SAND2007-7231C

Using Thyra and Stratimikos to Build Blocked and Implicitly Composed Solver Capabilities

Roscoe A. Bartlett
Department of Optimization & Uncertainty Estimation

Sandia National Laboratories

Trilinos Users Group Meeting, November 6th, 2007

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under contract DE-AC04-94AL85000.

Sandia National Laboratories

Motivation for Blocked and Implicitly Composed Vectors and Linear Operators

Example: An Attempt at a Physics-Based Preconditioner

Second-order (p2) Lagrange Operator: P_2

First-order (p1) Lagrange Operator: P_1

Idea for a preconditioner? $\bar{P}_2 = M_{22}^{-1} M_{12} P_1^{-1} M_{11}^{-1} M_{21}$

Preconditioned Linear System

$$P_2 \bar{P}_2 (\bar{P}_2^{-1} x) = b$$

- Preconditioner involves nested linear solves
- Major Problem: This is a singular preconditioner!
  - That's besides the point, since we did not see this right away but the numerical solve sure told us this!
- The Main Point: We want to be able to quickly try out interesting alternatives!

Sandia National Laboratories

Example: Implicit RK Method from Rythmos

Implicit ODE/DAE: $f(\dot{x}, x, t) = 0,\ t \in [t_0, t_f],\ \ x(t_0) = x_0,\ \ \dot{x}(t_0) = \dot{x}_0$

Fully Implicit RK Time Step Equations: Solve $\bar{f}(\bar{x}) = 0$ to advance from t_k to t_{k+1}

Collocation eqns: $$\bar{f}_i(\bar{x}) = f\left(\dot{x}_i, x_k + \Delta t \sum_{j=0}^{p-1} a_{ij}\dot{x}_j, t_k + c_i \Delta t\right) = 0,\ \text{for } i = 0 \ldots p-1$$

Stage derivatives: $$\bar{x}^T = \left[\ \dot{x}_0 \ \ \dot{x}_1 \ \ \cdots \ \ \dot{x}_{p-1}\ \right]^T$$

Butcher Tableau:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cccc} c_0 & a_{0,0} & a_{0,1} & \cdots & a_{0,p-1} \\ c_1 & a_{1,0} & a_{1,1} & \cdots & a_{1,p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{p-1} & a_{p-1,0} & a_{p-1,1} & \cdots & a_{p-1,p-1} \\ \hline & b_0 & b_1 & \cdots & b_{p-1} \end{array}$$

Newton System for RK Time Step Equations: $\Delta \bar{x} = -(\bar{W}_k)^{-1}\bar{f}(\bar{x}_k)$

Block Structure: $$\bar{f} = \begin{bmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{p-1} \end{bmatrix} \quad \bar{x} = \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_{p-1} \end{bmatrix} \quad \frac{\partial \bar{f}}{\partial \bar{x}} = \bar{W} = \begin{bmatrix} \bar{W}_{0,0} & \bar{W}_{0,1} & \cdots & \bar{W}_{0,p-1} \\ \bar{W}_{1,0} & \bar{W}_{1,1} & \cdots & \bar{W}_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{W}_{p-1,0} & \bar{W}_{p-1,1} & \cdots & \bar{W}_{p-1,p-1} \end{bmatrix}$$

$$\bar{W}_{i,j} = \frac{\partial \bar{f}_i}{\partial \bar{x}_j} = \frac{\partial f}{\partial \dot{x}} + \Delta t\, a_{i,j} \frac{\partial f}{\partial x},\ \text{for } i = 0 \ldots p-1,\ j = 0 \ldots p-1$$

Sandia National Laboratories

Example: Multi-Period Optimization Problem (MOOCHO)

Multi-Period Optimization Problem:

Minimize: $\sum_{i=0}^{N-1} \beta_i g_i(x_i, p)$

Subject to: $f(x_i, p, q_i) = 0, \text{ for } i = 0 \ldots N-1$

where: x_i : State variables for period i
p : Optimization parameters
q_i : Input parameters for period i

Use Cases:

- Parameter estimation using multiple data points
- Robust optimization under uncertainty
- Design under multiple operating conditions
- …

Abstract Form of Optimization Problem:

Minimize: $\bar{g}(\bar{x}, p)$

Subject to: $\bar{f}(\bar{x}, p) = 0$

where:

$$\bar{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{p-1} \end{bmatrix} \qquad \bar{g}(\bar{x}, p) = \sum_{i=0}^{N-1} \beta_i g_i(x_i, p) \qquad \frac{\partial \bar{f}}{\partial \bar{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_0} & & & \\ & \frac{\partial f}{\partial x_1} & & \\ & & \ddots & \\ & & & \frac{\partial f}{\partial x_{N-1}} \end{bmatrix}$$

$$\bar{f} = \begin{bmatrix} f(x_0, p, q_0) \\ f(x_1, p, p_1) \\ \vdots \\ f(x_{N-1}, p, q_{N-1}) \end{bmatrix}$$

Goals for our Numerical Software

- We want our numerical software to be general for all situations
  - Serial, or MPI, or any other configuration …
  - Medium-scale and large-scale problems …
  - Flat structure, or block structure, or whatever structure …
  - etc …
- We want to be able to build these solvers quickly
- We want our numerical algorithm software to be fast
- We don't want to get bogged down in MPI calls
  - We don't even want to have to think about parallelism in many cases!
- We want to user to be able to specialize almost any part of our algorithm without directly modifying source code
  - i.e. the Open Closed Principle (OCP) of OO design [Martin, 2003]

How can we do this?

Abstract Numerical Algorithms with Thyra!

Sandia National Laboratories

Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Categories of Abstract Problems and Abstract Algorithms

Trilinos Packages

- **Linear Problems:** Given linear operator (matrix) $A \in \mathbf{R}^{n \times n}$
  - Linear equations: Solve $Ax = b$ for $x \in \mathbf{R}^n$ — Belos
  - Eigen problems: Solve $Av = \lambda v$ for (all) $v \in \mathbf{R}^n$ and $\lambda \in \mathbf{R}$ — Anasazi
  - Preconditioners: Generate specialized preconditioner P for A — Meros
- **Nonlinear Problems:** Given nonlinear operator $f(x,p) \in \mathbf{R}^{n+m} \rightarrow \mathbf{R}^n$
  - Nonlinear equations: Solve $f(x) = 0$ for $x \in \mathbf{R}^n$ — NOX
  - Stability analysis: For $f(x,p) = 0$ find space $p \in \mathcal{P}$ such that $\frac{\partial f}{\partial x}$ is singular — LOCA
- **Transient Nonlinear Problems:**
  - DAEs/ODEs: Solve $f(\dot{x}(t), x(t), t) = 0, t \in [0,T]$, $x(0) = x_0$, $\dot{x}(0) = x'_0$ for $x(t) \in \mathbf{R}^n, t \in [0,T]$ — Rythmos
- **Optimization Problems:**
  - Unconstrained: Find $p \in \mathbf{R}^m$ that minimizes $g(p)$ — MOOCHO
  - Constrained: Find $x \in \mathbf{R}^n$ and $p \in \mathbf{R}^m$ that:
    minimizes $g(x,p)$
    such that $f(x,p) = 0$ — Aristos

Introducing Abstract Numerical Algorithms

What is an abstract numerical algorithm (ANA)?

An ANA is a numerical algorithm that can be expressed abstractly solely in terms of vectors, vector spaces, linear operators, and other abstractions built on top of these without general direct data access or any general assumptions about data locality

Key Points

- ANAs can be very mathematically sophisticated!
- ANAs can be extremely reusable!
- Flexibility needed to achieve high performance!

Example : Linear Conjugate Gradients

Given:

$A \in \mathcal{X} \rightarrow \mathcal{X}$: s.p.d. linear operator

$b \in \mathcal{X}$: right hand side vector

Find vector $x \in \mathcal{X}$ that solves $Ax = b$

Linear Conjugate Gradient Algorithm

Compute $r^{(0)} = b - \boxed{Ax^{(0)}}$ for the initial guess $x^{(0)}$.

for $i = 1, 2, \ldots$

$\rho_{i-1} = \langle r^{(i-1)}, r^{(i-1)} \rangle$

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ $(\beta_0 = 0)$

$p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}$ $(p^{(1)} = r^{(1)})$

$q^{(i)} = \boxed{Ap^{(i)}}$

$\gamma_i = \langle p^{(i)}, q^{(i)} \rangle$

$\alpha_i = \rho_{i-1} / \gamma_i$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence; continue if necessary

end

Types of operations

linear operator applications

vector-vector operations

Scalar operations

scalar product <x,y> defined by vector space

Types of objects

Linear Operators

- A

Vectors

- r, x, p, q

Scalars

- ρ, β, γ, α

Vector spaces?

- $\mathcal{X}$

Sandia National Laboratories

Example, Linear CG Coded Using Thyra Handle Layer

Math Notation for CG

Compute $r^{(0)} = b - Ax^{(0)}$ for the initial guess $x^{(0)}$.

for $i = 1, 2, \ldots$

$$\rho_{i-1} = \langle r^{(i-1)}, r^{(i-1)} \rangle$$
$$\beta_{i-1} = \rho_{i-1}/\rho_{i-2} \ (\beta_0 = 0)$$
$$p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)} \ (p^{(1)} = r^{(1)})$$
$$q^{(i)} = Ap^{(i)}$$
$$\gamma_i = \langle p^{(i)}, q^{(i)} \rangle$$
$$\alpha_i = \rho_{i-1}/\gamma_i$$
$$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$$
$$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$$

check convergence; continue if necessary

end

C++ Implementation Using Thyra (Handles)

```
// Initialization …
Vector<Scalar> r = b - A*x;
for( int iter = 0; iter <= maxNumIters; ++iter ) {
  rho  = inner(r,r);
  beta = (iter!=0 ? rho/rho_old : one);
  if(iter!=0) p = r + beta*p; else p() = r;
  q = A*p;
  gamma = inner(p,q);
  alpha = rho/gamma;
  x += alpha*p;
  r -= alpha*q;
  // Check convergence ...
  rho_old = rho;
}
```

- Works with any linear operator and vector implementation (e.g. Epetra, PETSc, etc.)
- Works in any computing configuration (i.e. serial, SPMD, client/server etc.)
- Works with any Scalar type (i.e. float, double, complex<double>, extended precision, etc.) that has a traits class
- Allows algorithm developers to code ANAs without (almost) any knowledge of parallel issues

See silliestCgSolve(…) for the real code …

Trilinos Strategic Goals

- **Scalable Computations:** As problem size and processor counts increase, the cost of the computation will remain nearly fixed.
- **Hardened Computations:** Never fail unless problem essentially intractable, in which case we diagnose and inform the user why the problem fails and provide a reliable measure of error.
- **Full Vertical Coverage:** Provide leading edge enabling technologies through the entire technical application software stack: from problem construction, solution, analysis and optimization.

Algorithmic Goals

- ***Grand* Universal Interoperability**: All Trilinos packages will be interoperable, so that any combination of solver packages that makes sense algorithmically will be possible within Trilinos.

Thyra is being developed to address this issue

- **Universal Accessibility:** All Trilinos capabilities will be available to users of major computing environments: C++, Fortran, Python and the Web, and from the desktop to the latest scalable systems.
- **Universal Solver RAS**: Trilinos will be:
  - Reliable: Leading edge hardened, scalable solutions for each of these applications
  - Available: Integrated into every major application at Sandia
  - Serviceable: Easy to maintain and upgrade within the application environment.

Software Goals

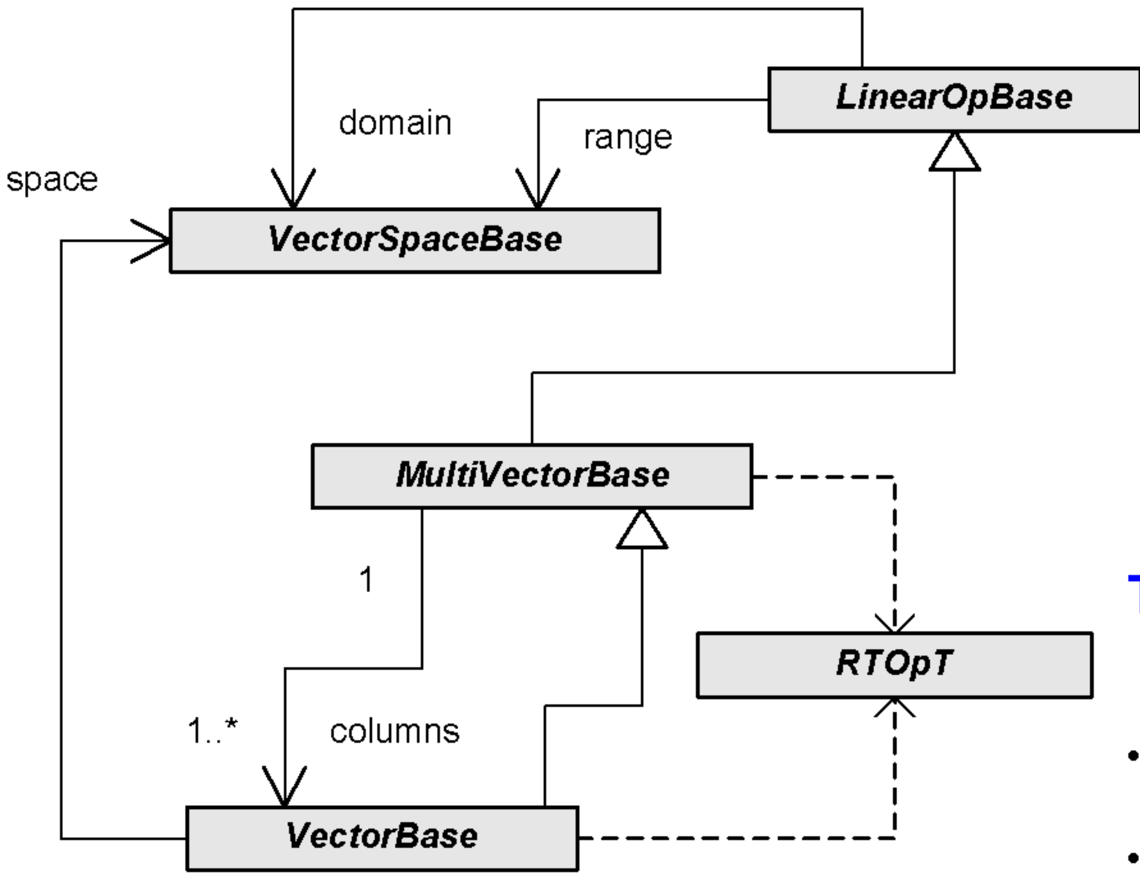
Courtesy of Mike Heroux, Trilinos Project Leader

Sandia National Laboratories

Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Fundamental Thyra ANA Operator/Vector Interfaces



A Few Quick Facts about Thyra Interfaces

- All interfaces are expressed as abstract C++ base classes (i.e. object-oriented)
- All interfaces are templated on a Scalar data (i.e. generic)

The Key to success!

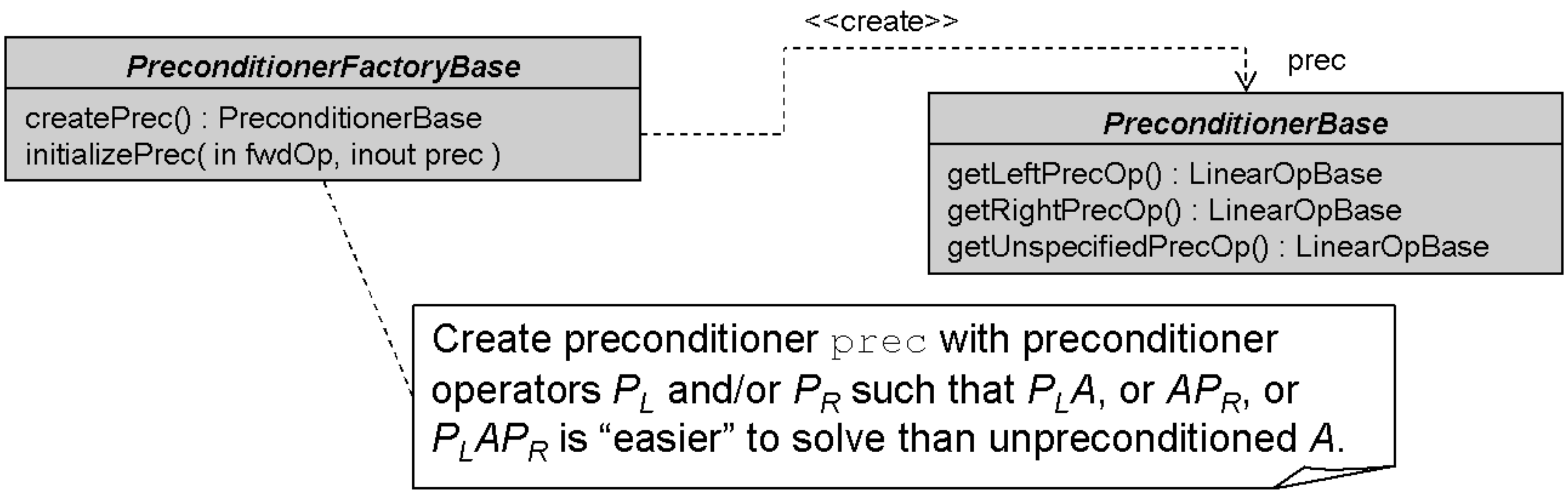
Reduction/Transformation Operators

- Supports all needed element-wise vector operations
- Data/parallel independence
- Optimal performance

R. A. Bartlett, B. G. van Bloemen Waanders and M. A. Heroux. *Vector Reduction/Transformation Operators*, ACM TOMS, March 2004

Preconditioners and Preconditioner Factories

PreconditionerFactoryBase : Creates and initializes **PrecondtionerBase** objects



- Allows unlimited creation/reuse of preconditioner objects
- Supports reuse of factorization structures
- Adapters currently available for Ifpack and ML
- New Stratimikos package provides a singe parameter-driver wrapper for all of these

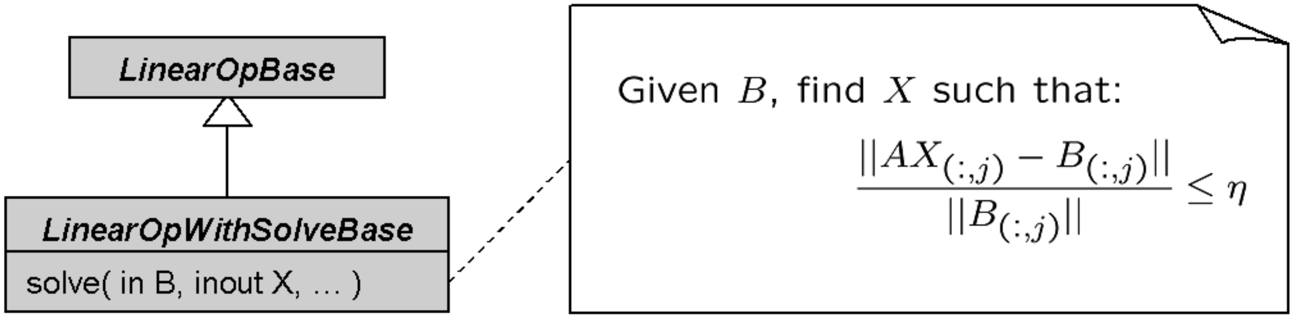
Key Points

- You can create your own PreconditionerFactory subclass!

Sandia National Laboratories

Linear Operator With Solve and Factories

LinearOpWithSolveBase : Combines a linear operator and a linear solver

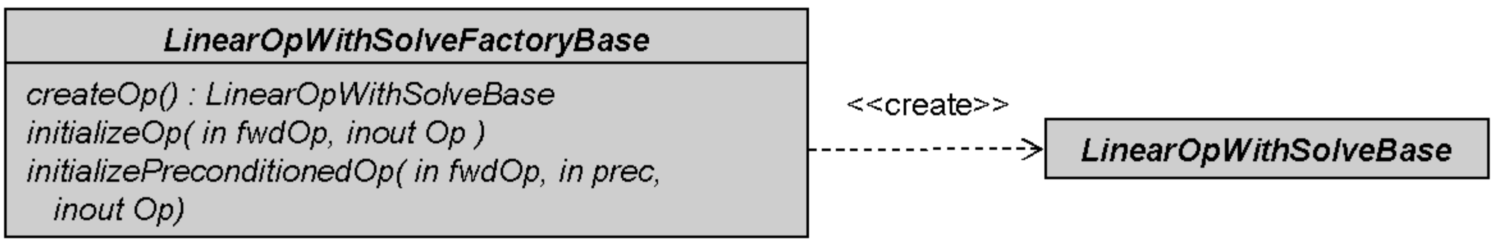


- Appropriate for both direct and iterative solvers
- Supports multiple simultaneous solutions as multi-vectors
- Allows targeting of different solution criteria to different RHSs
- Supports a “default” solve

Key Points

- You can create your own subclass!

LinearOpWithSolveFactoryBase : Uses LinearOpBase objects to initialize LOWSB objects



- Allows unlimited creation/reuse of LinearOpWithSolveBase objects
- Supports reuse of factorizations/preconditioners
- Supports client-created external preconditioners (which are ignored by direct solvers)
- Appropriate for both direct and iterative solvers
- Concrete adaptors for Amesos, AztecOO, and Belos are available
- New Stratimikos package provides a single parameter-driven wrapper to all of these!

Sandia National Laboratories

Introducing Stratimikos

- Stratimikos created Greek words "stratigiki“ (strategy) and "grammikos“ (linear)
- Defines class Thyra::DefaultRealLinearSolverBuilder: Really should be changed to Stratimikos::DefaultLinearSolverBuilder
  - Provides common access to:
    - Linear Solvers: Amesos, AztecOO, Belos, …
    - Preconditioners: Ifpack, ML, …
  - Reads in options through a parameter list (read from XML?)
  - Accepts any linear system objects that provide
    - Epetra_Operator / Epetra_RowMatrix view of the matrix
    - SPMD vector views for the RHS and LHS (e.g. Epetra_[Multi]Vector objects)
- Provides uniform access to linear solver options that can be leveraged across multiple applications and algorithms
- Future: TOPS-2 will add PETSc and other linear solvers and preconditioners!

Key Points

- Stratimikos is an important building block for creating more sophisticated linear solver capabilities!

Sandia National Laboratories

Stratimikos Parameter List and Sublists

```
<ParameterList name="Stratimikos">
  <Parameter name="Linear Solver Type" type="string" value="AztecOO"/>
  <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
  <ParameterList name="Linear Solver Types">
    <ParameterList name="Amesos">
      <Parameter name="Solver Type" type="string" value="Klu"/>
      <ParameterList name="Amesos Settings">
        <Parameter name="MatrixProperty" type="string" value="general"/>
        ...
        <ParameterList name="Mumps"> ... </ParameterList>
        <ParameterList name="Superludist"> ... </ParameterList>
      </ParameterList>
    </ParameterList>
    <ParameterList name="AztecOO">
      <ParameterList name="Forward Solve">
        <Parameter name="Max Iterations" type="int" value="400"/>
        <Parameter name="Tolerance" type="double" value="1e-06"/>
        <ParameterList name="AztecOO Settings">
          <Parameter name="Aztec Solver" type="string" value="GMRES"/>
          ...
        </ParameterList>
      </ParameterList>
      ...
    </ParameterList>
    <ParameterList name="Belos"> ... </ParameterList>
  </ParameterList>
<ParameterList name="Preconditioner Types">
    <ParameterList name="Ifpack">
      <Parameter name="Prec Type" type="string" value="ILU"/>
      <Parameter name="Overlap" type="int" value="0"/>
      <ParameterList name="Ifpack Settings">
        <Parameter name="fact: level-of-fill" type="int" value="0"/>
        ...
      </ParameterList>
    </ParameterList>
    <ParameterList name="ML"> ... </ParameterList>
  </ParameterList>
</ParameterList>
```

Top level parameters

Linear Solvers

Preconditioners

Sublists passed on to package code!

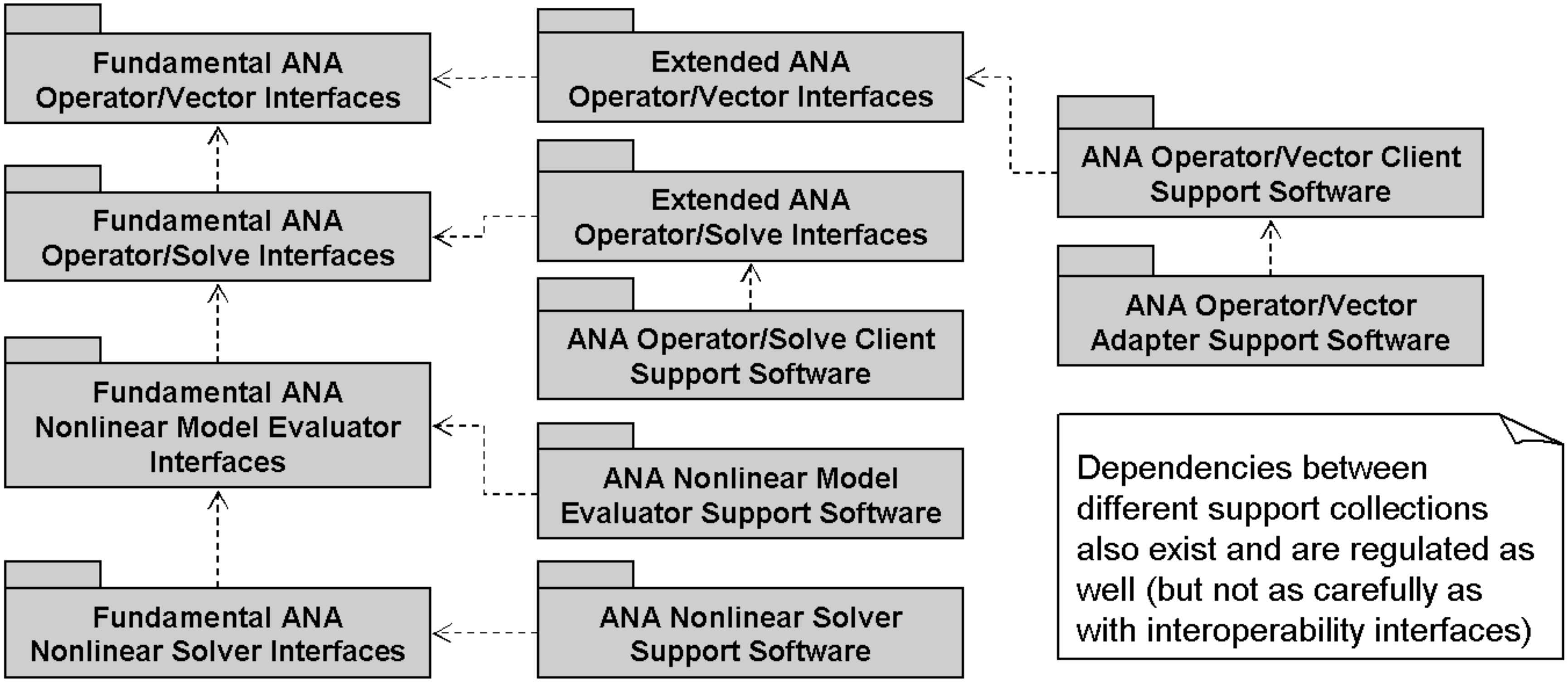
Every parameter and sublist not in red is handled by Thyra code and is fully validated!

See Doxygen documentation for Thyra::DefaultRealLinearSolverBuilder!

Outline

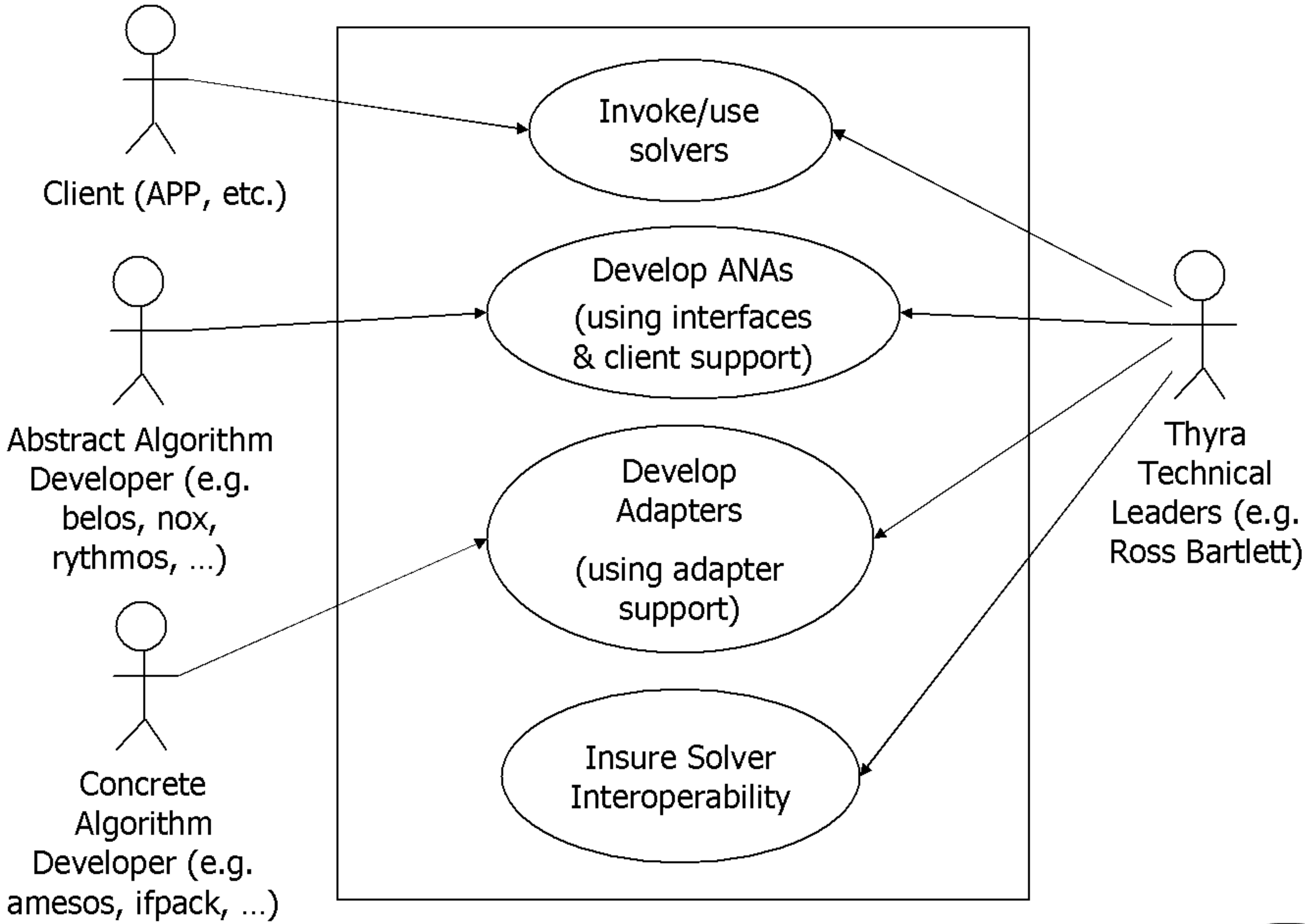
- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Dependencies between real Thyra “Packages”



- The Trilinos package `thyra` is not one monolithic piece of software.
- The interfaces are as minimal as possible and the dependencies between them is **very** carefully regulated.
- The support software is carefully separated from the interoperability interfaces.
- The Trilinos package `thyra` is really at least 11 different “packages” in the pure object-oriented sense [Martin, 2003].
- Of course the Epetra, EpetraExt, etc. adapters are also really separate “packages”.

Thyra Use Cases



Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Sandia National Laboratories

Thyra ANA Implicit Composable Operator/Vector Subclasses

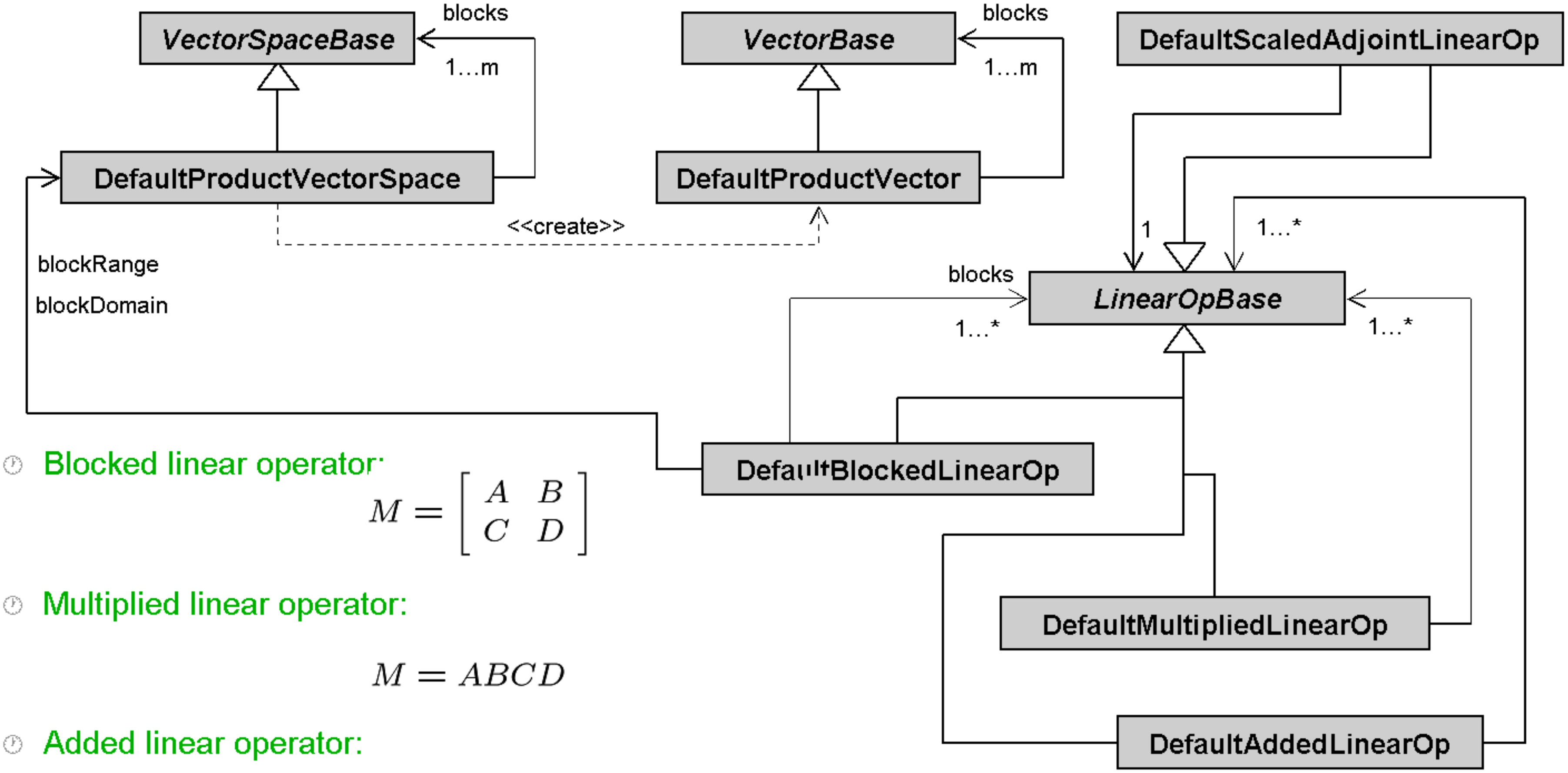
"Composite" subclasses allow a collection of objects to be manipulated as one object

- Product vector spaces and product vectors:
  - Product vector spaces: $\mathcal{X} = \mathcal{V}_1 \times \mathcal{V}_2 \times \ldots \times \mathcal{V}_m$
  - Product vectors: $x^T = \begin{bmatrix} v_1^T & v_2^T & \ldots & v_m^T \end{bmatrix}$

"Decorator" subclasses wrap an object and changes its behavior

- Scaled/Adjoint(transposed) linear operator:

$$M = \alpha A^H$$



- Blocked linear operator:

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

- Multiplied linear operator:

$$M = ABCD$$

- Added linear operator:

$$M = A + B + C + D$$

XXXLinearOpBase interfaces also!

Sandia National Laboratories

Example Composed Linear Operator: DefaultAddedLinearOp

```
template<class Scalar>
void DefaultAddedLinearOp<Scalar>::apply(
  const ETransp                     M_trans,
  const MultiVectorBase<Scalar>     &X,
  MultiVectorBase<Scalar>           *Y,
  const Scalar                      alpha,
  const Scalar                      beta
  ) const
{
  typedef Teuchos::ScalarTraits<Scalar> ST;
  …
  //
  // Y = alpha * op(M) * X + beta*Y
  //
  // =>
  //
  // Y = beta*Y + sum(alpha*op(Op[j])*X),j=0...numOps-1)
  //
  const int numOps = Ops_.size();
  for( int j = 0; j < numOps; ++j )
    Thyra::apply( *getOp(j), M_trans, X, Y, alpha, j==0?beta:ST::one() );
}
```

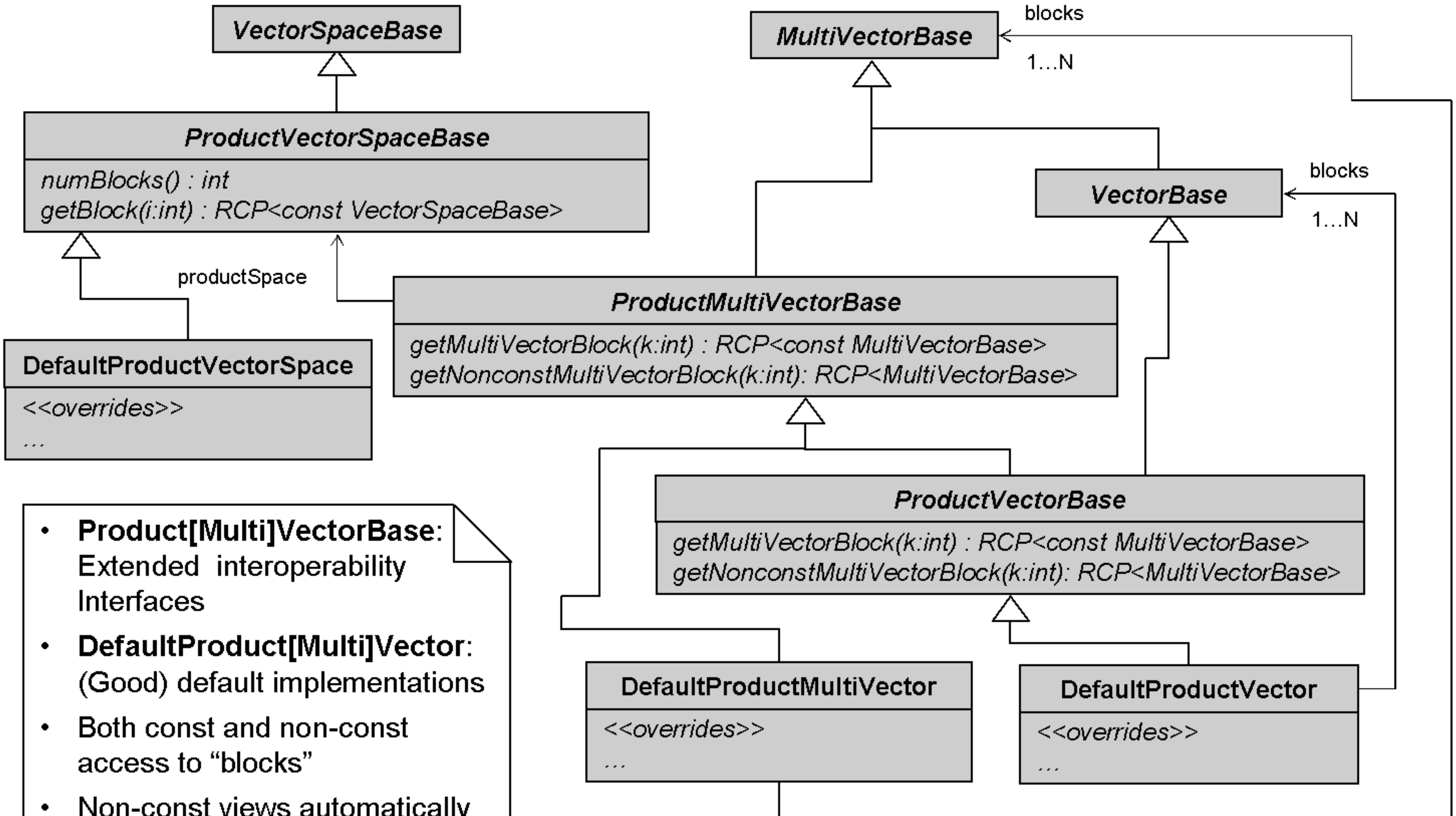
Key Points

- The work these implicit subclasses is very simple
- These are some of the most pure examples of the "Composite" design pattern!

Sandia National Laboratories

Product Vector and Product Space Interfaces & Implementations

- Product vector spaces: $\mathcal{X} = \mathcal{V}_1 \times \mathcal{V}_2 \times \ldots \times \mathcal{V}_m$
- Product vectors: $x^T = \left[\begin{array}{cccc} v_1^T & v_2^T & \ldots & v_m^T \end{array} \right]$

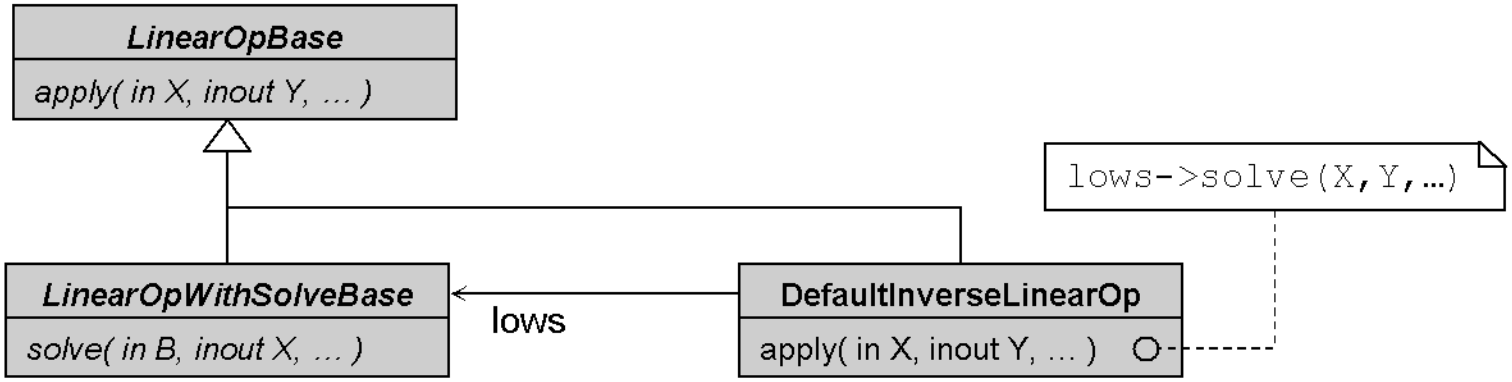


- **Product[Multi]VectorBase**: Extended interoperability Interfaces
- **DefaultProduct[Multi]Vector**: (Good) default implementations
- Both const and non-const access to “blocks”
- Non-const views automatically update parent

Sandia National Laboratories

Linear Solvers as Linear Operators

A linear solver as a linear operator:



C++ code for creating an inverse linear operator:

```
// Create LOWSFB object from Stratimikos
RCP<const LinearOpWithSolveFactoryBase<Scalar> >
   solverFactory = stratimikosLinearSolverBuilder.createSolverStrategy();

// Create an operator that applys the inverse!
RCP<const LinearOpBase<Scalar> > invA = inverse( *solverFactory, A );
```

Key Points:

- Allows a linear solver to be embedded as a linear operator using composed operators!
  - Physics-based preconditioners
  - Subdomain solves
  - etc …

Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

An Example of Composed Operators

Implicitly composed operators: Combine blocked, added, multiplied and adjoint operations

Example:

$$M = \begin{bmatrix} \begin{bmatrix} \gamma BA + C & E + F \\ J^H A & I \end{bmatrix}^H & \beta \begin{bmatrix} Q \\ K \end{bmatrix} \\ \begin{bmatrix} LN^H & \eta P \end{bmatrix} & diag(d) - Q^H Q \end{bmatrix}$$

$$y = Mx$$

Range vectors: $y = \begin{bmatrix} \begin{bmatrix} y_0 \\ y_1 \end{bmatrix} \\ y_2 \end{bmatrix}$

Domain vectors: $x = \begin{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \\ x_2 \end{bmatrix}$

Use Cases:

- Physics-based preconditioners (e.g. Meros)
- Certain optimization formulations
- Implicit RK methods
- “4D” for transient problems
- Stochastic finite element (SFE) methods for UQ
- Multi-physics
- …

See: exampleImplicitlyComposedLinearOperators.cpp

Example Object Structure for Composed Linear Operators

Example: $M_{(1,0)} = \left[\; LN^H \quad \eta P \; \right]$

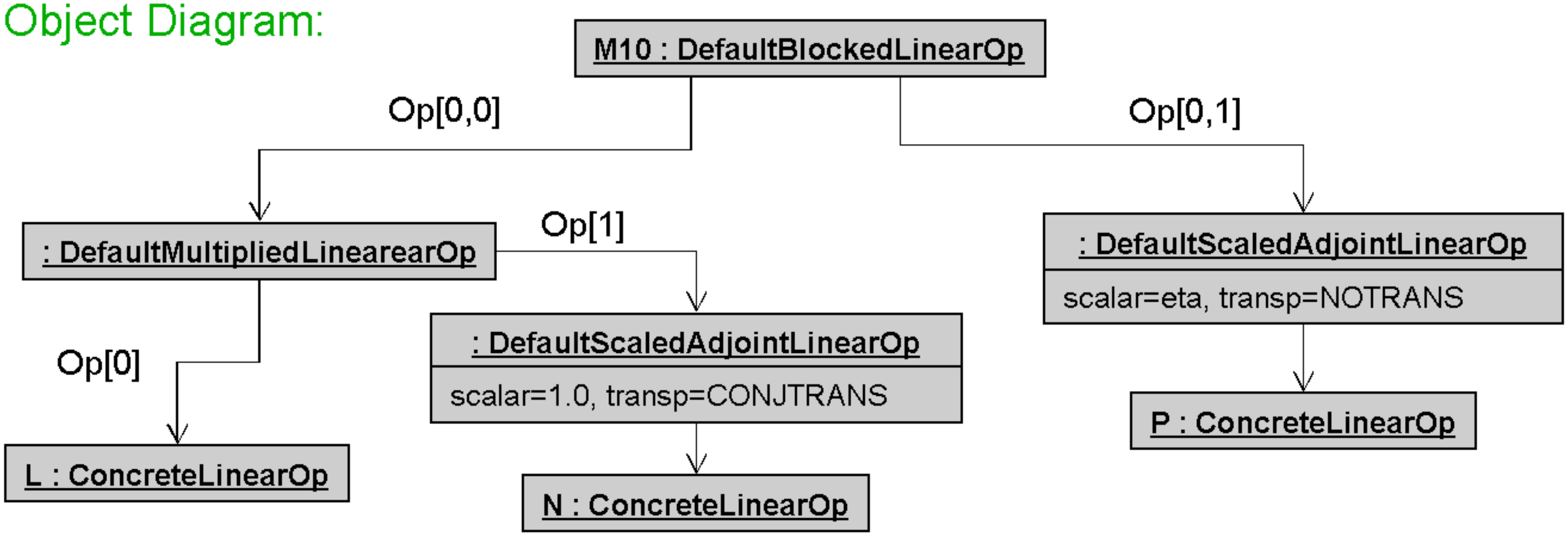
Key Points

- You need to understand this object structure for any sophisticated use!

Thyra C++ Code:

```
// M10 = [ L * N^H,  eta*P ]
const RCP<const LinearOpBase<Scalar> > M10 =
  block1x2(
    multiply(L,adjoint(N)),  scale(eta,P),
    "M10"
    );
```

Object Diagram:



See: exampleImplicitlyComposedLinearOperators.cpp

Example of Describable Output for Composed Operator Structure

Example: $M_{(1,0)} = \left[\begin{array}{cc} LN^H & \eta P \end{array}\right]$

Thyra C++ Code:

```
// M10 = [ L * N^H,  eta*P ]
const RCP<const LinearOpBase<Scalar> > M10 =
  block1x2(
    multiply(L,adjoint(N)),  scale(eta,P),
    "M10"
    );
out << "\nM10 = " << describe(*M10,verbLevel);
```

Key Points

- Very important do debug problems
- This type of introspection is critical to manage the complexity of complex structures

Output from describe(…):

```
M10 = "M10": Thyra::DefaultBlockedLinearOp<float>{rangeDim=4,domainDim=5,numRowBlocks=1,numColBlocks=2}
  Constituent LinearOpBase objects for M = [ Op[0,0] ... ; ... ; ... Op[numRowBlocks-1,numColBlocks-1] ]:
  Op[0,0] = "(L)*(adj(N))": Thyra::DefaultMultipliedLinearOp<float>{rangeDim=4,domainDim=2}
    numOps = 2
    Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
     Op[0] = "L": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=3}
     Op[1] = "adj(N)": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=3,domainDim=2}
       overallScalar=1
       overallTransp=CONJTRANS
       Constituent transformations:
        transp=CONJTRANS
         origOp = "N": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=3}
  Op[0,1] = "4*(P)": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=4,domainDim=3}
    overallScalar=4
    overallTransp=NOTRANS
    Constituent transformations:
     scalar=4
      origOp = "P": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=3}
```

Sandia National Laboratories

More Complete Example of Composed Operator Code

Example:

$$M = \begin{bmatrix} \begin{bmatrix} \gamma BA + C & E + F \\ J^H A & I \end{bmatrix}^H & \beta \begin{bmatrix} Q \\ K \end{bmatrix} \\ \begin{bmatrix} LN^H & \eta P \end{bmatrix} & diag(d) - Q^H Q \end{bmatrix}$$

Thyra C++ Code:

```
const RCP<const LinearOpBase<Scalar> > I =
  identity(space1,"I");

const RCP<const LinearOpBase<Scalar> > D =
  diagonal(d,"D");

const RCP<const LinearOpBase<Scalar> > M00 =
  adjoint(
    block2x2(
      add( scale(gamma,multiply(B,A)), C ),   add(E, F),
      multiply(adjoint(J),A),                 I
      ),
    "M00"
    );

const RCP<const LinearOpBase<Scalar> > M01 =
  scale(
    beta,
    block2x1( Q, K ),
    "M01"
    );

const RCP<const LinearOpBase<Scalar> > M10 =
  block1x2(
    multiply(L,adjoint(N)),  scale(eta,P),
    "M10"
    );

const RCP<const LinearOpBase<Scalar> > M11 =
  subtract( D, multiply(adjoint(Q),Q), "M11" );

const RCP<const LinearOpBase<Scalar> > M =
  block2x2(
    M00, M01,
    M10, M11,
    "M"
    );
```

Output from describe(…):

```
M = "M": Thyra::DefaultBlockedLinearOp<float>{rangeDim=9,domainDim=9,numRowBlocks=2,numColBlocks=2}
 Constituent LinearOpBase objects for M = [ Op[0,0] ... ; ... ; ... Op[numRowBlocks-1,numColBlocks-1] ]:
  Op[0,0] = "M00": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=5,domainDim=5}
    overallScalar=1
    overallTransp=CONJTRANS
    Constituent transformations:
     transp=CONJTRANS
      origOp = Thyra::DefaultBlockedLinearOp<float>{rangeDim=5,domainDim=5,numRowBlocks=2,numColBlocks=2}
        Constituent LinearOpBase objects for M = [ Op[0,0] ... ; ... ; ... Op[numRowBlocks-1,numColBlocks-1] ]:
         Op[0,0] = "(3*((B)*(A)))+(C)": Thyra::DefaultAddedLinearOp<float>{rangeDim=2,domainDim=2}
          numOps=2
          Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
           Op[0] = "3*((B)*(A))": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=2,domainDim=2}
             overallScalar=3
             overallTransp=NOTRANS
             Constituent transformations:
              scalar=3
               origOp = "(B)*(A)": Thyra::DefaultMultipliedLinearOp<float>{rangeDim=2,domainDim=2}
                 numOps = 2
                 Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
                  Op[0] = "B": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=4}
                  Op[1] = "A": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=2}
           Op[1] = "C": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=2}
         Op[0,1] = "(E)+(F)": Thyra::DefaultAddedLinearOp<float>{rangeDim=2,domainDim=3}
          numOps=2
          Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
           Op[0] = "E": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=3}
           Op[1] = "F": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=3}
         Op[1,0] = "(adj(J))*(A)": Thyra::DefaultMultipliedLinearOp<float>{rangeDim=3,domainDim=2}
          numOps = 2
          Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
           Op[0] = "adj(J)": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=3,domainDim=4}
             overallScalar=1
             overallTransp=CONJTRANS
             Constituent transformations:
              transp=CONJTRANS
               origOp = "J": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=3}
           Op[1] = "A": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=2}
         Op[1,1] = Thyra::DefaultIdentityLinearOp<float>{space=Thyra::DefaultSpmdVectorSpace<float>{globalDim=3,localSubDim=3,localOffset=0,comm=NULL}}
  Op[0,1] = "M01": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=5,domainDim=4}
    overallScalar=2
    overallTransp=NOTRANS
    Constituent transformations:
     scalar=2
      origOp = Thyra::DefaultBlockedLinearOp<float>{rangeDim=5,domainDim=4,numRowBlocks=2,numColBlocks=1}
        Constituent LinearOpBase objects for M = [ Op[0,0] ... ; ... ; ... Op[numRowBlocks-1,numColBlocks-1] ]:
         Op[0,0] = "Q": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=4}
         Op[1,0] = "K": Thyra::DefaultSpmdMultiVector<float>{rangeDim=3,domainDim=4}
  Op[1,0] = "M10": Thyra::DefaultBlockedLinearOp<float>{rangeDim=4,domainDim=5,numRowBlocks=1,numColBlocks=2}
    Constituent LinearOpBase objects for M = [ Op[0,0] ... ; ... ; ... Op[numRowBlocks-1,numColBlocks-1] ]:
     Op[0,0] = "(L)*(adj(N))": Thyra::DefaultMultipliedLinearOp<float>{rangeDim=4,domainDim=2}
       numOps = 2
       Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
        Op[0] = "L": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=3}
        Op[1] = "adj(N)": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=3,domainDim=2}
          overallScalar=1
          overallTransp=CONJTRANS
          Constituent transformations:
           transp=CONJTRANS
            origOp = "N": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=3}
     Op[0,1] = "4*(P)": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=4,domainDim=3}
       overallScalar=4
       overallTransp=NOTRANS
       Constituent transformations:
        scalar=4
         origOp = "P": Thyra::DefaultSpmdMultiVector<float>{rangeDim=4,domainDim=3}
  Op[1,1] = "M11": Thyra::DefaultAddedLinearOp<float>{rangeDim=4,domainDim=4}
    numOps=2
    Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
     Op[0] = "D": Thyra::DefaultDiagonalLinearOp<float>{rangeDim=4,domainDim=4}
     Op[1] = "-1*((adj(Q))*(Q))": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=4,domainDim=4}
       overallScalar=-1
       overallTransp=NOTRANS
       Constituent transformations:
        scalar=-1
         origOp = "(adj(Q))*(Q)": Thyra::DefaultMultipliedLinearOp<float>{rangeDim=4,domainDim=4}
           numOps = 2
           Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
            Op[0] = "adj(Q)": Thyra::DefaultScaledAdjointLinearOp<float>{rangeDim=4,domainDim=2}
              overallScalar=1
              overallTransp=CONJTRANS
              Constituent transformations:
               transp=CONJTRANS
                origOp = "Q": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=4}
            Op[1] = "Q": Thyra::DefaultSpmdMultiVector<float>{rangeDim=2,domainDim=4}
```

Tidbits About Thyra Implicitly Composed Linear Operators

- Try to use the exact types for arguments to the non-member constructor functions
  - Typically you want to use :

```
RCP<const LinearOpBase<Scalar> >
```

  - Any difference in types can cause a compilation failure
- If having trouble compiling, try using explicit namespacing and template arguments
  - Example:

```
C = Thyra::multiply<Scalar>( A, B );
```

- Make your code cleaner and avoid problems by injecting function names into your local scope:
  - Example:

```
void foo() {
  using Thyra::multiply;
  …
  RCP<const LinearOpBase<Scalar> > C = multiply(A,B);
```

- There are both const and non-const versions of non-member constructor functions
  - Example:

```
RCP<const LinearOpBase<Scalar> > C = multiply(A,B);
RCP<LinearOpBase<Scalar> > ncC = nonconstMultiply(A,B);
```

  - Non-const version allows components to be modified (rare but important)
- Implicitly composed operator subclasses handle both const and non-const component operators in a single class
  - Const is protected at runtime!

Sandia National Laboratories

Why Should You Use These Implicit Operator Implementations?

- Describable output to show structure
  - Very important to catch mistakes with incompatible objects
- Indented VerboseObject output from operators
  - Makes it easier to disambiguate nested solves (see MixedOrderPhysicsBasedPreconditioner.cpp)
- Supports all Scalar types (e.g. float, double, complex<float>, complex<double>, etc.)
  - See exampleImplicitlyComposedLinearOperators.cpp
- Operator-overloaded wrappers using Handle classes
  - Example:

```
ConstLinearOperator<Scalar>
  M10 = block1x2( L*adjoint(N), eta*P );
```

- Strong runtime checking of compatibility of spaces and good error messages
  - See next slide for an example

Key Points

- You can create these yourself but these are good reasons to use these implementations

Example of Error Output

C++ Code:

```
RCP<const Thyra::LinearOpBase<Scalar> >
  A6b = multiply(origA,origA);
```

Key Points

- As much effort in these classes goes into error detection and reporting than goes into actually functionality!

Exception error message:

```
p=0: *** Caught standard std::exception of type 'Thyra::Exceptions::IncompatibleVectorSpaces' :

 /home/rabartl/PROJECTS/Trilinos.base/Trilinos/packages/thyra/src/support/operator_vector/client_support/Thyra_Assert

 Throw number = 1

 Throw test that evaluated to true: !isCompatible

 DefaultMultipliedLinearOp<Scalar>::initialize(...):

 Spaces check failed for (Ops[0]) * (Ops[1]) where:

   Ops[0]: "origA": Thyra::DefaultSpmdMultiVector<double>{rangeDim=4,domainDim=2}

   Ops[1]: "origA": Thyra::DefaultSpmdMultiVector<double>{rangeDim=4,domainDim=2}

 Error, the following vector spaces are not compatible:

  Ops[0].domain() : Thyra::DefaultSpmdVectorSpace<double>{globalDim=2,localSubDim=2,localOffset=0,comm=Teuchos::Seria

  Ops[1].range() : Thyra::DefaultSpmdVectorSpace<double>{globalDim=4,localSubDim=4,localOffset=0,comm=Teuchos::Serial
```

See thyra/test/operator_vector/test_composite_ops.cpp for more examples

Sandia National Laboratories

Example: An Attempt at a Physics-Based Preconditioner

Operators generated in Sundance very easily!

Second-order (p2) Lagrange Operator: P_2

First-order (p1) Lagrange Operator: P_1

Idea for a preconditioner?

$$\bar{P}_2 = \underbrace{M_{22}^{-1} M_{12}}_{\text{Prolongation from p1 to p2}} \underbrace{P_1^{-1}}_{\text{Inversion on p1}} \underbrace{M_{11}^{-1} M_{21}}_{\text{Restriction from p2 to p1}}$$

Prolongation from p1 to p2

Inversion on p1

Using exact solve or preconditioner for P_1

Restriction from p2 to p1

<u>Preconditioned Linear System</u>

$$P_2 \bar{P}_2 (\bar{P}_2^{-1} x_2) = b_2$$

- Preconditioner involves nested linear solves
  - Inner solves with M_{11} and M_{22} use CG (they are easy)
  - Inner solve with P_1 uses GMRES or just preconditioner for P_1
- Outer preconditioned solve with P2 uses (flexible) GMRES
- Major Problem: This is a singular preconditioner!
  - That's besides the point, since we did not see this right away but the numerical solve sure told us this!
- The Main Point: We want to be able to quickly try out interesting alternatives!

Example: An Attempt at a Physic-Based Preconditioner

Example program: MixedOrderPhysicsBasedPreconditioner.cpp

A. Read in the problem matrices (non-ANA code)

B. Create the linear solver (and preconditioner) factories using Stratimikos

C. Create the physics-based preconditioner using implicit composed operators

D. Create the overall linear solver using implicit composed operators

E. Solve the overall linear system

See: stratimikos/example/MixedOrderPhysicsBasedPreconditioner.cpp

Example: An Attempt at a Physic-Based Preconditioner

A) Read in the problem matrices (non-ANA code)

```
typedef RCP<const Thyra::LinearOpBase<double> > LinearOpPtr;

LinearOpPtr P1=readEpetraCrsMatrixFromMatrixMarketAsLinearOp(
  baseDir+"/P1.mtx",comm,"P1");

LinearOpPtr P2= readEpetraCrsMatrixFromMatrixMarketAsLinearOp(
  baseDir+"/P2.mtx",comm,"P2");

LinearOpPtr M11=readEpetraCrsMatrixFromMatrixMarketAsLinearOp(
  baseDir+"/M11.mtx",comm,"M11");

LinearOpPtr M22=readEpetraCrsMatrixFromMatrixMarketAsLinearOp(
  baseDir+"/M22.mtx",comm,"M22");

LinearOpPtr M12=readEpetraCrsMatrixFromMatrixMarketAsLinearOp(
  baseDir+"/M12.mtx",comm,"M12");

LinearOpPtr M21=readEpetraCrsMatrixFromMatrixMarketAsLinearOp(
  baseDir+"/M21.mtx",comm,"M21");
```

See: MixedOrderPhysicsBasedPreconditioner.cpp

Example: An Attempt at a Physic-Based Preconditioner

B) Create the linear solver factories using Stratimikos (C++ Code)

```
// Read in the overall parameter list from an XML file
RCP<ParameterList> paramList =
  Teuchos::getParametersFromXmlFile( baseDir+"/"+paramsFile );

// Break of the Stratimikos sublists for each linear operator

Thyra::DefaultRealLinearSolverBuilder M11_linsolve_strategy_builder;
M11_linsolve_strategy_builder.setParameterList(
  sublist(paramList,"M11 Solver",true) );

Thyra::DefaultRealLinearSolverBuilder M22_linsolve_strategy_builder;
M22_linsolve_strategy_builder.setParameterList(
  sublist(paramList,"M22 Solver",true) );

Thyra::DefaultRealLinearSolverBuilder P1_linsolve_strategy_builder;
P1_linsolve_strategy_builder.setParameterList(
  sublist(paramList,"P1 Solver",true) );

Thyra::DefaultRealLinearSolverBuilder P2_linsolve_strategy_builder;
P2_linsolve_strategy_builder.setParameterList(
  sublist(paramList,"P2 Solver",true) );
```

See: MixedOrderPhysicsBasedPreconditioner.cpp

Example: An Attempt at a Physic-Based Preconditioner

B) Create the linear solver factories using Stratimikos (Abbreviated XML File)

```
<ParameterList>
  <ParameterList name="M11 Solver">
    <Parameter name="Linear Solver Type" type="string" value="Belos"/>
    <Parameter name="Preconditioner Type" type="string" value="Ifpack"/>
    <ParameterList name="Linear Solver Types">
      <ParameterList name="Belos">
        <Parameter name="Solver Type" type="string" value="Block CG"/>
        <ParameterList name="Solver Types">
          <ParameterList name="Block CG">
            …
          </ParameterList>
        </ParameterList>
        <ParameterList name="VerboseObject">
          <Parameter name="Verbosity Level" type="string" value="none"/>
        </ParameterList>
      </ParameterList>
    </ParameterList>
    <ParameterList name="Preconditioner Types">
      <ParameterList name="Ifpack">
        …
      </ParameterList>
    </ParameterList>
    …
  </ParameterList>
  <ParameterList name="M22 Solver"> … </ParameterList>
  <ParameterList name="P1 Solver"> … </ParameterList>
  <ParameterList name="P2 Solver"> … </ParameterList>
</ParameterList>
```

Key Points

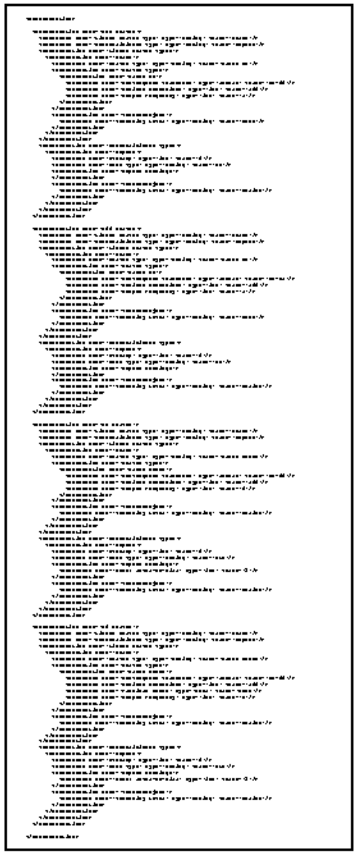
- Users can construct their own parameter sublist structures and embed Stratimikos sublists
- The "VerboseObject" sublist allows users to take full control of output … Very important for complex structures.

See: stratimikos/example/_MixedOrderPhysicsBasedPreconditioner.Belos.xml

Sandia National Laboratories

Example: An Attempt at a Physic-Based Preconditioner

B) Create the linear solver factories using Stratimikos (Full XML File)



Key Points

- Parameter lists are really the main interface that most users will interact with our solvers through in the future!
- We have much more complex examples in production codes
- There are many tools for manipulating XML files

See: stratimikos/example/_MixedOrderPhysicsBasedPreconditioner.Belos.xml

Example: An Attempt at a Physic-Based Preconditioner

C) Create the physics-based preconditioner (C++ code)

$$\bar{P}_2 = M_{22}^{-1} M_{12} P_1^{-1} M_{11}^{-1} M_{21}$$

```
LinearOpPtr invM11 = inverse(*M11_linsolve_strategy,M11);

LinearOpPtr invM22 = inverse(*M22_linsolve_strategy,M22);

LinearOpPtr invP1;
if(invertP1) {
  invP1 = inverse(*P1_linsolve_strategy,P1);
}
else {
  RCP<Thyra::PreconditionerBase<double> >
    precP1 = prec(*P1_prec_strategy,P1);
  invP1 = precP1->getUnspecifiedPrecOp();
}

LinearOpPtr P2ToP1 = multiply( invM11, M21 );

LinearOpPtr P1ToP2 = multiply( invM22, M12 );

LinearOpPtr precP2Op = multiply( P1ToP2, invP1, P2ToP1 );

*out << "\nprecP2Op = " << describe(*precP2Op,verbLevel) << "\n";
```

Key Points

- Very little user code to construct these types of composed linear operators!
- Easy to embed linear solvers as linear operators!

See: MixedOrderPhysicsBasedPreconditioner.cpp

Example: An Attempt at a Physic-Based Preconditioner

C) Create the physics-based preconditioner (Partial outputted structure)

$$\bar{P}_2 = M_{22}^{-1} M_{12} P_1^{-1} M_{11}^{-1} M_{21}$$

```
precP2Op = "((inv(M22))*(M12))*(invP1)*((inv(M11))*(M21))": Thyra::DefaultMultipliedLinearOp<doub
  numOps = 3
  Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
   Op[0] = "(inv(M22))*(M12)": Thyra::DefaultMultipliedLinearOp<double>{rangeDim=289,domainDim=81
     numOps = 2
     Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
      Op[0] = "inv(M22)": Thyra::DefaultInverseLinearOp<double>{rangeDim=289,domainDim=289}:
        lows = "M22": Thyra::BelosLinearOpWithSolve<double>{rangeDim=289,domainDim=289}
          iterativeSolver = Belos::BlockCGSolMgr<...,double>{Ortho Type='DGKS', Block Size=1}
          fwdOp = "M22": Thyra::EpetraLinearOp{rangeDim=289,domainDim=289}
            op=Epetra_CrsMatrix
          rightPrecOp = Thyra::EpetraLinearOp{rangeDim=289,domainDim=289
            op=Ifpack_AdditiveSchwarz<Ifpack_IC>
      Op[1] = "M12": Thyra::EpetraLinearOp{rangeDim=289,domainDim=81}
        op=Epetra_CrsMatrix
   Op[1] = "invP1": Thyra::EpetraLinearOp{rangeDim=81,domainDim=81}
     op=Ifpack_AdditiveSchwarz<Ifpack_ILU>
   Op[2] = "(inv(M11))*(M21)": Thyra::DefaultMultipliedLinearOp<double>{rangeDim=81,domainDim=289
     numOps = 2
     Constituent LinearOpBase objects for M = Op[0]*...*Op[numOps-1]:
      Op[0] = "inv(M11)": Thyra::DefaultInverseLinearOp<double>{rangeDim=81,domainDim=81}:
        lows = "M11": Thyra::BelosLinearOpWithSolve<double>{rangeDim=81,domainDim=81}
          …
      Op[1] = "M21": Thyra::EpetraLinearOp{rangeDim=81,domainDim=289}
        …
```

See: MixedOrderPhysicsBasedPreconditioner.cpp

Example: An Attempt at a Physic-Based Preconditioner

D) Create the overall linear solver

$$P_2\bar{P}_2(\bar{P}_2{}^{-1}x_2) = b_2$$

```
RCP<Thyra::LinearOpWithSolveBase<double> >
  P2_lows = P2_linsolve_strategy->createOp();
if(useP1Prec) {
  *out << "\nCreating the solver P2 using the specialized precP2Op\n";
  initializePreconditionedOp<double>( *P2_linsolve_strategy, P2,
    unspecifiedPrec(precP2Op), &*P2_lows );
}
else {
  *out << "\nCreating the solver P2 using algebraic preconditioner\n";
  initializeOp( *P2_linsolve_strategy, P2, &*P2_lows );
}
```

Key Points

- Switching between radically different preconditioning strategies is easy

See: MixedOrderPhysicsBasedPreconditioner.cpp

Example: An Attempt at a Physic-Based Preconditioner

E) Solve the overall linear system

$$P_2 \bar{P}_2 (\bar{P}_2{}^{-1} x) = b$$

```
VectorPtr x = createMember(P2->domain());
VectorPtr b = createMember(P2->range());
Thyra::randomize(-1.0,+1.0,&*b);
Thyra::assign(&*x,0.0); // Must give an initial guess!

Thyra::SolveStatus<double>
  solveStatus = solve( *P2_lows, Thyra::NOTRANS, *b, &*x );

*out << "\nSolve status:\n" << solveStatus;

*out << "\nSolution ||x|| = " << Thyra::norm(*x) << "\n";

if (showParams) {
  *out << "\nParameter list after use:\n\n";
  paramList->print(*out,PLPrintOptions().indent(2).showTypes(true));
}
```

- See whatever output from each nested linear solver by setting the "VerboseObject" sublist to the appropriate level

See: MixedOrderPhysicsBasedPreconditioner.cpp

Outline

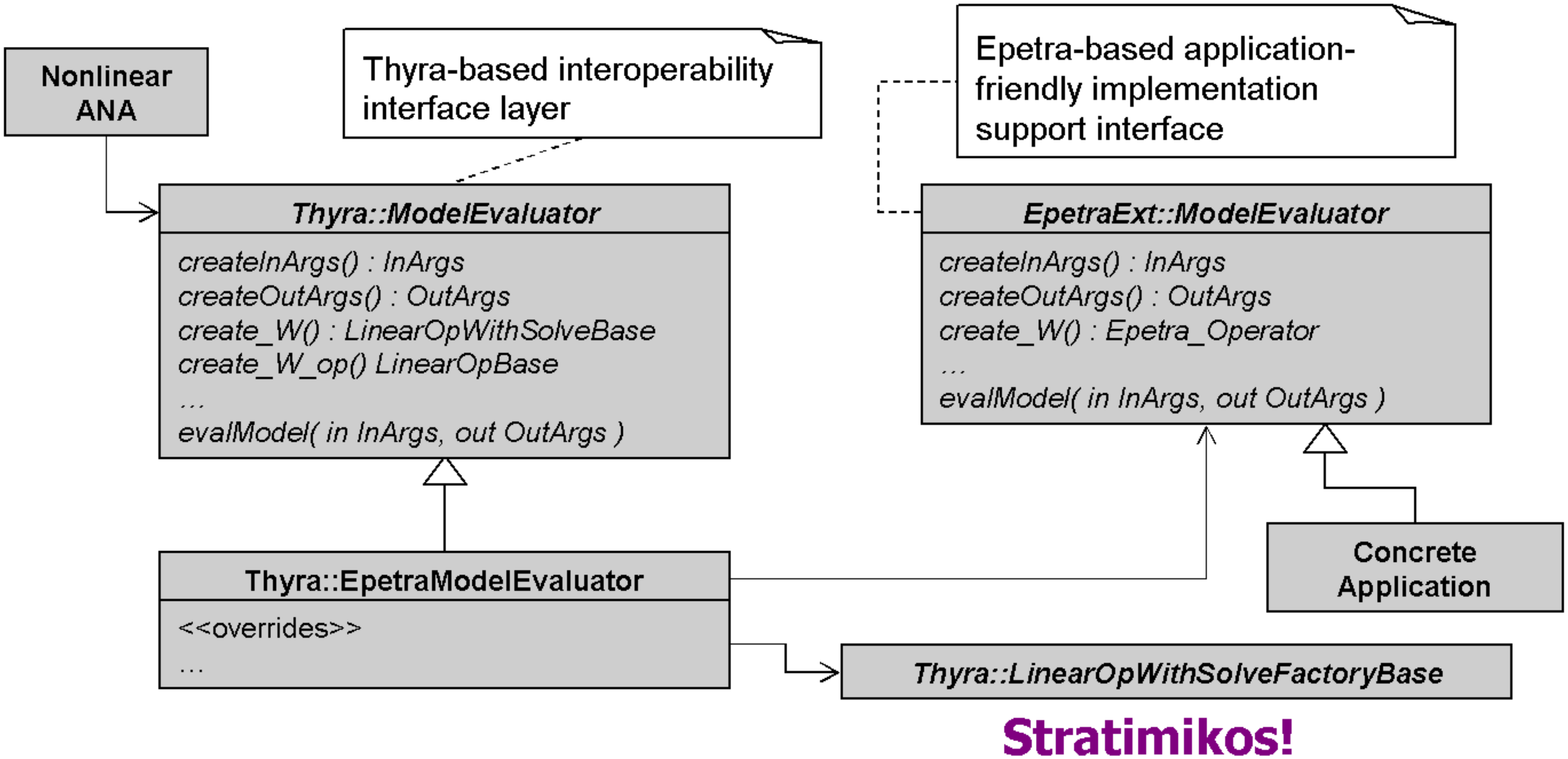
- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Some Examples of Nonlinear Problems Supported by ModelEvaluator

Nonlinear equations:	Solve $f(x) = 0$ for $x \in \mathbf{R}^n$
Stability analysis:	For $f(x,p) = 0$ find space $p \in \mathcal{P}$ such that $\frac{\partial f}{\partial x}$ is singular
Explicit ODEs:	Solve $\dot{x} = f(x,t) = 0, t \in [0,T]$, $x(0) = x_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$
DAEs/Implicit ODEs:	Solve $f(\dot{x}(t), x(t), t) = 0, t \in [0,T]$, $x(0) = x_0$, $\dot{x}(0) = x'_0$ for $x(t) \in \mathbf{R}^n, t \in [0,T]$
Explicit ODE Forward Sensitivities:	Find $\frac{\partial x}{\partial p}(t)$ such that: $\dot{x} = f(x,p,t) = 0, t \in [0,T]$, $x(0) = x_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$
DAE/Implicit ODE Forward Sensitivities:	Find $\frac{\partial x}{\partial p}(t)$ such that: $f(\dot{x}(t), x(t), p, t) = 0, t \in [0,T]$, $x(0) = x_0$, $\dot{x}(0) = x'_0$, for $x(t) \in \mathbf{R}^n, t \in [0,T]$
Unconstrained Optimization:	Find $p \in \mathbf{R}^m$ that minimizes $g(p)$
Constrained Optimization:	Find $x \in \mathbf{R}^n$ and $p \in \mathbf{R}^m$ that: minimizes $g(x,p)$ such that $f(x,p) = 0$
ODE Constrained Optimization:	Find $x(t) \in \mathbf{R}^n$ in $t \in [0,T]$ and $p \in \mathbf{R}^m$ that: minimizes $\int_0^T g(x(t),p)$ such that $\dot{x} = f(x(t),p,t) = 0$, on $t \in [0,T]$ where $x(0) = x_0$

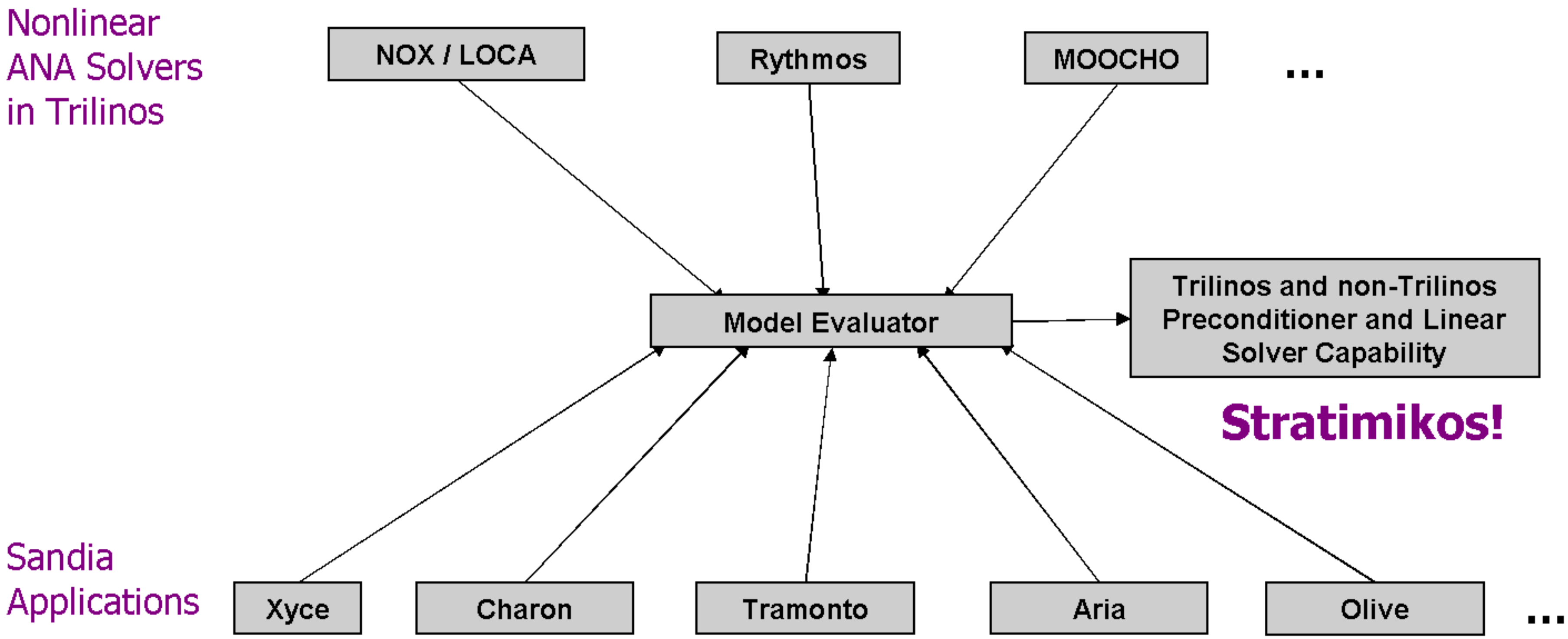
Sandia National Laboratories

Model Evaluator : Thyra and EpetraExt Versions



- Thyra::ModelEvaluator and EpetraExt::ModelEvaluator are near mirror copies of each other.
- Thyra::EpetraModelEvaluator is fully general adapter class that can use any linear solver through a Thyra::LinearOpWithSolveFactoryBase object it is configured with
- Stateless model that allows for efficient multiple shared calculations (e.g. automatic differentation)
- Adding input and output arguments involves
  - Modifying only the classes Thyra::ModelEvaluator, EpetraExt::ModelEvaluator, and Thyra::EpetraModelEvaluator
  - Only recompilation of Nonlinear ANA and Concrete Application code

Nonlinear Algorithms and Applications : Thyra & Model Evaluator!



Key Points

- Provide single interface from nonlinear ANAs to applications
- Provide single interface for applications to implement to access nonlinear ANAs
- Provides shared, uniform access to linear solver capabilities
- Once an application implements support for one ANA, support for other ANAs can quickly follow

Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Example: Implicit RK Method from Rythmos

Implicit ODE/DAE: $f(\dot{x}, x, t) = 0,\ t \in [t_0, t_f],\ \ x(t_0) = x_0,\ \ \dot{x}(t_0) = \dot{x}_0$

Fully Implicit RK Time Step Equations: Solve $\bar{f}(\bar{x}) = 0$ to advance from t_k to t_{k+1}

Collocation eqns: $\bar{f}_i(\bar{x}) = f\left(\dot{x}_i, x_k + \Delta t \sum_{j=0}^{p-1} a_{ij}\dot{x}_j, t_k + c_i \Delta t\right) = 0,$ for $i = 0 \ldots p{-}1$

Stage derivatives: $\bar{x}^T = \left[\ \dot{x}_0 \ \ \dot{x}_1 \ \ \cdots \ \ \dot{x}_{p-1}\ \right]^T$

Butcher Tableau:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array} = \begin{array}{c|cccc} c_0 & a_{0,0} & a_{0,1} & \cdots & a_{0,p-1} \\ c_1 & a_{1,0} & a_{1,1} & \cdots & a_{1,p-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{p-1} & a_{p-1,0} & a_{p-1,1} & \cdots & a_{p-1,p-1} \\ \hline & b_0 & b_1 & \cdots & b_{p-1} \end{array}$$

Newton System for RK Time Step Equations: $\Delta\bar{x} = -(\bar{W}_k)^{-1}\bar{f}(\bar{x}_k)$

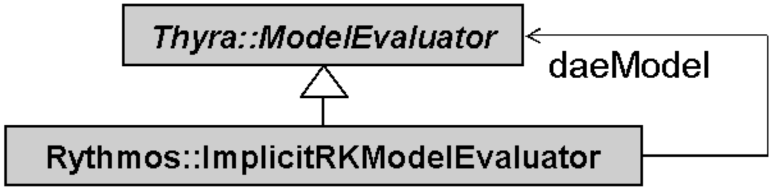
Block Structure:

$$\bar{f} = \begin{bmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{p-1} \end{bmatrix} \qquad \bar{x} = \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_{p-1} \end{bmatrix} \qquad \frac{\partial \bar{f}}{\partial \bar{x}} = \bar{W} = \begin{bmatrix} \bar{W}_{0,0} & \bar{W}_{0,1} & \cdots & \bar{W}_{0,p-1} \\ \bar{W}_{1,0} & \bar{W}_{1,1} & \cdots & \bar{W}_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{W}_{p-1,0} & \bar{W}_{p-1,1} & \cdots & \bar{W}_{p-1,p-1} \end{bmatrix}$$

$$\bar{W}_{i,j} = \frac{\partial \bar{f}_i}{\partial \bar{x}_j} = \frac{\partial f}{\partial \dot{x}} + \Delta t\, a_{i,j} \frac{\partial f}{\partial x},\ \text{for } i = 0 \ldots p{-}1,\ j = 0 \ldots p{-}1$$

See: Rythmos_ImplicitRKModelEvaluator.hpp

Example: Implicit RK Method from Rythmos



$$\bar{f} = \begin{bmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{p-1} \end{bmatrix} \qquad \bar{x} = \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_{p-1} \end{bmatrix}$$

```
template<class Scalar>
void ImplicitRKModelEvaluator<Scalar>::initializeIRKModel(
  const RCP<const Thyra::ModelEvaluator<Scalar> > &daeModel,
  const Thyra::ModelEvaluatorBase::InArgs<Scalar> &basePoint,
  const RCP<Thyra::LinearOpWithSolveFactoryBase<Scalar> > &irk_W_factory,
  const RKButcherTableau<Scalar> &irkButcherTableau
  )
{

  daeModel_ = daeModel;
  basePoint_ = basePoint;
  irk_W_factory_ = irk_W_factory;
  irkButcherTableau_ = irkButcherTableau;

  const int numStages = irkButcherTableau_.numStages();

  x_bar_space_ = productVectorSpace(daeModel_->get_x_space(),numStages);
  f_bar_space_ = productVectorSpace(daeModel_->get_f_space(),numStages);

}
```

Example: Implicit RK Method from Rythmos

$$\frac{\partial \bar{f}}{\partial \bar{x}} = \bar{W} = \begin{bmatrix} \bar{W}_{0,0} & \bar{W}_{0,1} & \cdots & \bar{W}_{0,p-1} \\ \bar{W}_{1,0} & \bar{W}_{1,1} & \cdots & \bar{W}_{1,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{W}_{p-1,0} & \bar{W}_{p-1,1} & \cdots & \bar{W}_{p-1,p-1} \end{bmatrix}$$

```
template<class Scalar>
RCP<Thyra::LinearOpBase<Scalar> >
ImplicitRKModelEvaluator<Scalar>::create_W_op() const
{
  // Create the block structure for W_op_bar right away!
  const int numStages = irkButcherTableau_.numStages();
  RCP<Thyra::PhysicallyBlockedLinearOpBase<Scalar> >
    W_op_bar = Thyra::defaultBlockedLinearOp<Scalar>();
  W_op_bar->beginBlockFill( f_bar_space_, x_bar_space_ );
  for ( int i = 0; i < numStages; ++i )
    for ( int j = 0; j < numStages; ++j )
      W_op_bar->setNonconstBlock( i, j, daeModel_->create_W_op() );
  W_op_bar->endBlockFill();
  return W_op_bar;
}
```

- create_W_op() is required to return LinearOpBase objects that have valid range and domain spaces!

See: Rythmos_ImplicitRKModelEvaluator.hpp

Example: Implicit RK Method from Rythmos

$$(\bar{x}) \rightarrow \bar{f},\;\; \bar{f}_i(\bar{x}) = f\left(\dot{x}_i, x_k + \Delta t \sum_{j=0}^{p-1} a_{ij}\dot{x}_j, t_k + c_i \Delta t\right),\ \text{for } i = 0 \ldots p-1$$

```
template<class Scalar>
void ImplicitRKModelEvaluator<Scalar>::evalModelImpl(
  const Thyra::ModelEvaluatorBase::InArgs<Scalar>& inArgs_bar,
  const Thyra::ModelEvaluatorBase::OutArgs<Scalar>& outArgs_bar
  ) const
{

  // Typedefs …

  //
  // A) Unwrap the inArgs and outArgs to get at product vectors and block op
  //

  …

  //
  // B) Assemble f_bar and W_op_bar by looping over stages
  //

  …

}
```

See: Rythmos_ImplicitRKModelEvaluator.hpp

Example: Implicit RK Method from Rythmos

```
template<class Scalar>
void ImplicitRKModelEvaluator<Scalar>::evalModelImpl(
  const Thyra::ModelEvaluatorBase::InArgs<Scalar>& inArgs_bar,
  const Thyra::ModelEvaluatorBase::OutArgs<Scalar>& outArgs_bar
  ) const
{

  using Teuchos::rcp_dynamic_cast;
  typedef ScalarTraits<Scalar> ST;
  typedef Thyra::ModelEvaluatorBase MEB;
  typedef Thyra::VectorBase<Scalar> VB;
  typedef Thyra::ProductVectorBase<Scalar> PVB;
  typedef Thyra::BlockedLinearOpBase<Scalar> BLWB;

  //
  // A) Unwrap the inArgs and outArgs to get at product vectors and block op
  //

  const RCP<const PVB> x_bar = rcp_dynamic_cast<const PVB>(inArgs_bar.get_x(), true);
  const RCP<PVB> f_bar = rcp_dynamic_cast<PVB>(outArgs_bar.get_f(), true);
  RCP<BLWB> W_op_bar = rcp_dynamic_cast<BLWB>(outArgs_bar.get_W_op(), true);

  …

}
```

Key Points

- Dynamic casting to get at the appropriate interfaces is required
- We don't dynamic cast to concrete classes!

See: Rythmos_ImplicitRKModelEvaluator.hpp

Sandia National Laboratories

Example: Implicit RK Method from Rythmos

```
//
// B) Assemble f_bar and W_op_bar by looping over stages
//

MEB::InArgs<Scalar> daeInArgs = daeModel_->createInArgs();
MEB::OutArgs<Scalar> daeOutArgs = daeModel_->createOutArgs();
const RCP<VB> x_i = createMember(daeModel_->get_x_space());
daeInArgs.setArgs(basePoint_);

const int numStages = irkButcherTableau_.numStages();

for ( int i = 0; i < numStages; ++i ) {

  // B.1) Setup the DAE's inArgs for stage f(i) ...
  …

  // B.2) Setup the DAE's outArgs for stage f(i) ...
  …

  // B.3) Compute f_bar(i) and/or W_op_bar(i,0) ...
  …

  // B.4) Evaluate the rest of the W_op_bar(i,j=1...numStages-1) ...
  …

}
```

See: Rythmos_ImplicitRKModelEvaluator.hpp

Example: Implicit RK Method from Rythmos

```
for ( int i = 0; i < numStages; ++i ) {

  // B.1) Setup the DAE's inArgs for stage f(i) ...
  assembleIRKState( i, irkButcherTableau_.A(), delta_t_, *x_old_, *x_bar, &*x_i );
  daeInArgs.set_x( x_i );
  daeInArgs.set_x_dot( x_bar->getVectorBlock(i) );
  daeInArgs.set_t( t_old_ + irkButcherTableau_.c()(i) * delta_t_ );
  daeInArgs.set_alpha(ST::one());
  daeInArgs.set_beta( delta_t_ * irkButcherTableau_.A()(i,0) );

  // B.2) Setup the DAE's outArgs for stage f(i) ...
  if (!is_null(f_bar))
    daeOutArgs.set_f( f_bar->getNonconstVectorBlock(i) );
  if (!is_null(W_op_bar))
    daeOutArgs.set_W_op(W_op_bar->getNonconstBlock(i,0));

  // B.3) Compute f_bar(i) and/or W_op_bar(i,0) ...
  daeModel_->evalModel( daeInArgs, daeOutArgs );

  // B.4) Evaluate the rest of the W_op_bar(i,j=1...numStages-1) ...
  if (!is_null(W_op_bar)) {
    for ( int j = 1; j < numStages; ++j ) {
      daeInArgs.set_beta( delta_t_ * irkButcherTableau_.A()(i,j) );
      daeOutArgs.set_W_op(W_op_bar->getNonconstBlock(i,j));
      daeModel_->evalModel( daeInArgs, daeOutArgs );
      daeOutArgs.set_W_op(Teuchos::null);
    }
  }

}
```

$$\bar{x} = \begin{bmatrix} \dot{x}_0 \\ \dot{x}_1 \\ \vdots \\ \dot{x}_{p-1} \end{bmatrix}$$

$$\bar{f} = \begin{bmatrix} \bar{f}_0 \\ \bar{f}_1 \\ \vdots \\ \bar{f}_{p-1} \end{bmatrix}$$

$$\bar{f}_i(\bar{x}) = f\left(\dot{x}_i, x_k + \Delta t \sum_{j=0}^{p-1} a_{ij}\dot{x}_j, t_k + c_i \Delta t\right)$$

$$\bar{W}_{i,j} = \frac{\partial f}{\partial \dot{x}} + \Delta t a_{i,j} \frac{\partial f}{\partial x}$$

Sandia National Laboratories

Example: Multi-Period Optimization Problem (MOOCHO)

Multi-Period Optimization Problem:

Minimize: $\sum_{i=0}^{N-1} \beta_i g_i(x_i, p)$

Subject to: $f(x_i, p, q_i) = 0, \text{ for } i = 0 \ldots N-1$

where: x_i: State variables for period *i*
p: Optimization parameters
q_i: Input parameters for period *i*

Use Cases:

- Parameter estimation using multiple data points
- Robust optimization under uncertainty
- Design under multiple operating conditions
- ...

Abstract Form of Optimization Problem:

Minimize: $\bar{g}(\bar{x}, p)$

Subject to: $\bar{f}(\bar{x}, p) = 0$

where:

$$\bar{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{p-1} \end{bmatrix} \quad \bar{g}(\bar{x}, p) = \sum_{i=0}^{N-1} \beta_i g_i(x_i, p) \quad \frac{\partial \bar{f}}{\partial \bar{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_0} & & & \\ & \frac{\partial f}{\partial x_1} & & \\ & & \ddots & \\ & & & \frac{\partial f}{\partial x_{N-1}} \end{bmatrix}$$

$$\bar{f} = \begin{bmatrix} f(x_0, p, q_0) \\ f(x_1, p, p_1) \\ \vdots \\ f(x_{N-1}, p, q_{N-1}) \end{bmatrix}$$

See: Thyra_DefaultMultiPeriodModelEvaluator.hpp

Outline

- Background and Introduction to Abstract Numerical Algorithms (ANAs)
- Thyra Operator/Vector Interfaces, Operator/Solve Interfaces, and Stratimikos
- Thyra Dependency Structure and Use Cases
- Overview Implicitly Composed Operators
- Examples of Implicitly Composed Operators
- Overview of Thyra Nonlinear ModelEvaluator Interface
- Examples of Composed Operators in the Construction of Composed ModelEvaluators
- Wrap Up

Upcoming Thyra Refactorings

- Refactorings that that <u>will not</u> require changes to user code
  - Explicit template instantiation [Optional]
    - Decrease build types
    - Real library object code
    - Improve the development cycle
  - Removal of support for different range and domain scalar types
- Refactorings that <u>will</u> require changes to user code
  - Pure non-member function interface
    - More consistent user API
    - Allows for future refactorings without requiring changes to user code
    - See technical report SAND2007-4078
  - Incorporation of new Teuchos memory-safe classes
    - Shorter argument lists
    - Fewer memory leaks and segfaults
- KEY POINT! All of these refactorings will leave deprecated interfaces in place for one major Trilinos release and will support a process to help users upgrade their codes! => See Tomorrows Teuchos Talk!

Sandia National Laboratories

Summary

- Thyra supports the interoperability and development of Abstract Numerical Algorithms (ANAs)
- Thyra provides implicitly composable linear operator and vector subclasses to support the creation of specialized solvers for:
  - Physics-based preconditioners (e.g. Meros)
  - Multi-period optimization
  - Implicit RK methods
  - “4D” for transient problems
  - Stochastic finite element (SFE) methods for UQ
  - Multi-physics
  - …
- Composable operators used to build composable nonlinear models (i.e. ModelEvaluator subclasses)

Thyra is ready to go, let’s use it!

Please talk with me about how Thyra might help you!

Sandia National Laboratories

THE END

- References:

[Martin, 2003] Robert C. Martin, *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall, 2003