Model Representation and an Open Solver Interface

Robert Fourer and Jun Ma
Northwestern University

Kipp Martin
University of Chicago

Matthew Saltzman
Clemson University

November 14, 2005

ICS Wants YOU!

The ***INFORMS Computing Society***
is where ***computation*** meets ***OR/MS.***

Check us out :

- Attend one of 34 ICS-sponsored sessions here
- Read the INFORMS Journal on Computing
- Visit us online at www.informs.org/Subdiv/Society/ICS/

Join the group where people speak your language.

ICS is INFORMS’ technology leading edge.

Outline

The Problem

The API

The Libraries
- OSCommon
- OSSolver
- OSAgent
- OSUtilities

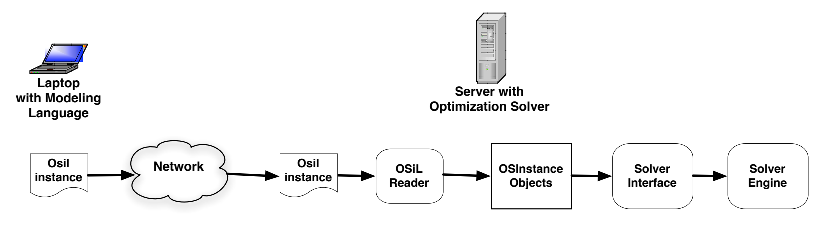
Summary

The Problem

Consider the following scenarios or problems

- You have a model you wish to optimize but don't have the appropriate solver on your machine. How do you access a solver on another machine?
- You need to test an algorithm on a variety of solvers but the solvers require inputs in different formats.
- You need to create a test bed of problem instances (linear, quadratic, nonlinear, stochastic, etc). What format do you use for storing these instances?

The Problem

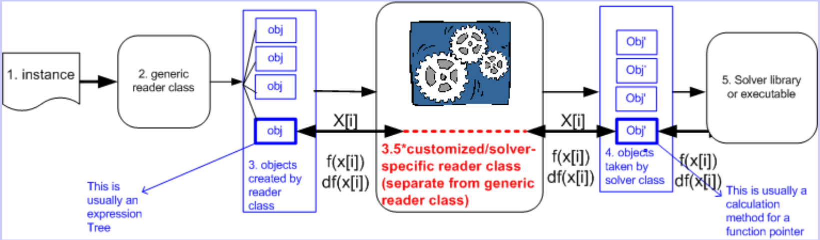

Laptop
with Modeling
Language
Server with
Optimization Solver
Osil
instance
Network
Osil
instance
OSiL
Reader
OSInstance
Objects
Solver
Interface
Solver
Engine

The Problem

How is communication done?

- Through an application program interface (API)
- Think of the API as a specification for methods.
- The methods then interact with an underlying data structure
- In our work it is through the **OSInstance** class and **OSExpression Tree.**

The API


1. instance
2. generic reader class
obj
obj
obj
obj
3. objects created by reader class
This is usually an expression Tree
X[i]
f(x[i])
df(x[i])
3.5*customized/solver-specific reader class (separate from generic reader class)
X[i]
f(x[i])
df(x[i])
Obj'
Obj'
Obj'
Obj'
4. objects taken by solver class
f(x[i])
df(x[i])
5. Solver library or executable
This is usually a calculation method for a function pointer

The API

Libraries are provided with the following features:

- Designed to read and write OSIL
- Written in C++
- Are platform independent
- Can be used in either a tightly or loosely coupled manner

The API

Key Idea: The API is the interface between the solver and the problem instance. We assume the problem instance is in OSiL format.

OSiL: Optimization Services instance Language. This is an XML based format for representing a wide variety of optimization problems.

More on OSiL later in this session.

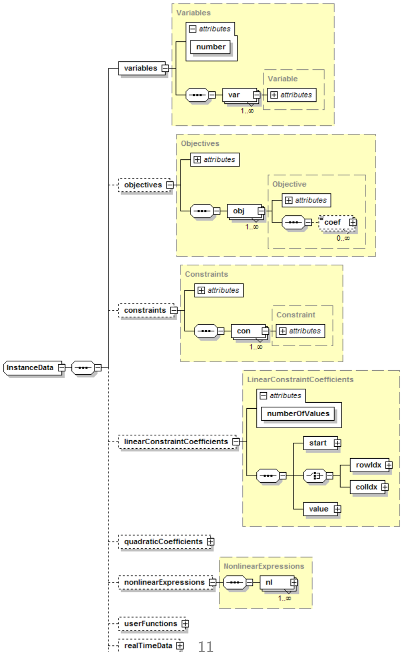
The API

```
<variables number="2">
     <var lb="0" name="x0" type="C"/>
     <var lb="0" name="x1" type="C"/>
</variables>
```

The representation of $\ln(x_0 x_1)$

```
<nl idx="1">
     <ln>
          <times>
               <variable coef="1.0" idx="0"/>
               <variable coef="1.0" idx="1"/>
          </times>
     </ln>
</nl>
```

The API


Variables
attributes
number
variables
var
1..∞
Variable
attributes
Objectives
attributes
objectives
obj
1..∞
Objective
attributes
coef
0..∞
Constraints
attributes
constraints
con
1..∞
Constraint
attributes
InstanceData
LinearConstraintCoefficients
attributes
numberOfValues
linearConstraintCoefficients
start
rowIdx
colIdx
value
quadraticCoefficients
NonlinearExpressions
nonlinearExpressions
nl
1..∞
userFunctions
realTimeData

The API

The job of the API is:

- Read/parse a problem instance (a file or string) in OSiL format
- Validate the problem
- Create an in-memory representation of the problem
- Provide a set of `get()` methods to access logical parts of an optimization instance, e.g. variable lower bounds
- Provide a set of `set()` methods to create/modify a problem instance
- Write problem instances from the in-memory representation

The COIN C++ Libraries

- **OSAgent:** client side library (this one only needed in a distributed environment)
- **OSCommon:** for reading and writing OSiL files and OSInstance objects
- **OSSolver:** solver side library
- **OSUtil:** various utilities such as nl2osil, and mps2osil

The OSCommon Library

Key Classes:

- `OSiLReader`: Takes an *OSiL* string and creates and *OSInstance* object. Xerces *not used* to parse the OSiL. We wrote our own C++ validating (actually stronger) parser.

- OSInstance: This is the in-memory representation. We are done with the linear, integer, and quadratic. Still to do is the nonlinear OSExpression tree. This is the bridge between the optimization instance and the solver internal representation.

- OSiLWriter: Use this class to write a string or file in OSiL format given an in-memory OSInstance object.

The OSCommon Library

The OSInstance class is used to access the problem data or create/modify the problem. For example, accessing a problem for the solver

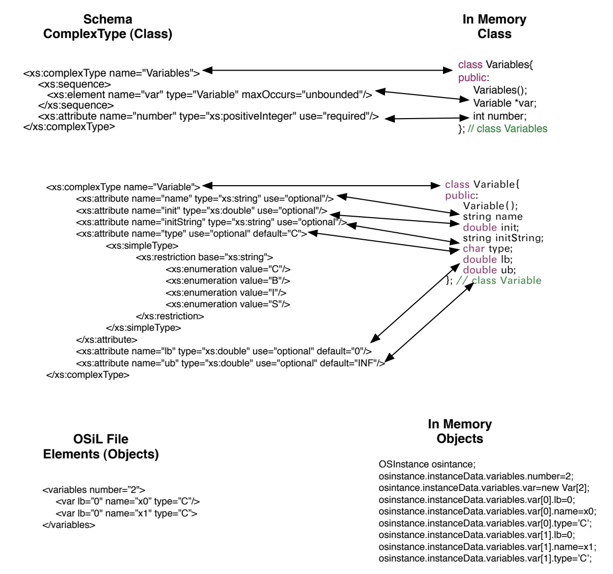
```
m_mdVarLB = osinstance->getVariableLowerBounds();
m_mdVarUB  = osinstance->getVariableUpperBounds();
solver->assignProblem(m_,  m_mdVarLB,  m_mdVarUB,
m_mmdObjDenseCoefValue,  m_mdConLB,  m_mdConUB);
```

or creating a problem

```
instanceData.linearConstraintCoefficients.start.el
= A_colstarts;
instanceData.linearConstraintCoefficients.value.el
= A_vals;
instanceData.linearConstraintCoefficients.rowIdx.el
= A_rownos;
```

Key Idea: It maps to the OSiL Schema.

The OSCommon Library



The OSSolver Library

Viewpoint: you wish to provide a solver service. How do you use the API?

- The API provides a `DefaultSolver` class. It is an abstract class.
- The `DefaultSolver` class has the pure virtual function

```
virtual string solve(string osil, string osol) = 0;
```

- Define your own class that inherits from `DefaultSolver` and implement the `solve()` method.

The OSSolver Library - COIN Solver Example

Here is how the CoinSolver class works.

```
class CoinSolver : public DefaultSolver{
public:
     string solve(string osil, string osol);
};
```

Now implement the CoinSolver solver.

```
string CoinSolver::solve(string osil, string osol)  {
     solverName = osol;
     OSiLReader* osilreader;
     OSInstance* theosinstance = 0;
     OsiSolverInterface* solver = 0;

                              ⋮
```

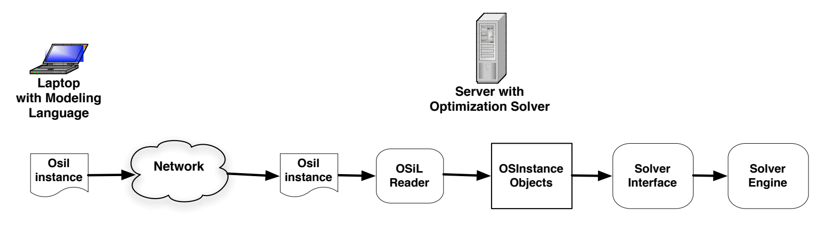
The OSSolver Library - COIN Solver Example

Implementation of `CoinSolver` `solve()` method (continued).

```
if(osol == "glpk")
     solver = new OsiGlpkSolverInterface();
else
     solver = new OsiClpSolverInterface();
osilreader->readOSiL( osil );
theosinstance = osilreader->getOSInstance();
if(!setOSInstance( theosinstance )) return 0;
return optimize();
}
```

Important: the `CoinSolver` class must put the OSInstance object into the COIN data structures such as the `CoinPackedMatrix`

The OSSolver Library – Where are we?



The OSSolver Library – tightly coupled

The OSSolver Library can be used locally or in a tightly coupled environment in one of two ways.

Method 1: Use the `OSServiceUtil` class inside a `main()` method.

```
OSServiceUtil serviceUtil;
string osol = "glpk";
serviceUtil.m_solver = new CoinSolver();
cout << serviceUtil.solve(osil, osol)
```

-OR-

Method 2: Create a solver class directly and use its solve method.

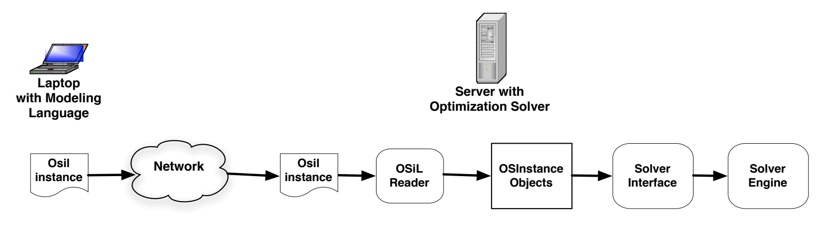
```
LindoSolver lindosolver;
cout << lindosolver.solve( osinstance);
```

The OSSolver Library – Loosely Coupled

Key Idea: Link the OSSolver library with a Web Server such as Apache Axis.

- There is a C++ Apache Axis that can be used in conjunction with the Apache Web server to support Web services.
- Create a Web service on the server machine – for example `OSCoinSolverService` or `OSLindoSolverService`. This service uses the solver OSSolver library service.
- ```
  xsd__string OSSolverService::solve(xsd__string osil,
       xsd__string osol)
  {
       OSServiceUtil serviceUtil;
       serviceUtil.m_solver = new CoinSolver();
       char* osrl = &serviceUtil.solve(osil, osol)[0];
       return osrl;
  }
  ```
  ```

The OSAgent Library – Where we are



The OSAgent Library

Key Idea: Access the solver over a network using Web Services. Use the OSAgent library to do his.

- The API provides a `OShL` class (Optimization Services hookup Language). It is an abstract class.
- The `OShL` class has the pure virtual function

```
virtual string solve(string osil, string osol) = 0;
```

- The `OSSolverAgent` class inherits from `DefaultSolver` and implement the `solve()` method.

The OSAgent Library

What does the `OSSolverAgent` class `solve()` method do?

- `solve()` invokes and object in the WSUtil class and:
- Creates a SOAP envelop containing the optimization instance
- Uses the C socket API to contact a server, send the server the SOAP envelop using HTTP
- Receives the result back from the server

Note: Apache Axis client **is not** necessary. We have written the necessary socket layer software to handle the communication with the server.

The OSAgent Library

Here is a simple illustration of the client using the OSAgent library

```
string osol = "clp";
OSSolverAgent* osagent;
osagent = new
OSSolverAgent("128.135.130.17/axis/OSCoinSolverService");
cout <<  osagent->solve(osil, osol);
```

The OSUtilities Library

Key Idea: you need instances in OSiL format to use the libraries. More on this in the Tuesday, 1:30 PM session. But the following are designed to generate OSiL.

nl2osil: Convert AMPL nl files to OSiL (for linear integer programs)

mps2osil: Convert MPS to OSiL. Not yet written but on the to do list.

Summary

Modeling languages that can generate OSiL:

- AMPL (linear OSiL – use nl2osil.exe)
- OSmL (native linear and nonlinear)
- POAMS (native linear OSiL???)

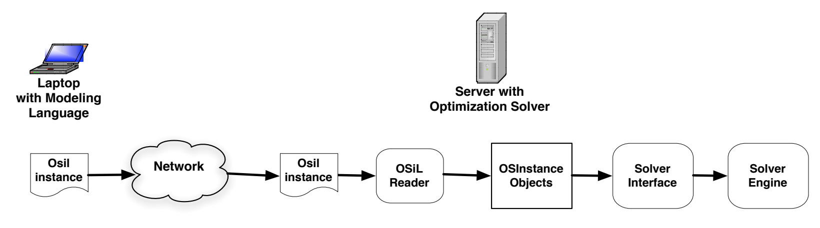
Solvers:

- CLP - through COIN OSI
- FORTMP - LPFML
- GLPK – through COIN OSI
- IMPACT - native support
- KNITRO - using function callbacks
- LINDO – using instruction list format

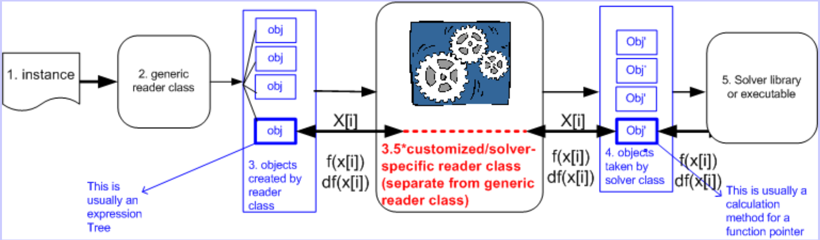
Summary

An ideal world: each solver supports the function

`string solve(string osil, string osol);`



Summary



1. instance
2. generic reader class
obj
obj
obj
obj
3. objects created by reader class
This is usually an expression Tree
X[i]
f(x[i])
df(x[i])
3.5*customized/solver-specific reader class (separate from generic reader class)
X[i]
f(x[i])
df(x[i])
Obj'
Obj'
Obj'
Obj'
4. objects taken by solver class
f(x[i])
df(x[i])
5. Solver library or executable
This is usually a calculation method for a function pointer