradientEvaluators("f", "fe2", "fe2g2",  fe2g2);
```

Fe2 with with Multi-Fidelity Sensitivities


```
// response f1="x1^2+x2^2"
// partial derivative df1/dx1 ="2.0*x1"
// partial derivative df1/dx2="2.0*x2"
Evaluator df1edx1e1 = expression("df1edx1e1", "2.0*x1",args(sm.getDesignVar("x1")));
Evaluator df1edx2e1 = expression("df1edx2e1", "2.0*x2",args(sm.getDesignVar("x2")));
List<Evaluator> f1eg1 = list(df1edx1e1, df1edx2e1);
sm.setGradientEvaluators("f1", "f1e", "f1eg1",  f1eg1);
// response f2="x1+x2"
// partial derivative df2/dx1="1.0"
// partial derivative df2/dx2="1.0"
Evaluator df2edx1e1 = expression("df2edx2e1","1.0");
Evaluator df2edx2e1 = expression("df2edx2e1", "1.0");
List<Evaluator> f2eg1 = list(df2edx1e1, df2edx2e1);
sm.setGradientEvaluators("f2", "f2e", "f2eg1",  f2eg1);
// response f3="-5.0*f2"
// partial derivative df3/df2="-5.0"
Evaluator df3edf2e1 = expression("df3edf2e1","-5.0");
List<Evaluator> f3eg1 = list(df3edf2e1);
sm.setGradientEvaluators("f3", "f3e", "f3eg1",  f3eg1);
// response f4:f4e=2.0*x3^2+x4^2
// df4dxie1
// partial derivative df4/dx3:df4edx1e1="4.0*x3"
// partial derivative df4/dx4:df4edx4e1="2.0*x4"
Evaluator df4edx3e1 = expression("df4edx3e1","4.0*x3",args(sm.getDesignVar("x3")));
Evaluator df4edx4e1 = expression("df4edx4e1", "2.0*x4",args(sm.getDesignVar("x4")));
List<Evaluator> f4eg1 = list(df4edx3e1, df4edx4e1);
sm.setGradientEvaluators("f4", "f4e", "f4eg1",  f4eg1);
// df4dxie2
// partial derivative df4/dx3:df4edx3e2=(f4e(x3+deltax3) -f4e(x3))/deltax3
// partial derivative df4/dx4:df4edx4e2=(f4e(x4+deltax4) -f4e(x4))/deltax4
   Evaluator df4edx3e2 = new FiniteDifferenceEvaluator("df4edx3e2", sm.getResponseVar("f4").getEvaluator("f4e"), 0.1, "x2");
   Evaluator df4edx4e2 = new FiniteDifferenceEvaluator("df4edx4e2", sm.getResponseVar("f4").getEvaluator("f4e"), 0.1, "x4");
List<Evaluator> f4eg2 = list(df4edx3e2, df4edx4e2);
sm.setGradientEvaluators("f4", "f4e", "f4eg2",  f4eg2);
```

g1e1 with with Multi-Fidelity Sensitivities


```
//g1: g1e1
// response g1:g1e1="x1^2+x1+x2^2-x2+x3^2+x3+x4^2-x4-8.0"
// dg1e1dxie1
// derivativeResponse dg1dx1:dg1e1dx1 = "2.0*x1+1.0"
// derivativeResponse dg1dx2:dg1e1dx2 = "2.0*x2-1.0"
// derivativeResponse dg1dx3:dg1e1dx3 = "2.0*x3+1.0"
// derivativeResponse dg1dx4:dg1e1dx4 = "2.0*x4-1.0"
Evaluator dg1e1dx1e1 = expression("dg1e1dx1e1", "2.0*x1+1",args(sm.getDesignVar("x1")));
Evaluator dg1e1dx2e1 = expression("dg1e1dx2e1", "2.0*x2-1.0",args(sm.getDesignVar("x2")));
Evaluator dg1e1dx3e1 = expression("dg1e1dx3e1", "2.0*x3+1.0",args(sm.getDesignVar("x3")));
Evaluator dg1e1dx4e1 = expression("dg1e1dx4e1", "2.0*x4-1.0",args(sm.getDesignVar("x4")));
List<Evaluator> g1e1g1 = list(dg1e1dx1e1, dg1e1dx2e1, dg1e1dx3e1, dg1e1dx4e1);
sm.setGradientEvaluators("g1", "g1e1", "g1e1g1",  g1e1g1);
// dg1e1dxie2
Evaluator dg1e1dx1e2 = new FiniteDifferenceEvaluator("dg1e1dx1e2", sm.getResponseVar("g1").getEvaluator("g1e1"), 0.1, "x1");
Evaluator dg1e1dx2e2 = new FiniteDifferenceEvaluator("dg1e1dx2e2", sm.getResponseVar("g1").getEvaluator("g1e1"), 0.1, "x2");
Evaluator dg1e1dx3e2 = new FiniteDifferenceEvaluator("dg1e1dx3e2", sm.getResponseVar("g1").getEvaluator("g1e1"), 0.1, "x3");
Evaluator dg1e1dx4e2 = new FiniteDifferenceEvaluator("dg1e1dx4e2", sm.getResponseVar("g1").getEvaluator("g1e1"), 0.1, "x4");
List<Evaluator> g1e1g2 = list(dg1e1dx1e2, dg1e1dx2e2, dg1e1dx3e2, dg1e1dx4e2);
sm.setGradientEvaluators("g1", "g1e1", "g1e1g2",  g1e1g2);
```

g1e2 with with Multi-Fidelity Sensitivities


```
 //g1: g1e2
// response g1:g1e2="f1+f2-2.0*x2+x3^2+x3+x4^2-x4-8.0"
// partial derivative dg1/df1="1.0"
// partial derivative dg1/df2="1.0"
// partial derivative dg1/dx2="-2.0"
// partial derivative dg1/dx3=2.0*x3+1.0"
// partial derivative dg1/dx4=2.0*x4-1.0"
// dg1e2dxie1
Evaluator dg1e2df1e1 = expression("dg1e2df1e1", "1.0");
Evaluator dg1e2df2e1 = expression("dg1e2df2e1", "1.0");
Evaluator dg1e2dx2e1 = expression("dg1e2dx2e1", "-2.0");
Evaluator dg1e2dx3e1 = expression("dg1e2dx3e1", "2.0*x3+1.0",args(sm.getDesignVar("x3")));
Evaluator dg1e2dx4e1 = expression("dg1e2dx4e1", "2.0*x4-1.0",args(sm.getDesignVar("x4")));
List<Evaluator> g1e2g1 = list(dg1e2df1e1, dg1e2df2e1, dg1e2dx2e1, dg1e2dx3e1, dg1e2dx4e1);
sm.setGradientEvaluators("g1", "g1e2", "g1e2g1",  g1e2g1);
// dg1e2dxie2
Evaluator dg1e2df1e2 = new FiniteDifferenceEvaluator("dg1e2df1e2", sm.getResponseVar("g1").getEvaluator("g1e2"), 0.1, "f1");
Evaluator dg1e2df2e2 = new FiniteDifferenceEvaluator("dg1e2df2e2", sm.getResponseVar("g1").getEvaluator("g1e2"), 0.1, "f2");
Evaluator dg1e2dx2e2 = new FiniteDifferenceEvaluator("dg1e2dx2e2", sm.getResponseVar("g1").getEvaluator("g1e2"), 0.1, "x2");
Evaluator dg1e2dx3e2 = new FiniteDifferenceEvaluator("dg1e2dx3e2", sm.getResponseVar("g1").getEvaluator("g1e2"), 0.1, "x3");
Evaluator dg1e2dx4e2 = new FiniteDifferenceEvaluator("dg1e2dx4e2", sm.getResponseVar("g1").getEvaluator("g1e2"), 0.1, "x4");
List<Evaluator> g1e2g2 = list(dg1e2df1e2, dg1e2df2e2, dg1e2dx2e2, dg1e2dx3e2, dg1e2dx4e2);
sm.setGradientEvaluators("g1", "g1e2", "g1e2g2",  g1e2g2);
```

g2,g3 Sensitivities

```
// response g2="x1^2-x1+2.0*x2^2+x3^2+2.0*x4^2-x4-10.0"
// partial derivative dg2dx1 = "2.0*x1-1.0"
// partial derivative dg2dx2 = "4.0*x2"
// partial derivative dg2dx3 = "2.0*x3"
// partial derivative dg2dx4 = "4.0*x4-1.0"
Evaluator dg2edx1e1 = expression("dg2edx1e1", "2.0*x1-1",args(sm.getDesignVar("x1")));
Evaluator dg2edx2e1 = expression("dg2edx2e1", "4.0*x2",args(sm.getDesignVar("x2")));
Evaluator dg2edx3e1 = expression("dg2edx3e1", "2.0*x3",args(sm.getDesignVar("x3")));
Evaluator dg2edx4e1 = expression("dg2edx4e1", "4.0*x4-1.0",args(sm.getDesignVar("x4")));
List<Evaluator> g2eg1 = list(dg2edx1e1, dg2edx2e1, dg2edx3e1, dg2edx4e1);
sm.setGradientEvaluators("g2", "g2e", "g2eg1",  g2eg1);


// response g3="2.0*x1^2+2.0*x1+x2^2-x2+x3^2-x4-5.0"
// partial derivative dg3dx1 = "4.0*x1+2.0"
// partial derivative dg3dx2 = "2.0*x2-1.0"
// partial derivative dg3dx3 = "2.0*x3"
// partial derivative dg3dx4 = "-1.0"
Evaluator dg3edx1e1 = expression("dg3edx1e1", "4.0*x1+2.0",args(sm.getDesignVar("x1")));
Evaluator dg3edx2e1 = expression("dg3edx2e1", "2.0*x2-1.0",args(sm.getDesignVar("x2")));
Evaluator dg3edx3e1 = expression("dg3edx3e1", "2.0*x3",args(sm.getDesignVar("x3")));
Evaluator dg3edx4e1 = expression("dg3edx4e1", "-1.0",args());
List<Evaluator> g3eg1 = list(dg3edx1e1, dg3edx2e1, dg3edx3e1, dg3edx4e1);
sm.setGradientEvaluators("g3", "g3e", "g3eg1",  g3eg1);
```

Using the Multi-Fidelity Model


```
sm.setDesignVarValues(set("x1", 2.0), set("x2", 3.0), set("x3", 4.0), set("x4", 5.0));
sm.setParameterVarValues(set("p1", 50.0));
//logger.info("\n\ndoSensitivityAnalysis out: " + sm.getSensitivities());

Var f = sm.getResponseVar("f");
logger.info("\n fe1 fe1g2 **********:\n "+sm.getPartialGradients() );
// Check the partial derivative calculations
//f
// fe1/g1 partials
f.selectEvaluator("fe1");
logger.info("\n current f function value:\n" + f.getValue());
//logger.info("\n fe1 fe1g1 **********:\n" );
//logger.info("-- partial derivative f wrt x1 with evaluator fe1 gradient fe1g1 fe1|fe1g1: " + f.getPartialDerivative("x1", "fe1g1"));
//logger.info("-- partial derivative f wrt x2 with evaluator fe1 gradient fe1g1 fe1|fe1g1: " + f.getPartialDerivative("x2", "fe1g1"));
//logger.info("-- partial derivative f wrt x3 with evaluator fe1 gradient fe1g1 fe1|fe1g1: " + f.getPartialDerivative("x3", "fe1g1"));
//logger.info("-- partial derivative f wrt x4 with evaluator fe1 gradient fe1g1 fe1|fe1g1: " + f.getPartialDerivative("x4", "fe1g1"));
//logger.info("-- partial gradient f wrt xi with evaluator fe1 gradient fe1g1 fe1|fe1g1: " + f.getPartialGradient("fe1g1"));
// fe1/g2 partials


//logger.info("-- partial derivative f wrt x1 with evaluator fe1 gradient fe1g2 fe1|fe1g2: " + f.getPartialDerivative("x1", "fe1g2"));
//logger.info("-- partial derivative f wrt x2 with evaluator fe1 gradient fe1g2 fe1|fe1g2: " + f.getPartialDerivative("x2", "fe1g2"));
//logger.info("-- partial derivative f wrt x3 with evaluator fe1 gradient fe1g2 fe1|fe1g2: " + f.getPartialDerivative("x3", "fe1g2"));
//logger.info("-- partial derivative f wrt x4 with evaluator fe1 gradient fe1g2 fe1|fe1g2: " + f.getPartialDerivative("x4", "fe1g2"));
//logger.info("-- partial gradient f wrt xi with evaluator fe1 gradient fe1g2 fe1|fe1g2: " + f.getPartialGradient("fe1g2"));

// fe2/g1 partials
f.selectEvaluator("fe2");
logger.info("\n fe2 fe2g1 **********:\n" );
logger.info("-- partial derivative f wrt f1 with evaluator fe2 gradient fe2g1 fe1|feg1: " + f.getPartialDerivative("f1", "fe2g1"));
logger.info("-- partial derivative f wrt f3 with evaluator fe2 gradient fe2g1 fe1|feg1: " + f.getPartialDerivative("f3", "fe2g1"));
logger.info("-- partial derivative f wrt f4 with evaluator fe2 gradient fe2g1 fe1|feg1: " + f.getPartialDerivative("f4", "fe2g1"));
logger.info("-- partial derivative f wrt x3 with evaluator fe2 gradient fe2g1 fe1|feg1: " + f.getPartialDerivative("x3", "fe2g1"));
logger.info("-- partial derivative f wrt xl1 with evaluator fe2 gradient fe2g1 fe1|feg1: " + f.getPartialDerivative("xl1", "fe2g1"));
```

MSTC SORCER Engineering 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

Shape and sizing aeroelastic optimization

Minimize
Objective Function

$$f(\beta_i) = W_{Structural}$$

Subject to

Inequality Constraints

$$g_1(\beta_i) = 1.0 - \frac{C_{M_{roll}}}{C_{M_{roll},ref}}$$

$$g_j(\beta_i) = 1.0 - \frac{\sigma_{VonMises}}{\sigma_{MaxAllowable}} \leq 0 \qquad j = 2, N_{Constr.}$$

Side Constraints

$$\beta_i^L \leq \beta_i \leq \beta_i^U \qquad i = 1, N_{DesignVariables}$$

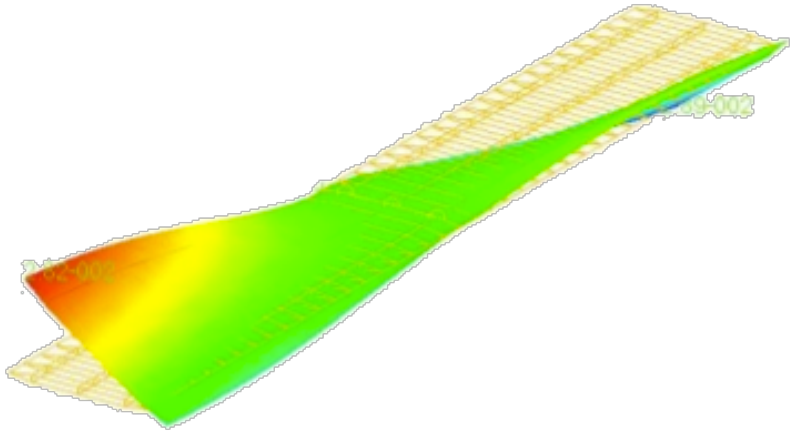
β_i - Structural sizes, and planform sweep

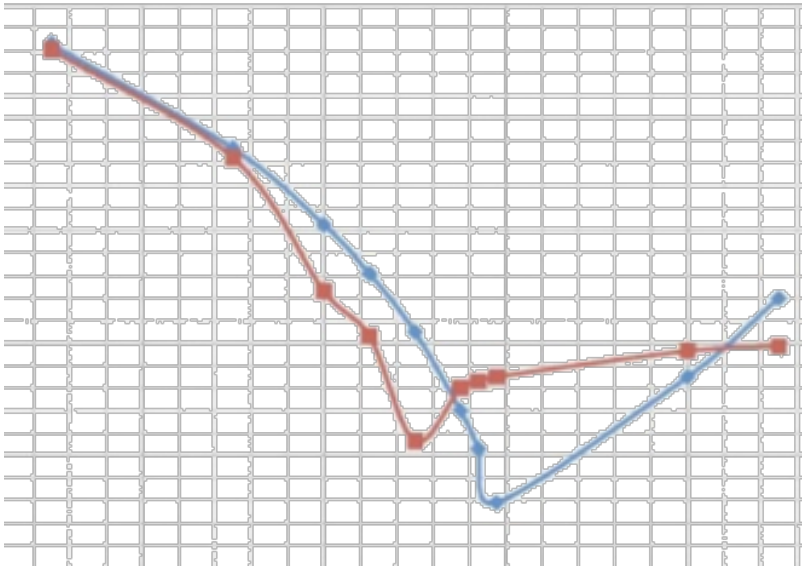
Ernest Thompson – AFRL/RBSD

Min weight shape & sizing opti

SORCER

- Euler Static Aeroelastic Calculations





Ernest Thompson – AFRL/RBSD

Sorcer network configuration for min weight (shape and sizing)

SORCER

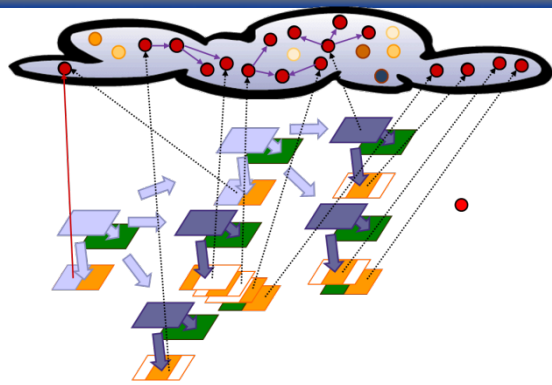
Coalescence

MDICE

Avus

RBSDZ0010143

Microsoft Windows



NMEDA

MDICE

Avus

Whittle

Gridgen

DOT

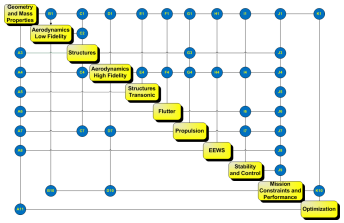
NASTRAN

Gladius

StructureTransform

NASTRAN

Client



Xserv2

MDICE

Avus

Gothmog

Java

JavaSpace

Ernest Thompson – AFRL/RBSD

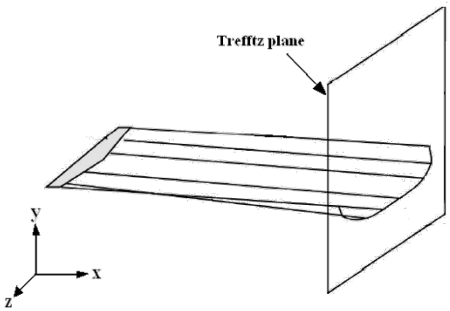
Concluding Remarks

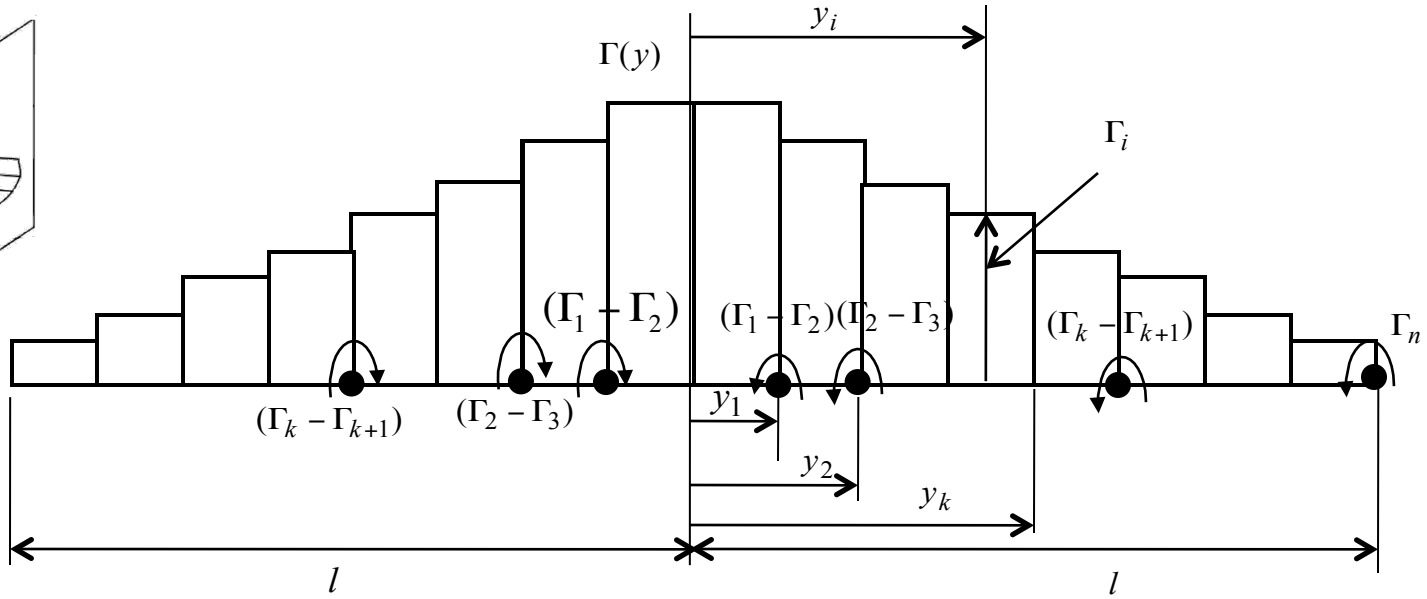
- MSTC is responsible for executing large scale system level MDA/MDO in a heterogenous computing environment
- SORCER exposes Sun's Jini connection Technology to create a service oriented programming and computing environment for large scale MDA/MDO
- SORCER Functionality
  - Create providers and expose them as jini services.
  - Exertion Oriented Programming – Ability to orchastrate Services
  - Models for analysis, sensitvitities, and design
  - VEF – Functional demand driven computations
  - Space Computing

Backups

Engineering Example

Trefftz-Plane Induced Drag With Euler Aerodynamics





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

$Lpus_j(\alpha, \beta_i)$ – Lift per unit span (Euler calculations)

α - angle of attack

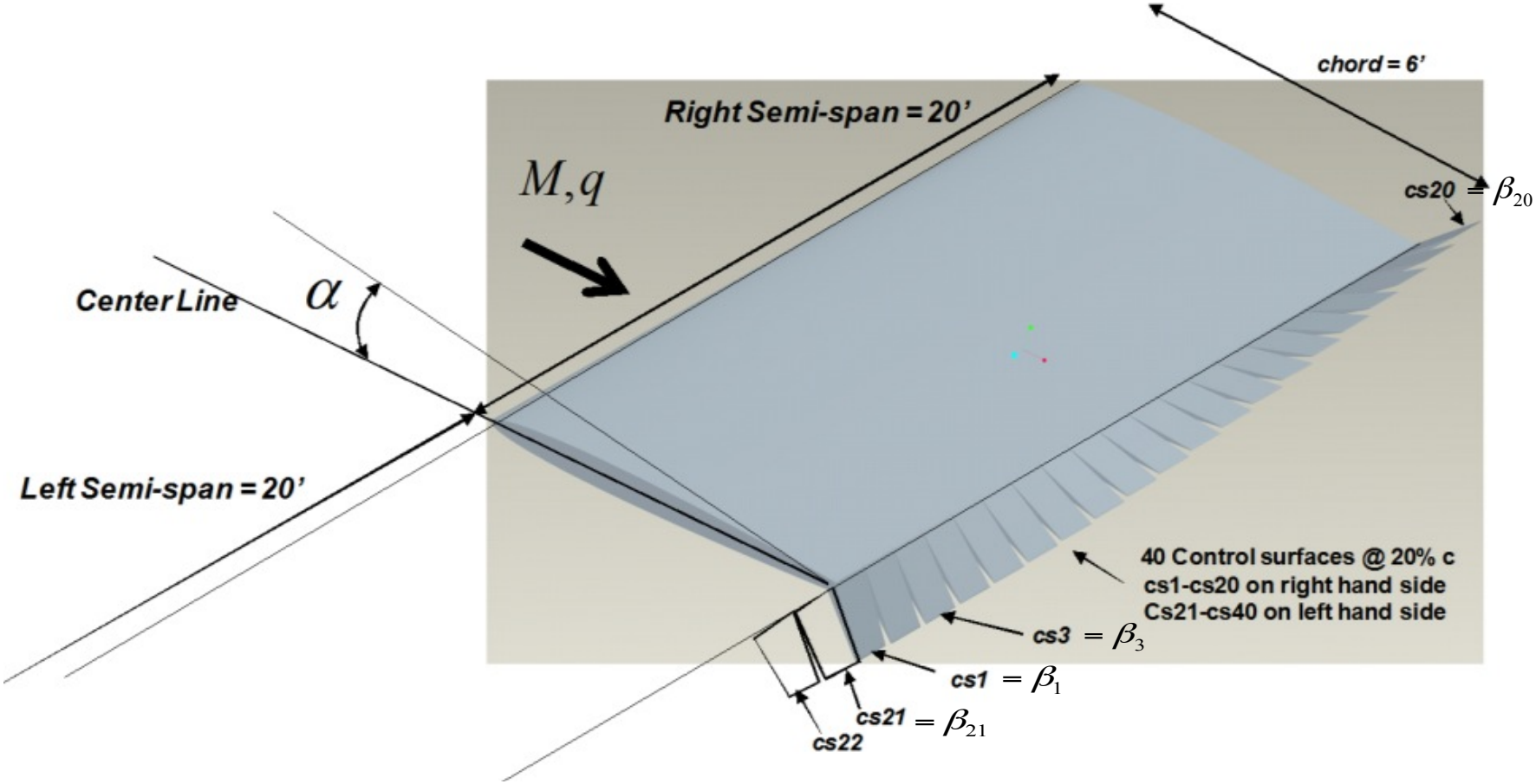
β_i - control surface deflection of ith surface

$$Lpus_k = \rho_\infty V_\infty \Gamma_k \quad \text{(Kutta-Joukowski theorem)}$$

$$\text{Design Variables } \bar{x} = \{\alpha, \beta_i\}^T$$

$$\text{Note: } q = \frac{1}{2}\rho_\infty V_\infty^2 = \frac{1}{2}\gamma p_s M^2$$

Example Wing with Control Surfaces



Optimization Problem

$$\text{Minimize:} \quad D_I(Lpus_i(\alpha,\beta_i,\beta l_i),q)$$

$$\text{Such That:} \quad L_T(Lpus_i(\alpha,\beta_i,\beta l_i)) = L$$

$$-5.0 \le \alpha \le 5.0$$

$$-10.0 \le \beta_i \le 10.0$$

$$\text{Design Variables } x_i = \{\alpha,\beta_{1-20},M,\gamma,p_s\}^T$$

$$\text{Linked Variables } \beta l_i = \{\beta l_{21-40} = \beta_{1-20}\}^T$$

$$\text{Response Variables: } D_I(Lpus_i(\alpha,\beta_1,\beta l_i),q),\ L_T(Lpus_i(\alpha,\beta_1,\beta l_i)),$$
$$Lpus_i(\alpha,\beta_1,\beta l_i),\ q(M,\gamma,p_s)$$

$D_I(Lpus_i(\alpha,\beta_1,\beta l_i),\text{q})$ and $L_T(Lpus_i(\alpha,\beta_1,\beta l_i))$ are functions of fuctions and $Lpus_i(\alpha,\beta_1,\beta l_i)$ is a function of the independent and linked design variables

Multi-Fidelity Responses

For the responses

$$D_I(Lpus_i(\alpha,\beta_i,\beta l_i),q), L_T(Lpus_i(\alpha,\beta_i,\beta l_i)), \text{ and } Lpus_i(\alpha,\beta_i,\beta l_i)$$

Multi ways of computing these functions are desired: This is necessary for efficiency and accuracy during the optimization process.

Realizations for $D_I(Lpus_i(\alpha,\beta_i,\beta l_i),q)$ – DIExact, DISOA, DIMOA, DIKrig

Realizations for $L_T(Lpus_i(\alpha,\beta_i,\beta l_i))$ – LTExact, LTSOA, LTMOA

Realizations for $Lpus_i(\alpha,\beta_i,\beta l_i))$ – LpusiExact, LpusiSOA

SOA – Standard One Point Approximation
MOA – Modified One Point Approximation
Krig – Kriging Model

Standard One Point Approximation (SOA)

$$\tilde{f}(\bar{x}) = f(\bar{x}^0) + \nabla f(\bar{x}^0)\{\bar{x} - \bar{x}^0\}$$

$$\frac{\partial \tilde{f}(\bar{x})}{\partial x_i} = \frac{\partial f(\bar{x}^0)}{\partial x_i}$$

In terms of the induced drag function

$$\tilde{D}_I(\bar{x}) = D_I(\bar{x}^0) + \nabla D_I(\bar{x}^0)\{\bar{x} - \bar{x}^0\}$$

$$\frac{\partial \tilde{D}_I(\bar{x})}{\partial x_i} = \frac{\partial \tilde{D}_I(\bar{x}^0)}{\partial x_i} = \text{constant}_i$$

Modified One Point Approximation (MOA)

Apply SOA to the lift per unit span Lpusi not to D_I

$$\widetilde{L}pus_i(\bar{x}) = \widetilde{L}pus_i(\bar{x}^0) + \nabla\widetilde{L}pus_i(\bar{x}^0)\{\bar{x} - \bar{x}^0\}$$

$$\widetilde{D}_I(\bar{x}) = \frac{-1}{4\pi\rho_\infty V_0^2}\sum_{i=1}^{n}\widetilde{L}pus_i\Delta y_i\sum_{k=1}^{n}\left[\widetilde{L}pus_k - \widetilde{L}pus_{k+1}\right]\left[\frac{1}{y_i - y_k} - \frac{1}{y_i + y_k}\right]$$

$$\frac{\partial\widetilde{D}_I(\bar{x})}{\partial x_i} \neq \text{constant}$$

Kriging Approximation

$$\tilde{y}(\bar{x}) = \sum_{i=0}^{p-1} \beta_i \phi_i(\bar{x}) + Z(\bar{x})$$

$\sum_{i=0}^{p-1} \beta_i \phi_i(\bar{x})$ *Is a generalized regression model (often chosen as a constant in practice)*

$\phi_i(\bar{x})$ *Are a set of chosen basis functions*

β_i *Are found by solving* $F^T R^{-1} F \bar{\beta} = F^T R^{-1} Y$ *(least squares if R is identity)*

$F_{i,j} = \phi_j(\bar{x}_j)$

$R(\bar{\theta})$ *Is a correlation matrix*

Y *Vector of known responses at random sampling points*

Constant Regression and Full 2nd Order Regression with monomials

- ***LDS Sampling and adaptive LDS Sampling***

Kriging Approximation

$$\tilde{f}(\bar{x}) = \sum_{i=0}^{p-1} \beta_i \phi_i(\bar{x}) + Z(\bar{x})$$

$Z(\bar{x})$ ***is one realization of a Gaussian process with mean of zero, variance*** σ^2 ***, and a covariance that satisfies***

$$Cov[Z(\bar{x}^i), Z(\bar{x}^j)] = \sigma_Z^2 \bar{R}[R(\bar{x}^i, \bar{x}^j)]$$

R ***Is the correlation function. Two considered in this work are***

$$R(\bar{\theta}, \bar{x}) = \prod_{k=1}^{n} e^{-\left(\frac{x_k}{\theta_k}\right)^2}$$ ***(Gaussian correlation function)***

$$R(\bar{\theta}, \bar{x}) = s(z)$$

$$z = \theta |x|$$

$$s(z) = \begin{cases} 1 - \frac{3}{a} z^2 + \frac{1+a}{a^2} z^3, & 0 \le z \le a \\ \frac{1}{1-a}(1-z)^3, & a \le z \le 1 \\ 0, & z \ge 1 \end{cases}$$ ***(Cubic Spline correlation function)***

Optimization Problem

Minimize: $D_I(Lpus_i(\alpha,\beta_i,\beta l_i),q)$

Such That: $L_T(Lpus_i(\alpha,\beta_i,\beta l_i)) = L$

$-5.0 \le \alpha \le 5.0$

$-10.0 \le \beta_i \le 10.0$

Design Variables $x_i = \{\alpha,\beta_{1-20},M,\gamma,p_s\}^T$

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

Response Variables: $D_I(Lpus_i(\alpha,\beta_1,\beta l_i),q)$, $L_T(Lpus_i(\alpha,\beta_1,\beta l_i))$, $Lpus_i(\alpha,\beta_1,\beta l_i)$, $q(M,\gamma,p_s)$

$D_I(Lpus_i(\alpha,\beta_1,\beta l_i),$q) and $L_T(Lpus_i(\alpha,\beta_1,\beta l_i))$ are functions of fuctions and $Lpus_i(\alpha,\beta_1,\beta l_i)$ is a function of the independent and linked design variables

Optimization Model Definition In SORCER

```
OptimizationModel omodel = responseModel("Induced Drag",
designVars(vars(loop("i",1,20),"beta$i$", 0.0, -10.0, 10.0)),
linkedVars(names(loop("i",21,40),"betal$i$")),
designVars(var("alpha", 1.0, -5.0, 5.0)),
designVars("mach", "gamma","pstatic"),

responseVars("DI",
    realization((,
        evaluation("DIExacte",
            differentiation(wrt(loop("i",1,20),"beta$i$", "alpha","q"), gradient("DIExacteg1")),
        evaluation("DISOAe"),
            differentiation(wrt(loop("i",1,20),"beta$i$", "alpha","q"), gradient("DISOAeg1")),
        evaluation("DIMOAe"),
            differentiation(wrt(loop("i",1,20),"beta$i$", "alpha","q"), gradient("DIMOAeg1")),
        evaluatioin("DIKrige"),
            differentiation(wrt(loop("i",1,20),"beta$i$", "alpha","q"), gradient("DIKrigeg1")),

responseVars("LT",
   realization((
        evaluation("LTExacte",
            differentiation(wrt(loop("i",1,20),"beta$i$", "alpha","q"), gradient("LTExacteg1")),
        evaluation("LTMOAe"),
            differentiation(wrt(loop("i",1,20),"beta$i$", "alpha","q"), gradient("LTMOAeg1")),

responseVars(names(loop("i",1,20),Lpus$i$"),
realization(loop("i",1,20),"Lpus$i$",
   evaluation("Lpus$i$Exacte",
       differentiation(wrt(loop("k",1,20),"beta$k$", "alpha"), gradient("Lpus$i$Exacteg1")),
   evaluation("Lpus$i$SOAe",
       differentiation(wrt(loop("k",1,20),"beta$k$", "alpha"), gradient("Lpus$i$SOAeg1")),

responseVars("q",
    realization(evaluation("qExact", differentiation( wrt("mach","gamma","pstatic"), gradient("qExactg1")))),

objectiveVars(var("DIo", "DI", Target.min )),
constraintVars(var("LTc", "LT", Relation.eq, 1000.0),
);
```

Note: Multi-fidelity for responses

Optimization Model Configuration

- Need to Configure the Variables
  - In General the configuration of a variable consists of one or more *Evaluations*.
    - Each *Evaluation* is an *Evaluator*, *Filter*, *Differentiation* triplet
  - Design Variables have a default Evaluator. Hence only a Filter (if necessary) needs to be developed to Complete the configuration of the Design Variables.
    - For this example all design variables reside in ascii text files

Design Variable Configuration

- Each independent design variable configuration consists of constructing zero, one, or more *Filter*s.
- For this case the independent design variables reside in two locations. The beta_i variables reside in an ascii text file(AVUS_Boundary_Condition.dat)
- Alpha, mach, gamma, and pstatic reside in a separate ascii text file (AVUS_Job.job).
- In order to have the ability to get and set the values in these files from anywhere on the network (we have no idea where avus may run) we need to create *Filter* objects and place them into the Variables.
- Filters take a larger entity and "filter" out a portion of that entity for either reading, writing or passing to another filter.
- Since we are working with ascii text files in this case we will create BasicFileFilter objects (a list of the all currently available *Filter* types can be found in the sorcer.vfe.filter package) .

Design Variable Configuration

- Each *BasicFileFilter* requires a *Pattern* object which is used to locate the desired quantity in the entity.
- Here our pattern is essentially *line, field, delimiter* information.
- Other patterns such as regular expressions are available as well.
- Below are excerpts from the AVUS_Boundary_Condition.dat file and the AVUS_Job.job file(the line numbers are not actually in the file. They are shown here to indicate where in the file the excerpt has been taken from).

Design Variable Configuration

```
187 Upper CS01
188 Transpiration
189 Boundary
190 Yes
191 TBC-Type  Rx  Ry  Rz  Thetax Thetay Thetaz
192 1 4.5 0.0 0.0 0.0 12.0 0.0
193 ##################################################################
194  32
195 Upper CS02
196 Transpiration
197 Boundary
198 Yes
199 TBC-Type  Rx  Ry  Rz  Thetax Thetay Thetaz
200 1 4.5 0.0 0.0 0.0 0.0 0.0
201 ##################################################################
202  33
203 Upper CS03
204 Transpiration
205 Boundary
206 Yes
207 TBC-Type  Rx  Ry  Rz  Thetax Thetay Thetaz
208 1 4.5 0.0 0.0 0.0 0.0 0.0
209 ##################################################################
210  34
211 Upper CS04
212 Transpiration
213 Boundary
214 Yes
215 TBC-Type  Rx  Ry  Rz  Thetax Thetay Thetaz
216 1 4.5 0.0 0.0 0.0 0.0 0.0
217 ##################################################################
```

Excerpt from AVUS_Boundary_Condition.dat

Design Variable Configuration

```
94 ***********************************************************
95 UNITS (1=MKS, 2=CGS, 3=FOOT-SLUG-SEC, 4=INCH-SNAIL-SEC)
96   3
97 MACH NO.    ANGLE OF ATTACK    ANGLE OF SIDESLIP
98 0.85 5.14159 0.0
99 STATIC PRESSURE     STATIC TEMPERATURE
100   -1.               -1.
101 GAMMA     GAS CONSTANT     PRANDTL NUMBER     GRAVITY
102   -1.        -1.              -1.                0.
103 ***********************************************************
104                INITIAL CONDITIONS
```

Excerpt from AVUS_Job.job file

Design Variable Configuration

Below is the syntax for creating the *Pattern* objects for the filters.

```
// create the patterns for the betai filters
Pattern beta1p= new BasicPattern(“beta1", "File", "Double", 192, 6, " ");
Pattern beta2p = new BasicPattern(“beta2", "File", "Double", 200, 6, " " );
Pattern beta3p = new BasicPattern(“beta3", "File", "Double", 208, 6, " " );
Pattern beta4p = new BasicPattern(“beta4", "File", "Double", 216, 6, " " );
.
..
Pattern alphap = new BasicPattern("alpha", "File", "Double", 98, 2, " ");
Pattern machp = new BasicPattern(“mach", "File", "Double", 198, 3, " ");
Pattern gammap = new BasicPattern(“gamma", "File", "Double", 298, 4, " ");
Pattern pstaticp = new BasicPattern(“pstatic", "File", "Double", 398, 4, " ");
```

The arguments for the BasicPattern consturcution are the String *patternName , String patternType, String dataType, int line, int field, String delimiter .*

For the above cases we see that beta1p identifies the value at line 192, field 6, using a “space” delimiter between each field. Hence it refereces the Thetay value that currently holds a value = 12.0.

Design Variable Configuration

- **Once all of the Patterns are created then the *BasicFileFilters* can be constructed for each item.**

```
BasicFileFilter beta1bff = new BasicFileFilter(bcURL, beta1p);
BasicFileFilter beta2bff = new BasicFileFilter(bcURL, beta2p);
BasicFileFilter beta3bff = new BasicFileFilter(bcURL, beta3p);
BasicFileFilter beta4bff = new BasicFileFilter(bcURL, beta4p);
.
..
BasicFileFilter abff = new BasicFileFilter(avusjobURL, alphap);
BasicFileFilter machbff = new BasicFileFilter(avusjobURL, machp);
BasicFileFilter gammabff = new BasicFileFilter(avusjobURL, gammap);
BasicFileFilter pstaticbff = new BasicFileFilter(avusjobURL,pstaticp);
Need to create a Map mapOfBetaiFilters = map(pair("beta1",beta1bff),
                                             pair("beta2",beta2bff),
                                             …
                                             pair("beta2",beta20bff),
                                             ));
```

The arguments for the *BasicFileFilter* constructor used here(there are other constructors for *BasicFileFilter*) are a URL object and a Pattern object. The URL objects are references to the respective files; AVUS_Boundary_Condition.dat for all the betai and AVUS_Job.job for alpha,mach,gamma and pstatic.

Design Variable Configuration

- **The final step is to add the *BasicFileFilter* objects to each of the design Variables. The below lines add the filters to the design variables in the model.**

vars(omodel, loop(“i”,1,20), “beta$i$”, mapOfBetaiFilters);

vars(omodel, var(“alpha”, abff), var(“mach”,machbff), var(“gamma”, gammabff), var(“pstatic”,pstaticbff)) ;

This completes the configuration of all *DesignVariables*

Linked Variable Configuration

- The ***Linked Variable***s are related to the ***Independent Variables*** by a set relationship that are computed by an ***Evaluator.*** For this case there are 20 Linked Variables with the relationship.

$$\text{Linked Variables } \beta l_i = \{\beta_{21-40} = \beta_{1-20}\}^T$$

vars(omodel, loop(list("i:21","k:1"),20), betal$i$", evaluator("betal$i$,"1.0*beta$k$"),args("beta$k$"));

Response Variables – Evaluators, Filters, and Differentiation

- The Response Variables expose calculated quantities or engineering responses computed within the SORCER environment.

- Response Variables consists of an ***Evaluator*** and one or more ***Filters*** and ***Differentiation***. This triplet is termed an ***Evaluation***.

- Evaluators are the actual compute engines and they have potential dependencies. (other Response, Design, and indirectly Linked Variables) as their input.

- In SORCER a Response Variable, like all Variables, **must reduce to a single scalar**
- ***Evaluator***s compute "blobs" (large amount of data)
- ***Filter***s are used to "filter" the "blobs" to the scalar value the response Variable represents

Response Variable Configuration (Evaluator Creation)

Lpus$i$Exacte – *Evaluator*

Task to execute Avus

Lpus$i$Exacteg1 – *DerivativeEvaluator*

Finite Difference

$$\frac{\partial Lpus_{iExacteg1}}{\partial x_j} = (Lpus_{iExacte}(x_j + \Delta x_j) - Lpus_{iExacte}(x_j))/\Delta x_j$$

- There are 20 response Variables $Lpus_j(\alpha, \beta_i)$
- All 20 are computed by performing a single Avus run. (All use the same *Evaluator*)
- This *Evaluator* executes Avus for a given set of alpha and betai and produces an *AvusOutput* Object(engineering.provider.auvs.AvusOutput.java)
- The type of *Evaluator* used is an *ExertionEvaluator*. An *ExertionEvaluator* can be either a single Task or an entire Job.

```
// configure the lpusi response variables. For the LpusiExacte - Evaluator
//construct the exertion evaluator for the lift per unit span (Lpusi)

// Construct the avusTask(see Appendix B for this method and the construction of the context for this task)
Task avusTask = getAvusTask();
Evaluator avusEvaluator = avusTask.getEvaluator();
// add the dependencies to the avusEvaluator evaluator (all betai and alpha)
avusEvaluator.addDependents(loop(1:20),"beta%i%","alpha");
```

Evaluator avusEvaluator

Response Variable Configuration (Evaluator Creation)

Lpus%i%Exact – *Evaluator*

- Once the Evaluator with it's dependencies has been constructed
- Each response Variable sets it's Evaluator to the avusEvaluator.

// Set the Evaluator for each of the Lpusi response variables (assumes the first 20 // response variables in the rvs model are the Lpus)

```
rvs.get(1).setEvaluator(avusEvaluator);
rvs.get(2).setEvaluator(avusEvaluator);
.
.
.
rvs.get(20).setEvaluator(avusEvaluator);
```

How to do this via composition?
Evals(model,loop(1:20),"Lpus%i%",avusEvaluator);
Do we wait until we have the filters to do this and add the dependencies to the Evaluator in a single composition?

Response Variable Configuration (Filter Creation)

Lpus%i%Exact – *Filter*

- Now construct the Filters for each Lpus_i response Variable.

- The goal is to filter from the result of the Evaluator (Task in this case) to the individual values for the Lpusi

- To do this examine the contents of the Avus Task

Task – Avus Task

MethodSignature – AvusMethod ServiceSignature("executeAvus", engineering.provider.avus.AvusRemoteInterface.class)

Context - AvusContext
"AVUS/BCFILE",URL_to_AVUS_BC_FILE
"AVUS/JOBFILE", URL_to_AVUS_JOB_FILE
"AVUS/RESTARTFILE", URL_to_AVUS_JOB_RESTART_FILE
"AVUS/GRIDFILE", URL_to_AVUS_GRID_FILE
"AVUS/TAPFILE",URL_to_AVUS_TAP_FILE
"AVUS/TRIPFILE", URL_to_AVUS_TRIP_FILE
"AVUS/AVUSOUTPUT",AVUS_OUTPUT_OBJECT

Response Variable Configuration (Filter Creation)

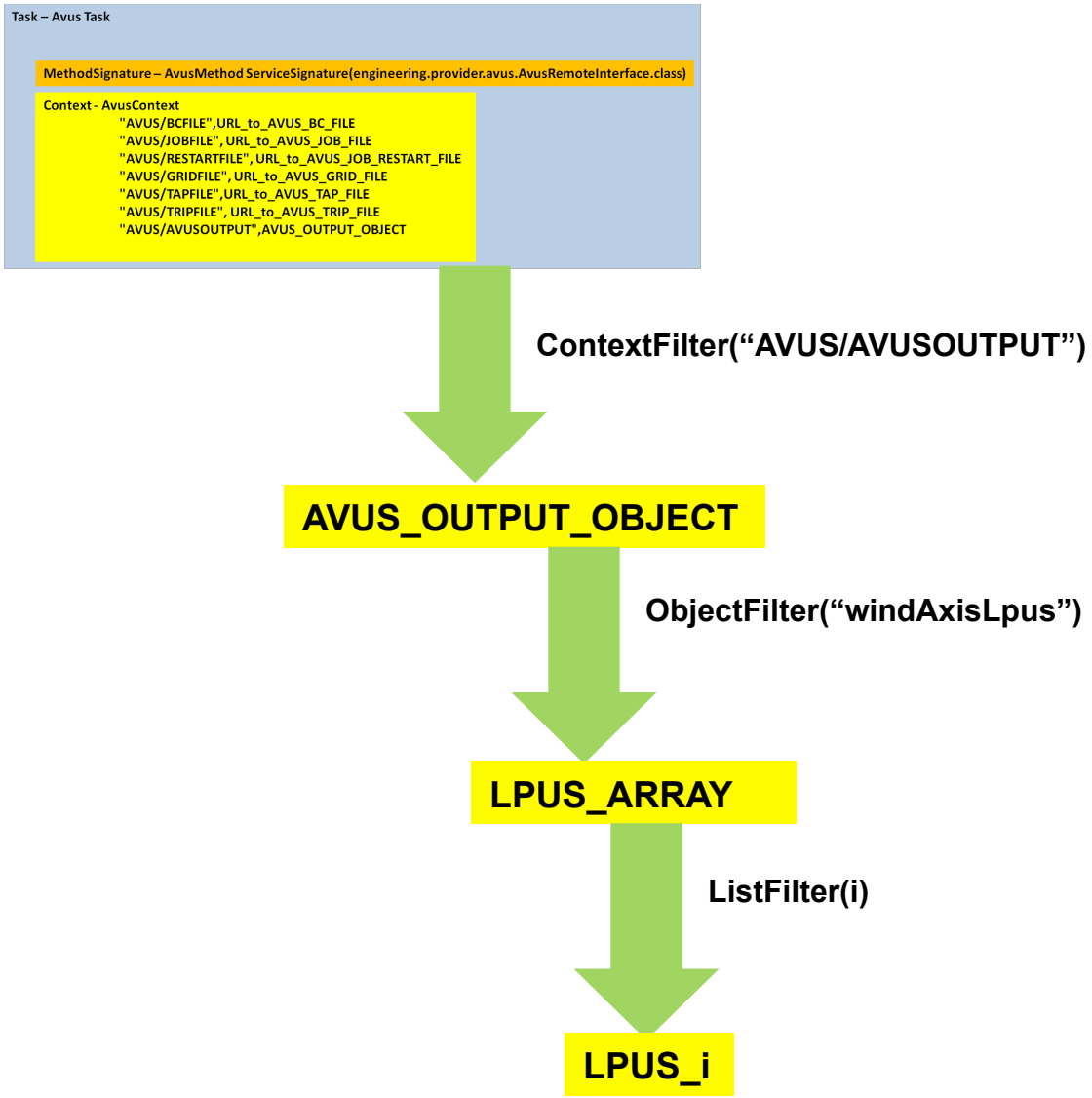
Lpus%i%Exact – *Filter*

- The Context contains the input and output produced by executing the task/Evaluator.
- Last entry has the path “AVUS/AVUSOUTPUT” and the data AVUS_OUTPUT_OBJECT .
- The AVUS_OUTPUT_OBJECT (AvusOutput.java) contains a field “windAxisLpus”
- This field is an a array containing the Lpusi values.
- Hence a set of filters are necessary to do the following:

1. extract the AVUS_OUTPUT_OBJECT from the Task (*ContextFilter*)
2. extract the “windAxisLpus” field from the AVUS_OUTPUT_OBJECT (*ObjectFilter*)
3. extract the ith entry from the array (*ListFilter*)

Response Variable Configuration (Filter Creation)

Lpus%i%Exact – *Filter*



Response Variable Configuration (Filter Creation)

// create the filter pipeline for each of the lpus rvs's
// ***ContextFilter*** that extracts the AVUS_OUTPUT_OBJECT from the Evaluator(AvusTask)
// Context at the path location of "AVUS/AVUSOUTPUT"

```
ContextFilter contextFilter = new ContextFilter("avusTaskContext", "AVUS/AVUSOUTPUT");
```

// ***ObjectFilter*** to extract the windAxisLpus field from the AVUS_OUTPUT_OBJECT

```
ObjectFilter objFilter = new ObjectFilter("windAxisLpus");
```

// ***ListFilter*** to extract the ith Lpus from the windAxisLpus Array

```
ListFilter lf1 = new ListFilter(1);
ListFilter lf2 = new ListFilter(2);
ListFilter lf3 = new ListFilter(3);
.
ListFilter lf20 = new ListFilter(20);
```

LpusiExact – *Filter*

```
Need
List (loop(1,20),ListFilter)
Map(loop("i",20),ListFilter,"string$i$")
List|Map(loop("i",20),ClassName,ListofItems)
ListFilter[] lfsi = filters(loop(1:20), "lf%i%");
```

// Finally create **the pipeline of filters** (filtering will occur in order specified left to right)
List(Loop,Class, lists …)

```
Filter rvFilter1= new Filter(contextFilter, objFilter, lf1);
```

// Set the filter on the ith(1st here) Response Variable

```
rvs.get(1).setFilter(rvFilter1);
Filter rvFilter = new Filter(contextFilter, objFilter, lf2);
rvs.get(2).setFilter(rvFilter2);
Filter rvFilter = new Filter(contextFilter, objFilter, lf3);
```

Lpusi Configuration

Lpus%i%Exact – Composition

What does composition look like for Lpusi?

var(<modelName>,loop(i:j),”<varName>,<evaluator>,<filterName>,<argNames>)

vars(omodel,loop(“I”,20),“Lpus$i$”,avusEvaluator, pair(“Lpus$i$, mapofPiplineFilters),args (names (loop(“k”,1,20),”beta$k$”),”alpha”));

Lpusi Sensitivities Configuration

Lpus%i%Exacteg1 – *DerivaviteEvaluator*

Finite Difference

$$\frac{\partial Lpus_{iExacteg1}}{\partial x_j} = (Lpus_{iExacte}(x_j + \Delta x_j) - Lpus_{iExacte}(x_j))/\Delta x_j$$

Evaluator dLpus%k%Exactedbeta%i% = **new FiniteDifferenceEvaluator("**dLpus%k%Exactedbeta%i%**", sm.getResponseVar("Lpus%k%").getEvaluator("Lpus%k%Exacte"),0.1, "beta%i%");**

Evaluator dLpus%k%Exactedalpha = **new FiniteDifferenceEvaluator("**dLpus%k%Exactedalpha**", sm.getResponseVar("Lpus%k%").getEvaluator("Lpus%k%Execte"),0.1, "beta%i%");**

List<Evaluator> LpusExacte%k%g1= *list(*dLpus%k%Exactedbeta%i% , dLpus%k%Exactedalpha *);*

model.setGradientEvaluators("Lpus%k%", "Lpus%k%Exact", "LpusExacte%k%g1", LpusExacte%k%g1);

Evaluator dLpus$k$dbeta$i$e1 = **new FiniteDifferenceEvaluator("dLpus$k$dbeta$i$e1", sm.getResponseVar("Lpus$K$").getEvaluator("Lpus$k$Exacte"),0.1, "beta$i$");**
Evaluator dLpus1dbeta1e1 = **new FiniteDifferenceEvaluator("dLpus1dbeta1e1", sm.getResponseVar("Lpus1").getEvaluator("Lpus1Exacte"),0.1, "x1");**
Evaluator dLpus1dbeta2e1 = **new FiniteDifferenceEvaluator("dLpus1dbeta2e1", sm.getResponseVar("Lpus1").getEvaluator("Lpus1Exacte"),0.1, "x1");**
List fdEvals = fdEvaluators(model, "Lpus1", "Lpus1Exacte", "Lpus1Exacteg1", pair("beta1", .01), pair("beta2", .01), pair("alpha", .1));
//Evaluator(omodel, loop(20), "Lpus$i$","Lpus$i$Exacte", "Lpus$i$Exacteg1", .1);
evaluators(omodel, loop(20),
fdEvaluators("Lpus$i$", "Lpus$i$Exacte", "Lpus1Exacteg1", pair("beta1", .01), pair("beta2", .01), pair("alpha"));
);
evaluators(omodel, loop(20),
fdEvaluators("Lpus$i$", "Lpus$i$Exacte", "Lpus1Exacteg1",fdEvals);
);

Using the Lpusi Variables

- Example 1

// set cs1=8.0

```
dvs.get(1).setValue(8.0);
```

// get value of Lpus1

```
rvs.get(1).getValue();
```

  - Behavior:
    - The statement set cs1=15.0 causes cs1 and cs21 (remember they are linked) to be changed in the boundary condition file.
    - rvs.get(1).getValue(); Will cause the Evaluator to “evaluate” the value of Lpus1 since some of it’s dependencies (in this case cs1, cs21 have changed.
    - The Evaluator will execute the Avustask to compute a new Lpus1 value.

- Example 2

// set cs2=5.0

```
dvs.get(1).setValue(5.0)
```

// get value of Lpus1

```
rvs.get(1).getValue();
```

// get value of Lpus2

```
rvs.get(2).getValue();
```

  - Behavior:
    - The statement set =15.0 causes cs2 and cs22 (remember they are linked) to be changed
    - rvs.get(1).getValue(); Will cause the Evaluator to “evaluate” the value for Lpus1The Evaluator will execute the Avus task to compute a new Lpus1.
    - The statement rvs.get(2).getValue( ) retunes the value for Lpus2. It does not cause the Evaluator to “evaluate”. Remember all the response variables have the same Evaluator (just different Filters) and none of this Evaluator’s dependencies have changed since the rvs.get(1).getValue() command was issued.

Need example on Sensitivity calculations

Configuration of LpusiSOA Evaluation

Lpus$i$SOAe – *Evaluator*

$$Lpus_i(\bar{x}) = Lpus_i(\bar{x}^0) + \nabla Lpus_i(\bar{x}^0)\{\bar{x} - \bar{x}^0\}$$

Lpus$i$SOAeg1 – *GradientEvaluator*

Analytic

$$\frac{\partial \bar{L}pus_{iSOAe}(\bar{x})}{\partial x_i} = \text{constant}_i$$

Lpus%i%SOAe – *Evaluator*

// Construct Evaluator for LPUSISOA

// this is a method evaluator $Lpus_i(\bar{x}^0)$ $\bar{x}^0$ $\nabla Lpus_i(\bar{x}^0)$

SOA soaLpus = new SOA((expansionPntLpsu%i%, expansioPntXbar, expansionPntGradients);

// Create the Method Evaluator (Note: two options – 20 Method Evaluators each returning one Lpusi value, or 1 Method Evaluator and 20 ListFilters. The MehtodEvaluator returns a list of Lpusis

MethodEvaluator lpus1SOAe = new MethodEvaluator(”Lpus1SOA”,soaLpus1,”evaluate”) ;

MethodEvaluator lpusSOAe = new MethodEvaluator(”LpusSOA”,soaLpus,”evaluate”) ;

// Set the arguments for the Method

LpusSOAe.setArgs(vars[ ]);

$$\bar{x} = \{\beta_i, \alpha\}^T$$

Lpus%i%SOAe – *Evaluator*

/ create **the filter for each Lpus%i%**

```
Filter rvFilter1= new Filter(new ListFilter(1));
…
Filter rvFilter20= new Filter(new ListFilter(20));
```

var(model, loop(1:20),loop(1:20),“lpus%i%”,lpus%i%SOAe,rvFilter%i%,”beta%k%”,”alpha”);

Need k to loop 1-20 for each i

LpusiSOAeg1 – *GradientEvaluator*

Analytic

$$\frac{\partial \bar{L}pus_{iSOAe}(\bar{x})}{\partial x_i} = \frac{\partial \bar{L}pus_{iSOAe}(\bar{x}^0)}{\partial x_i} = \text{constant}_i$$

$$\frac{\partial Lpus_{kSOAe}(\bar{x}^0)}{\partial \beta_i}$$

Evaluator dLpus%k%SOAedbeta%i%e = *evaluator("*dLpus%k%SOAedbeta%i%e *", " dLpus%k%dbeta%i%atxzero");*

Evaluator dLpus%k%SOAedalphae = *evaluator("*dLpus%k%SOAedalphae *", " dLpus%k%dalphaatxzero");*

$$\frac{\partial Lpus_{kSOAe}(\bar{x}^0)}{\partial \alpha}$$

List<Evaluator> Lpus%k%g1= *list(*dLpus%k%SOAedbeta%i% , dLpus%k%SOAedalpha *);*

model.setGradientEvaluators("Lpus%k%", "Lpus%k%SOAe", "lpus%k%g1", Lpus%k%g1);

q Configuration

qExact – *Evaluator*

Analytic Expression

$$q(M,\gamma,p_s) = \frac{1}{2}\gamma p_s M^2$$

qExactg1 – Gradient*Evaluator*

Analytic Expressions

$$q(M,\gamma,p_s) = \frac{1}{2}\gamma p_s M^2$$

$$\frac{\partial q}{\partial M} = \gamma p_s M, \quad \frac{\partial q}{\partial \gamma} = \frac{1}{2} p_s M^2, \quad \frac{\partial q}{\partial p_s} = \frac{1}{2}\gamma M^2$$

qExact – *Evaluator*

*var(model, “q", ”qExact", evaluator(”qExact”, ”0.5*gamma*pstatic*mach^2"), dependent (“mach”,”pstatic”,”gamma”));*

qExactg1 – Gradient*Evaluator*

Evaluator dqExactdmachg1 = *evaluator(”* dqExactdmachg1*", ”gamma*pstatic*mach", args(“mach”,”pstatic”,”gamma”));*
Evaluator dqExactdgammag1 = *evaluator(”* dqExactdgammag1*", ”0.5*pstatic*mach^2", args(“mach”,”pstatic”));*
Evaluator dqExactdpstaticg1 = *evaluator(”* dqExactdpstaticg1*", ”0.5*gamma*mach^2", args(“mach”,”gamma”));*

*Differentiation(model, “q", ”qExact", ”qExactg1", evaluators(*dqExactdmachg1 , dqExactdgammag1 , dqExactdpstaticg1*));*

Configuration of DI

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

DI has 4 different Evaluations - DIExacte, DISOAe, DIMOAe, DIKrige
Each Evaluation has a single way of computing the gradients

- DIExacte – DiExacteg1
- DISOAe – DISOAeg1
- DIMOAe – DIMOAeg1
- DIKrige – DIKrigeg1

Configuration of DI

Evaluation - DIExacte

$$D_I(Lpus_i(\alpha,\beta_i,\beta l_i),qdp)$$

A separate Class called InducedDrag.java has been created to compute DI. This class has a method called *evaluateInducedDrag* which takes as an argument a list of Variables. These Variables are expected to be the Lpus_i response Variables and q. The method Computes

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

For this a MethodEvaluator will be used.

```
// Construct the InducedDrag Object
InducedDrag idragObj = new InducedDrag("avusIdrag", yiA);
// Create the Method Evaluator
MethodEvaluator iDragMethodEval = new MethodEvaluator(“DIExacte”,idragObj, "evaluateIDrag");
// Set the arguments for the Method
iDragMethodEval.setArgs(lpusVars[ ],”evaluatorName”,qdp);
```

Note: One could also do the above using a Groovy Evaluator

Evaluation - DIExacte

```
// set the iDrag evaluator dependencies
iDragMethodEval.addDependents(lpusVars[ ],qdp);
// add the Evaluator to the iDrag response Variable
rvs.getVariable("DI").setEvaluator(iDragMethodEval);
```

Note: Since the method returns the value DI, there is no need for a filter.

```
var(model, "DI", "DIExacte", iDragMethodEval,args (names(Loop(20),"Lpus$i$), "q"))
```

Using the iDrag variable

Expected behavior:
// set beta1=15.0
dvs.get(1).setValue(15.0)
// get value of
rvs.getVariable("iDrag").getValue();

- The statement set beta1=15.0 causes beta1 and beta21 (remember they are linked) to be changed in the boundary condition file.

- The statement rvs.getVariable("iDrag").getValue(); Will cause the Evaluator to "evaluate" the value for DI.

- Since some of it's implicit dependencies (beta1 in this case) have changed. It causes the Lpus_i Evaluator (avusEvaluator) to "evaluate" since it is its explicit dependency, beta1 has changed.

- Once this completes, the idrag-MethodEvaluator evaluates with the updated set of Lpusi's , its explicit dependencies, and produce a new value for induced drag.

Sensitivities for DIExacte

GradientEvaluator – DIExacteg1

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

$$\frac{\partial D_I}{\partial q} = \frac{1}{8\pi q^2}\sum_{i=1}^{n} Lpus_i \Delta y_i \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_i - y_k} - \frac{1}{y_i + y_k}\right]$$

$$\frac{\partial D_I}{\partial Lpus_{j=1,\text{ or } j=n}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

$$\frac{\partial D_I}{\partial Lpus_{j=2,\text{ to } n-1}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] + \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_{j-1}} - \frac{1}{y_i + y_{j-1}}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

GradientEvaluator – DIExacteg1

$$\frac{\partial D_I}{\partial Lpus_{j=1,j=n}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

$$\frac{\partial D_I}{\partial Lpus_{j=2,n-1}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] + \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_{j-1}} - \frac{1}{y_i + y_{j-1}}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

A separate Class called InducedDrag.java has been created to compute DI and the above partial derivatives. This class has a method called *evaluateInducedDragSensitivity* which takes as arguments Lpusi Variables, q Variable and the Variable to compute the sensitivity wrt. These Variables are expected to be the Lpus_i response Variables and q.

For this a MethodEvaluator will be used.

```
// Construct the InducedDrag Object
InducedDrag idragObj = new InducedDrag("avusIdrag", yiA);
// Create the Method Evaluator
MethodEvaluator dDIExactedLpusie = new MethodEvaluator(“dDIExactedLpusie ”,idragObj,
"evaluateIDragSensitivity");
// Set the arguments for the Method
dDIExactdLpusi.setArgs(LpusiVars, qVar, varwrt);
```

GradientEvaluator – DIExacteg1

// add the dependencies to the evaluator
dDIExactedLpusie.addDependents(lpusVars[ ],qdp);
List<Evaluator> DIExacteg1 = *list(dDIExactdLpusie, dDIExactdLpusie, ….21 times);*
sm.setGradientEvaluators(“DI", ”DIExacte", ”DIExacteg1", DIExacteg1);

DISOAe Evaluator

Evaluation - DISOAe

$$\overline{D}_I(\bar{x}) = D_I(\bar{x}^0) + \nabla \overline{D}_I(\bar{x}^0)\{\bar{x} - \bar{x}^0\}$$

Gradient Evaluation – DISOAeg1

$$\frac{\partial \overline{D}_I(\bar{x})}{\partial x_i} = \frac{\partial \overline{D}_I(\bar{x}^0)}{\partial x_i} = \text{constant}_i$$

// Construct Evaluator for DISOAe

// this is a method evaluator

SOA DISOA = new SOA((DIxBar0, xBar0, gradsDixBar0);

// Create the Method Evaluator

Evaluator DISOAe = new MethodEvaluator(”DISOA”,DISOA,”evaluate”) ;

// Set the arguments for the Method

DISOAe.setArgs(Lpusi[], q);

Var(Model, “DI”,”DISOAe”,DISOAe, args(Loop(1:20), “Lpus%i%”, “q”);

// Construct the GradientEvaluator for DISOAeg1, use a simple expression evaluator

Evaluator dDISOAedbeta1 = *expression(”dDISOAedbeta1", gradsDixBar0.getValue(“beta1”);*

Evaluator dDISOAedbeta2 = *expression(”dDISOAedbeta2", gradsDixBar0.getValue(“beta2”);*

….

Evaluator dDISOAedbeta20 = *expression(”dDISOAedbeta20", gradsDixBar0.getValue(“beta20”);*

Evaluator dDISOAedalpha = *expression(”dDISOAedbeta20", gradsDixBar0.getValue(“beta20”);*

Gradient Evaluation – DISOAeg1

List<Evaluator> DISOAeg1 = *list(*dDISOAedbeta1*,* dDISOAedbeta2*,*
dDISOAedbeta3*, …* dDISOAedbeta20, dDISOAedalpha*);*

sm.setGradientEvaluators(“DI", ”DISOAe", "DISOAeg1 ", DISOAeg1);

DIMOAe Evaluator

DIMOAe – *Evaluator*

$$D_I = \frac{-1}{8\pi q}\sum_{i=1}^{n} \tilde{L}pus_i \Delta y_i \sum_{k=1}^{n}\left[\tilde{L}pus_k - \tilde{L}pus_{k+1}\right]\left[\frac{1}{y_i - y_k} - \frac{1}{y_i + y_k}\right]$$

DIMOAeg1 – *GradientEvaluator*

$$\frac{\partial D_I}{\partial \tilde{L}pus_{j-1,j-n}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[\tilde{L}pus_k - \tilde{L}pus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n}\tilde{L}pus_i\Delta y_i\left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

$$\frac{\partial D_I}{\partial \tilde{L}pus_{j-2,n-1}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[\tilde{L}pus_k - \tilde{L}pus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] + \frac{1}{8\pi q}\sum_{i=1}^{n}\tilde{L}pus_i\Delta y_i\left[\frac{1}{y_i - y_{j-1}} - \frac{1}{y_i + y_{j-1}}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n}\tilde{L}pus_i\Delta y_i\left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

// The only difference between DIMOAe and DIExacte is that Lpusi is replaced by a standard first order approximation(SOA) for Lpusi. This is indicated by the tilda above the L

// The Lpusi Variables with the SOA evaluator can be created here as it was done in the previous slides

/ Slides are duplicated here for completness

Configuration of LpusiSOA Evaluation for DIMOAe

DIMOAe – *Evaluator*

Lpus%i%SOAe – *Evaluator*

$$Lpus_i(\bar{x}) = Lpus_i(\bar{x}^0) + \nabla Lpus_i(\bar{x}^0)\{\bar{x} - \bar{x}^0\}$$

Lpus%i%SOAe – *Evaluator*

// Construct Evaluator for LPUSISOA

// this is a method evaluator $Lpus_i(\bar{x}^0)$ $\bar{x}^0$ $\nabla Lpus_i(\bar{x}^0)$

SOA soaLpus = new SOA((expansionPntLpsu%i%, expansioPntXbar, expansionPntGradients);

// Create the Method Evaluator (Note: two options – 20 Method Evaluators each returning one Lpusi value, or 1 Method Evaluator and 20 ListFilters. The MehtodEvaluator returns a list of Lpusis

MethodEvaluator lpus1SOAe = new MethodEvaluator(”Lpus1SOA”,soaLpus1,”evaluate”) ;

MethodEvaluator lpusSOAe = new MethodEvaluator(”LpusSOA”,soaLpus,”evaluate”) ;

// Set the arguments for the Method

LpusSOAe.setArgs(vars[ ]);

$$\bar{x} = \{\beta_i, \alpha\}^T$$

DIMOAe – *Evaluator*

Lpus%i%SOAe – *Evaluator*

/ create **the filter for each Lpus%i%**

```
Filter rvFilter1= new Filter(new ListFilter(1));
…
Filter rvFilter20= new Filter(new ListFilter(20));
```

Variable[ ] LpusiVars = var(model, loop(1:20),loop(1:20),“lpus%i%”,lpus%i%SOAe,rvFilter%i%,”beta%k%”,”alpha”);

Need k to loop 1-20 for each i

DIMOAe – *Evaluator*

```
/ Create the Method Evaluator for DIMOAe using LpusiVars created previously
// Construct the InducedDrag Object
InducedDrag idragObj = new InducedDrag("avusIdrag", yiA);
// Create the Method Evaluator
MethodEvaluator iDragMethodEval = new MethodEvaluator(“DIMOAe”,idragObj,
"evaluateIDrag");
// make sure the evaluator for LpusiVars is set to “LpusiSOAe”
  setEvaluator(LpusiVars, Loop(1:20) “Lpus%i%SOAe”)
// Set the arguments for the Method
iDragMethodEval.setArgs(lpusVarsSOA[ ],qdp);
```

GradientEvaluator – DIMOAeg1

$$\frac{\partial D_I}{\partial Lpus_{j=1,j=n}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

$$\frac{\partial D_I}{\partial Lpus_{j=2,n-1}} = \frac{-1}{8\pi q}\Delta y_j \sum_{k=1}^{n}\left[Lpus_k - Lpus_{k+1}\right]\left[\frac{1}{y_j - y_k} - \frac{1}{y_j + y_k}\right] + \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_{j-1}} - \frac{1}{y_i + y_{j-1}}\right] - \frac{1}{8\pi q}\sum_{i=1}^{n} Lpus_i \Delta y_i \left[\frac{1}{y_i - y_j} - \frac{1}{y_i + y_j}\right]$$

Computed the same way as for DIExacteg1 except use LpusiSOA instead of LpusiExact

A separate Class called InducedDrag.java has been created to compute DI and the above partial derivatives. This class has a method called *evaluateInducedDragSensitivity* which takes as arguments Lpusi Variables, q Variable and the Variable to compute the sensitivity wrt. These Variables are expected to be the Lpus_i response Variables and q.

For this a MethodEvaluator will be used.

```
// Construct the InducedDrag Object
InducedDrag idragObj = new InducedDrag("avusIdrag", yiA);
// Create the Method Evaluator
MethodEvaluator dDIMOAedLpusie = new MethodEvaluator(“dDIMOAedLpusie ”,idragObj,
"evaluateIDragSensitivity");
// Set the arguments for the Method
dDIMOAedLpusi.setArgs(LpusiVarsSOA, qVar, varwrt);
```

GradientEvaluator – DIMOAeg1

// add the dependencies to the evaluator
dDIMOAedLpusie.addDependents(lpusVarsSOA[ ],qdp);
List<Evaluator> DIMOAeg1 = *list(dDIMOAedLpusie, dDIMOAedLpusie, ….21 times);*
sm.setGradientEvaluators(“DI", ”DIMOAe", ”DIMOAeg1", DIMOAeg1);

DlKrige Evaluator

TBD

- **LT is configured in a similar fashion as DI**

As can be seen in SORCER Models consists of three components, Variables, Evaluators, and Filters. The purpose of this example is to take the engineering functional realationship defined in and create a Model in SORCER. Currently SORCER has three types of models, ResponseModel, ParametricModel, and SensitivityModel. In the near future an OptimizationModel will be incorporated as well. Here we will show the development of each of the three available models. First, the development of a ResponseModel will be demonstrated. This will require the definition of the Model, the Variables, and then the configuration of the Variables with the appropriate Filters and Evaluators.

For the case depicted in there are a total of 21 design variables, 20 design variables associated with and one design variable associated with . The remaining control surface varibles will be “linked” (a set relationship) to . The response variables consists of the 20 lift per unit span values, , and the induced drag, . The primary classes that are used to develop the model can be found in the engineering.avus.requestor and engineering.avus.provider packages. The class names are AvusModeler.java and AvusModelConfigurator.java, InducedDrag.java, and AvusOutput.java.

Below are the significant statements and steps required to build the model.

Define the Model and Variables - This can be done in one step with the following functional composition syntax.

```
ResponseModel avusRM = responseModel("Induced Drag Response Analysis",
designVars("cs",20),
designVars("alpha"),
linkedVars("cs", 20,21),
responseVars("lpus", 20),
responseVars("iDrag"));
```

This is a function with embeded statements. An instance of ResponseModel is created called avusRM, the name of the model is “Induced Drag Response Analysis”. This name is arbitrary and user specified. The next argument in the constructor is
DesignVars("cs",20) - Returns a list of 20 Variable objects with the *type attribute set to “DESIGN” and names “cs1”, “cs2”, ... “cs20”.*
DesignVars("alpha") - Returns a single Variable object with *type attribute of “DESIGN” and name “alpha”*
LinkedVars("cs", 20,21) - Returns a list of 20 Variable objects with *type attribute “DESIGN” and kind attribute of “LINKED” with names “cs21”, “cs22”, ...”cs40”*
ResponseVars("lpus", 20)- Returns a list of 20 Variable objects with *type attribute of “RESPONSE” with names “lpus1”, “lpus2”, ...”lpus20”*
ResponseVars("iDrag")Returns a Variable object with *type attribute of “RESPONSE” with name “iDrag”*
A formal definition of the Variable class and some of its available attributes can be found in Appendix A.

2. The next step required is to *configure the DesignVariables, LinkedVariables, and the ResposneVariables.*

a. configure the Design Variables - Each indepenedent design variable consists of zero, one, or more Filters. For this case the independent design variables reside in two locations. The variables reside in an ascii text file(AVUS_Boundary_Condition.dat) and resides in a separate ascii text file (AVUS_Job.job). In order to have the ability to get and set the values in these files from anywhere on the network (we have no idea where avus may run) we need to create Filter objects and place them into the Variables. Filters take a larger entity and “filter” out a portion of that entity for either reading, writing or passing to another filter. Since we are working with ascii text files in this case we will create BasicFileFilter objects (a list of the all currently available Filter types can be found in the sorcer.vfe.filter package) . Each BasicFileFilter requires a Pattern object which is used to locate the desired quantity in the entity. Here our pattern is essentially *line, field, delimeter information. Other patterns such as regular expressions are available as well. Below is an excerpt from the AVUS_Boundary_Condition.dat file and the AVUS_Job.job file(the line numbers are not actually in the file. They are shown here to indicate where in the file the excerpt has been taken from).*

Appendix B (Avus Task)

```
/ construct the ProviderContext
        ServiceContext avusContext = getAvusContext(); // Method to get Avus Context
// construct Method and then Task for the Avus job
        Signature avusMethod = new ServiceSignature("executeAvus",
engineering.provider.avus.AvusRemoteInterface.class, “AVUS-Engineering-DNA”);
        ServiceTask avusTask = new ServiceTask("run Avus", avusMethod);
// add the context to the Task
        avusTask.setContext(avusContext);
```

Avus Context

```
private ServiceContext getAvusContext() throws Exception {
// get the context nodes
        ContextNode[] avusNodes = getAvusContextNodes();
        // construct the context
        ServiceContext context = new ServiceContext("AvusContext","AvusContext");
        context.putValue( IN_VALUE + CPS + "AVUS/BCFILE",avusNodes[1],AVUS_BC_TXT);
        context.putValue( IN_VALUE + CPS + "AVUS/JOBFILE", avusNodes[2], avusJob);
        context.putValue( IN_VALUE + CPS + "AVUS/RESTARTFILE", avusNodes[3],avusJob);
        context.putValue( IN_VALUE + CPS + "AVUS/GRIDFILE",avusNodes[4], avusGrid);
        context.putValue( IN_VALUE + CPS + "AVUS/OLDRESTARTFILE",avusNodes[5], avusOldRestart);
        context.putValue( IN_VALUE + CPS + "AVUS/TAPFILE",avusNodes[6], avusTap);
        context.putValue( IN_VALUE + CPS+ "AVUS/TRIPFILE", avusNodes[7], avusTrip);

        // create and add the AvusOutput object to context
        AvusOutput avusOut;
        File avusOutputFile = new File(properties.getProperty("requestor.avusbaselineoutput"));
        avusOut = new AvusOutput(avusOutputFile);
        context.putValue( OUT_VALUE + CPS+ "AVUS/AVUSOUTPUT",avusOut, avusOutput);
        return context;
        }
```

Avus Context Nodes

```
private ContextNode[] getAvusContextNodes() throws Exception {
                ContextNode fCN1 = null; //  bcinputfile
                ContextNode fCN2 = null; // bcinputfil gen2 format
                ContextNode fCN3 = null; // jobinputfile
                ContextNode fCN4 = null; // jobrestartfile
                ContextNode fCN5 = null; // gridinputfile
                ContextNode fCN6 = null; // oldrestartinputfile
                ContextNode fCN7 = null; // tapinputfile
                ContextNode fCN8 = null; // tripinputfile

                // Create context nodes associated with Files needed to run AVUS
                String basedataURL = Sorcer.getDataServerUrl();
                String avusPath = properties.getProperty("requestor.avus.datapath");
                String dataURL = basedataURL + "/" + avusPath;
                out.println("dataURL = " + dataURL);

                // File Context Node for BCinputFile
                String bcInputFileName = properties.getProperty("requestor.bcinfile");
                File bcInputFile = new File(bcInputFileName);
                fCN1 = createURLNode("AvusBCData", bcInputFile, dataURL);

                // File Context Node for JobinputFile
                String jobInputFileName = properties.getProperty("requestor.jobinfile");
                File jobInputFile = new File(jobInputFileName);
                fCN3 = createURLNode("AvusJobData", jobInputFile, dataURL);

                String jobRestartFileName = properties.getProperty("requestor.jobrestartfile");
                File jobRestartFile = new File(jobRestartFileName);
                fCN4 = createURLNode("AvusRestartJobData", jobRestartFile,dataURL);

                // File Context Node for gridinputFile
                String gridInputFileName = properties.getProperty("requestor.gridinfile");
                File gridInputFile = new File(gridInputFileName);
                fCN5 = createURLNode("AvusGridData", gridInputFile, dataURL);
```

```
                // File Context Node for oldrestartinputFile
                String oldrestartInputFileName = properties.getProperty("requestor.oldrestartinfile");
                File oldrestartInputFile = new File(oldrestartInputFileName);
                fCN6 = createURLNode("AvusOldRestartData", oldrestartInputFile,dataURL);

                // File Context Node for tapinputFile
                String tapInputFileName = properties.getProperty("requestor.tapinfile");
                File tapInputFile = new File(tapInputFileName);
                fCN7 = createURLNode("AvusTapData", tapInputFile, dataURL);

                // File Context Node for tripinputFile
                String tripInputFileName = properties.getProperty("requestor.tripinfile");
                File tripInputFile = new File(tripInputFileName);
                fCN8 = createURLNode("AvusTripData", tripInputFile, dataURL);

                return new ContextNode[] { fCN1, fCN2, fCN3, fCN4, fCN5, fCN6, fCN7,fCN8};
}

public static ContextNode createURLNode(String nodeName, File fileName, String base) {
                URL dataURL = null;
                try {
                        dataURL = new URL(base + "/" + fileName);
                } catch (MalformedURLException e) {
                        out.println("Cannot create URLNode with filename = "+ fileName + " base = " + base);
                        e.printStackTrace();
                }
                return new ContextNode(nodeName, dataURL);
```