

Optimization Services, Web Services, and Excel

Kipp Martin
University of Chicago
Visiting Professor
University of Cincinnati

Other Collaborators: Bob Fourer, Gus Gassmann, Jun Ma, and
Wayne Sheng.

January 12, 2009

Outline

The Optimization Services Project

The Excel Project

Obtaining the Code

Running the Classes

The OSInstance Class

The Web Service Class

The Optimization Services Project

Optimization Services (OS) integrates numerous COIN-OR projects. The OS project provides:

- ▶ A set of XML based standards for representing optimization instances (OSiL), optimization results (OSrL), and optimization solver options (OSoL).
- ▶ A robust API for linear and nonlinear problems.
- ▶ A command line executable OSSolverService for reading problem instances (OSiL format, nl format, MPS format) and calling a solver either locally or on a remote server.
- ▶ Utilities that convert AMPL nl files into the OSiL format and MPS files into the OSiL format.

The Optimization Services Project

OS Continued ...

- ▶ Standards that facilitate the communication between clients and solvers using Web Services.
- ▶ Client side software that is used to create Web Services SOAP packages with OSiL instances and OSoL options and contact a server for the OSrL solution.
- ▶ Server software (reference implementation) that works with Apache Tomcat.

The Optimization Services Project

OS available as COIN-OR project that builds and runs on:

- ▶ Windows using Microsoft Visual Studio (project files available)
- ▶ Windows using MSYS and Microsoft cl
- ▶ Windows using MINGW (MSYS + gcc)
- ▶ Windows using Cygwin and gcc
- ▶ Windows using Cygwin and cl
- ▶ Numerous flavors of GNU/Linux (32 and 64 bit)
- ▶ Solaris
- ▶ Mac OS X (both Intel and Power PC)
- ▶ IBM AIX

For **unit testing platforms** see:

[https:](https://projects.coin-or.org/TestTools/wiki/NightlyBuildInAction)

[//projects.coin-or.org/TestTools/wiki/NightlyBuildInAction](https://projects.coin-or.org/TestTools/wiki/NightlyBuildInAction)

The Excel Project

Motivation: originally from teaching a masters level VBA course.

- ▶ Have students solve large problems – get away from typical toy problems
- ▶ Business school theorem – if it is in Excel it is applied
- ▶ Make licensing issues go away – easy to do with COIN-OR
- ▶ Make problem size issues go away – easy to do with COIN-OR
- ▶ Teach VBA – students use VBA to create the model instance
- ▶ Teach Web Services – show the ease and utility of Web Services in Excel/VBA (also XML)

Decided to include as part of the OS project for individuals want to use Excel with COIN-OR solvers.

The Excel Project

Important Disclaimer! Not meant to compete with the Frontline Systems Solver product.

Full Disclosure! Knowledge of VBA *absolutely required* to use this software!!!

My Objective: if you know a little VBA (arrays, loops, if-then logic), be able to build and solve big problems with *no license and size worries*.

The Excel Project

As Aside: COIN-OR in the classroom. I have experimented with three levels at the College of Business at the University of Cincinnati.

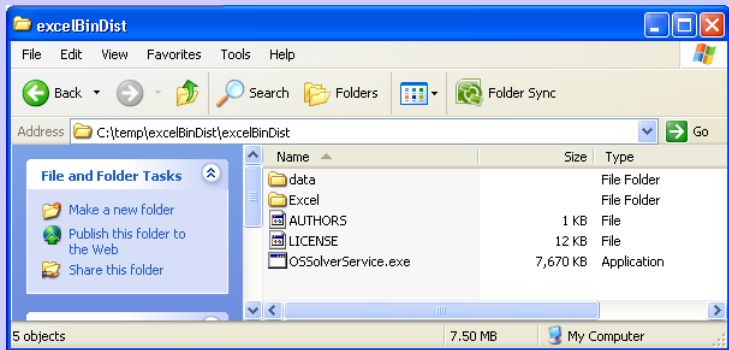
- ▶ Provide Visual Studio project files. Have students build models using the OS API and call the COIN-OR solvers. Students need to compile their own code and link with OS lib (among others). Knowledge of C++ a must.
- ▶ Provide VBA class files and have students build models in VBA and call COIN-OR solvers. Only need to know some VBA. This experiment motivated this talk.
- ▶ Even less programming – Build GAMS models and call COIN-OR solvers (locally or remotely with GAMSlinks). Make use of GAMS scripts for column, cut generation, etc.

Obtaining the Code

A zip of all the files necessary is at:

http:

`//www.coin-or.org/download/binary/OS/excelBinDist.zip`



Obtaining the Code

The Excel files:

- ▶ `clsOSInstance.cls` – user defined class
- ▶ `parIncVBA.xlsm` – example model
- ▶ `clsWS_OSSolverServiceServic.cls` – user defined class
- ▶ `OSWebServiceGUI.xlsm` – example of remote call

Building and Solving a Model

The big picture:

- ▶ The **OSSolverService.exe** that is part of the download is the executable that will optimize the model formulation. You call this executable from inside Excel. It contains the following solvers:
 - ▶ COIN-OR **Bonmin** – an integer + nonlinear solver
 - ▶ COIN-OR **Cbc** – an integer programming solver based on branch-and-cut
 - ▶ COIN-OR **Clp** – a linear programming solver
 - ▶ COIN-OR **Couenne** – an integer + nonlinear solver + global
 - ▶ COIN-OR **DyLP** – a linear programming solver
 - ▶ COIN-OR **Ipopt** – a nonlinear solver (assumes continuous variables)
 - ▶ COIN-OR **SYMPHONY** – a linear integer programming solver

Building and Solving a Model

The big picture (continued):

- ▶ The **OSSolverService.exe** takes a model instance in the OSiL (Optimization Services input Language) XML format.
- ▶ You import **clsOSInstance.cls** from the download into your VBA project. This class has methods that use the XML DOM (Document Object Model) to create a string in the OSiL XML format.
- ▶ You do not need to worry about the DOM – you read data from ranges in the spreadsheet that contain the data and call methods in the **clsOSInstance.cls** that take your data and put it into the DOM.

The OSiL XML string is written to file which is then read and executed by the **OSSolverService.exe** executable

Building and Solving a Model

The first step is open up a workbook and then import **clsOSInstance.cls** into VBA project. Then create a blank module where you put your code.

Next create an **instance object**.

```
Dim osinstance As New clsOSInstance
```

Use the **osinstance** to:

- ▶ define the variables
- ▶ define the objective function
- ▶ define the constraints
- ▶ define the constraint matrix

Building and Solving a Model

In the download there is a workbook **parIncVBA.xlsm** that illustrates this process. The worksheet **parInc** has the data and when the model is solved the result gets written to the **result** worksheet. This workbook illustrates solving the problem:

$$\begin{aligned} \text{MAX} \quad & 10 * X1 + 9 * X2 \\ & .7 * X1 + X2 \leq 630 \\ & .5 * X1 + (5/6) * X2 \leq 600 \\ & X1 + (2/3) * X2 \leq 708 \\ & .1 * X1 + .25 * X2 \leq 135 \\ & X1, X2 \geq 0 \end{aligned}$$

Building and Solving a Model

Variables: the variables are defined in OSiL as

```
<variables numberOfVariables="2">  
  <var name="x0" lb="0" />  
  <var name="x1" lb="0" />  
</variables>
```

to define these in VBA

```
osinstance.numVar = amatrix.Columns.count  
For I = 1 To osinstance.numVar  
  varLBArray(I) = 0  
  varTypeArray(I) = "C"  
  varUBArray(I) = osinstance.PosInf  
Next I  
Call osinstance.OSgenerateVariables(varLBArray,  
  varUBArray, varTypeArray)
```

Building and Solving a Model

Objective Function: the objective function is defined in OSiL as

```
<obj maxOrMin="max" numberOfObjCoef="2">  
  <coef idx="0">10</coef>  
  <coef idx="1">9</coef>  
</obj>
```

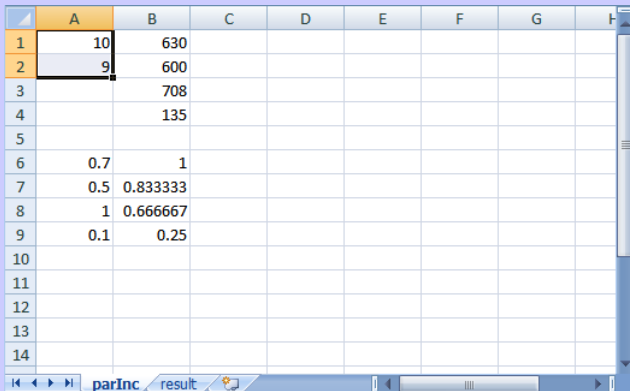
to define these in VBA

```
osinstance.isMax = True  
For I = 1 To osinstance.numVar  
  objCoefArray(I) = objCoef.Cells(I, 1)  
Next I  
Call osinstance.OSgenerateObjective(objCoefArray)
```

No knowledge of XML is required!!!

Building and Solving a Model

Objective Function (continued): the objective function coefficients for **objCoefArray(I)** come from the range **objCoef**.



	A	B	C	D	E	F	G	H
1	10	630						
2	9	600						
3		708						
4		135						
5								
6	0.7	1						
7	0.5	0.833333						
8	1	0.666667						
9	0.1	0.25						
10								
11								
12								
13								
14								

Building and Solving a Model

Constraints: the constraints are defined in OSiL as

```
<constraints numberOfConstraints="4">
  <con ub="630" />
  <con ub="600"/>
  <con ub="708"/>
  <con ub="135"/>
</constraints>
```

to define these in VBA

```
osinstance.numCon = amatrix.Rows.count
For I = 1 To osinstance.numCon
  conUBArray(I) = rhs.Cells(I, 1)
  conLBArray(I) = -osinstance.PosInf
Next I
Call osinstance.OSgenerateConstraints(conLBArray,
  conUBArray)
```

Building and Solving a Model

Constraints (continued): the right-hand-side values for `conUBArray(I)` come from the range `rhs`.

	A	B	C	D	E	F	G	H
1	10	630						
2	9	600						
3		708						
4		135						
5								
6	0.7	1						
7	0.5	0.833333						
8	1	0.666667						
9	0.1	0.25						
10								
11								
12								
13								
14								

Building and Solving a Model

Constraint Matrix Coefficients: the constraint matrix coefficients (sparse storage) are defined in OSiL as

```
<linearConstraintCoefficients numberOfValues="8">
  <start>
    <el>0</el><el>4</el><el>8</el>
  </start>
  <rowIdx>
    <el>0</el><el>1</el><el>2</el>
    <el>3</el><el>0</el><el>1</el>
    <el>2</el><el>3</el>
  </rowIdx>
  <value>
    <el>0.7</el><el>.5</el><el>1.</el>
    <el>.1</el><el>1.0</el><el>0.8333</el>
    <el>0.6667</el><el>0.25</el>
  </value>
</linearConstraintCoefficients>
```

Building and Solving a Model

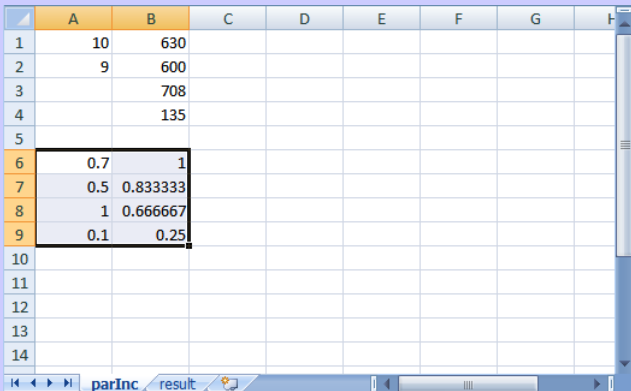
Constraint Matrix Coefficients (continued): the corresponding VBA code is

```
For J = 1 To oinstance.numVar
    startsArray(J + 1) = startsArray(J) +
        amatrix.Rows.count
    For I = 1 To oinstance.numCon
        indexesArray(startsArray(J) + I - 1) = I
        valuesArray(startsArray(J) + I - 1) =
            amatrix.Cells(I, J)
    Next I
Next J
Call oinstance.OSgenerateLinearConstraintMatrix(
    startsArray, indexesArray, valuesArray)
```

Example problem happens to be dense, but the class **clsOSInstance.cls** is based on a sparse matrix storage system.

Building and Solving a Model

Constraint Matrix Coefficients (continued): coefficients for the values array `valuesArray(I)` come from the range `amatrix`.



	A	B	C	D	E	F	G	H
1	10	630						
2	9	600						
3		708						
4		135						
5								
6	0.7	1						
7	0.5	0.833333						
8	1	0.666667						
9	0.1	0.25						
10								
11								
12								
13								
14								

Building and Solving a Model

Example of sparse input array. Model is built by row, each row has variable indexes.

	A	B	C	D	E	F	G	H	I
1247	4	1	2	24	51				
1248	4	3	5	13	22				
1249	4	1	13	17	22				
1250	4	1	14	35	38				
1251	4	2	28	40	70				
1252	4	15	34	36	72				
1253	4	13	22	35	49				
1254	4	9	35	38	49				
1255	4	1	4	9	13				
1256	4	10	38	40	61				
1257	4	1	6	8	9				
1258	4	9	35	44	62				
1259	4	1	5	58	67				
1260	4	11	13	22	25				
1261	4	1	3	21	38				
1262	4	9	34	35	40				
1263	4	1	10	46	55				
1264	4	1	10	17	28				
1265	4	27	36	41	58				
1266	4	5	9	23	29				
1267	4	25	26	29	35				
1268	4	1	10	13	46				

Building and Solving a Model

Solving the model: there are two solve methods in class `clsOSInstance.cls`. One is a synchronous solve and the other is an asynchronous solve. Both of these methods take two arguments:

- ▶ Argument 1: the location of the OSiL file that was created
- ▶ Argument 2: the location of the OSrL (result file) that is written by **OSSolverService.exe**

To write the result to a spreadsheet call the procedure:

```
Call osinstance.OSWriteResult("result", "C:/result.osrl")
```

you get the result on the following page.

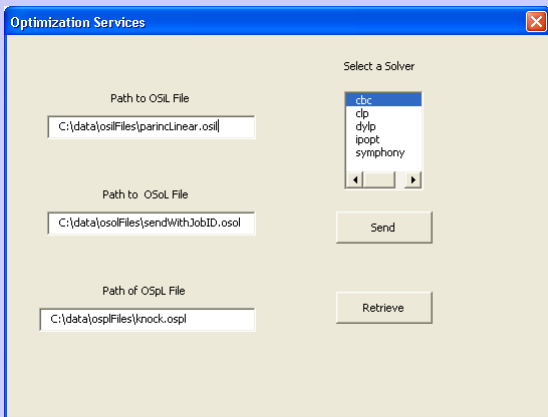
Building and Solving a Model

The solution result:

	A	B	C	D	E	F	G	H
1	Optimal Value	Variable	Value	Reduced Cost		Constraint	Dual Value	
2	7668	0	540	0		0	4.375	
3		1	252	0		1	0	
4						2	6.9375	
5						3	0	
6								
7								
8								
9								
10								
11								
12								
13								
14								

The Web Service Class

Call a Web Service: the class `clsws_OSSolverService.cls` allows the user to send the instance to a *remote solver*. This is illustrated in the workbook **OSWebServiceGUI.xlsm**.



The Web Service Class

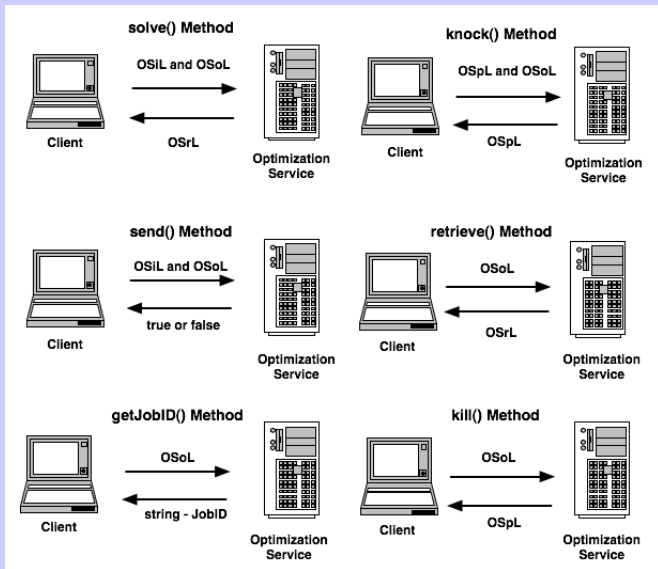
How should a client communicate with a solver server over the network?

- ▶ The client wants to communicate an instance to the solver server.
- ▶ The client want to know if a job is done.
- ▶ The client may wish to “kill” a job.
- ▶ The client wants to retrieve the result when the problem is optimized.

The OShL (Optimization Services hookup Language) is a protocol that uses WSDL to specify six key methods for communication.

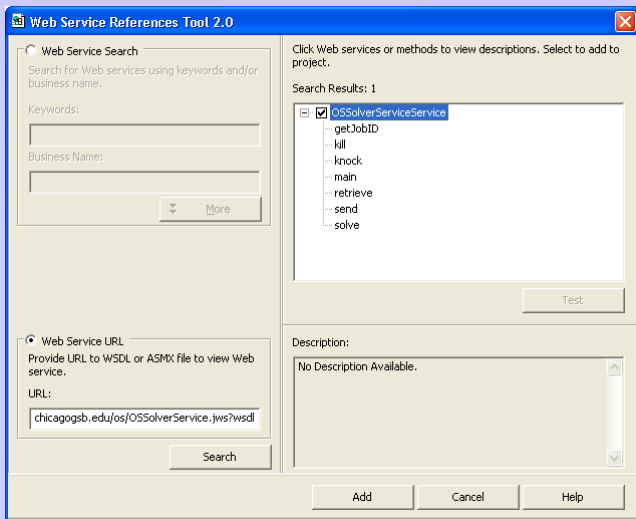
The Web Service Class

The six OShL methods:



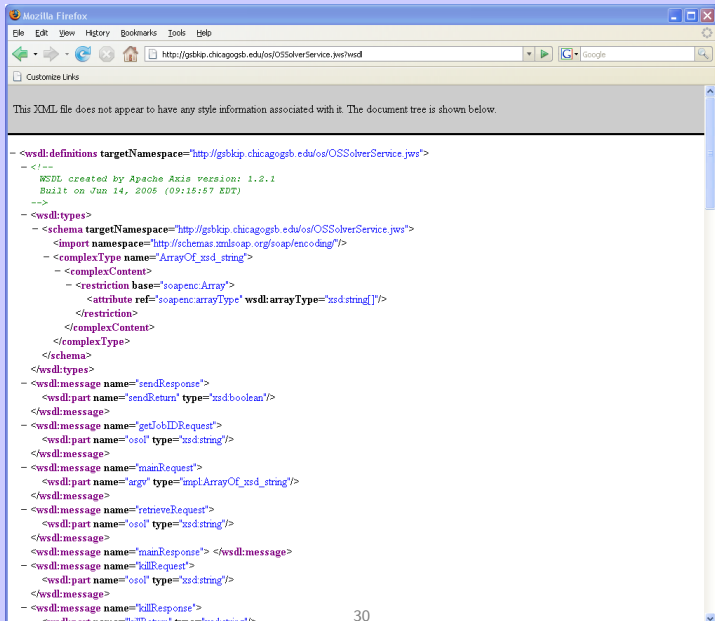
The Web Service Class

Building a Web Service in Excel is trivial.



The Web Service Class

The WSDL on the server.



```
<?xml version="1.0" encoding="UTF-8" style="display:none" />
<wdd:definitions targetNamespace="http://gsbkcp.chicagogsb.edu/os/OSSolverService.jws">
  <!--
    WSDL created by Apache Axis version: 1.2.1
    Built on Jun 14, 2005 (09:15:57 EDT)
  -->
  <wdd:types>
    <schema targetNamespace="http://gsbkcp.chicagogsb.edu/os/OSSolverService.jws">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding"/>
      <complexType name="ArrayOf_xsd_string">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]" />
          </restriction>
        </complexContent>
      </complexType>
    </schema>
  </wdd:types>
  <wdd:message name="sendResponse">
    <wdd:part name="sendReturn" type="xsd:boolean"/>
  </wdd:message>
  <wdd:message name="getJobIDRequest">
    <wdd:part name="osol" type="xsd:string"/>
  </wdd:message>
  <wdd:message name="mainRequest">
    <wdd:part name="argv" type="impl:ArrayOf_xsd_string"/>
  </wdd:message>
  <wdd:message name="retrieveRequest">
    <wdd:part name="osol" type="xsd:string"/>
  </wdd:message>
  <wdd:message name="mainResponse"> </wdd:message>
  <wdd:message name="killRequest">
    <wdd:part name="osol" type="xsd:string"/>
  </wdd:message>
  <wdd:message name="killResponse">
    <wdd:part name="osol" type="xsd:string"/>
  </wdd:message>
</wdd:definitions>
```

The Web Service Class

The VBA code is trivial, for example, retrieving the result of the solve from the Web Server is as easy as:

```
Dim ws As New clsws_OSSolverServiceService
    osr1 = Retrieve(osol, ws)
```

where this **Retrieve** method calls underlying function

```
sc_OSSolverServiceService.Retrieve(str_osol)
```

created by the Excel Web Services tool.