An Open Interface for Hooking Solvers to Modeling Systems

Robert Fourer and Jun Ma
Northwestern University

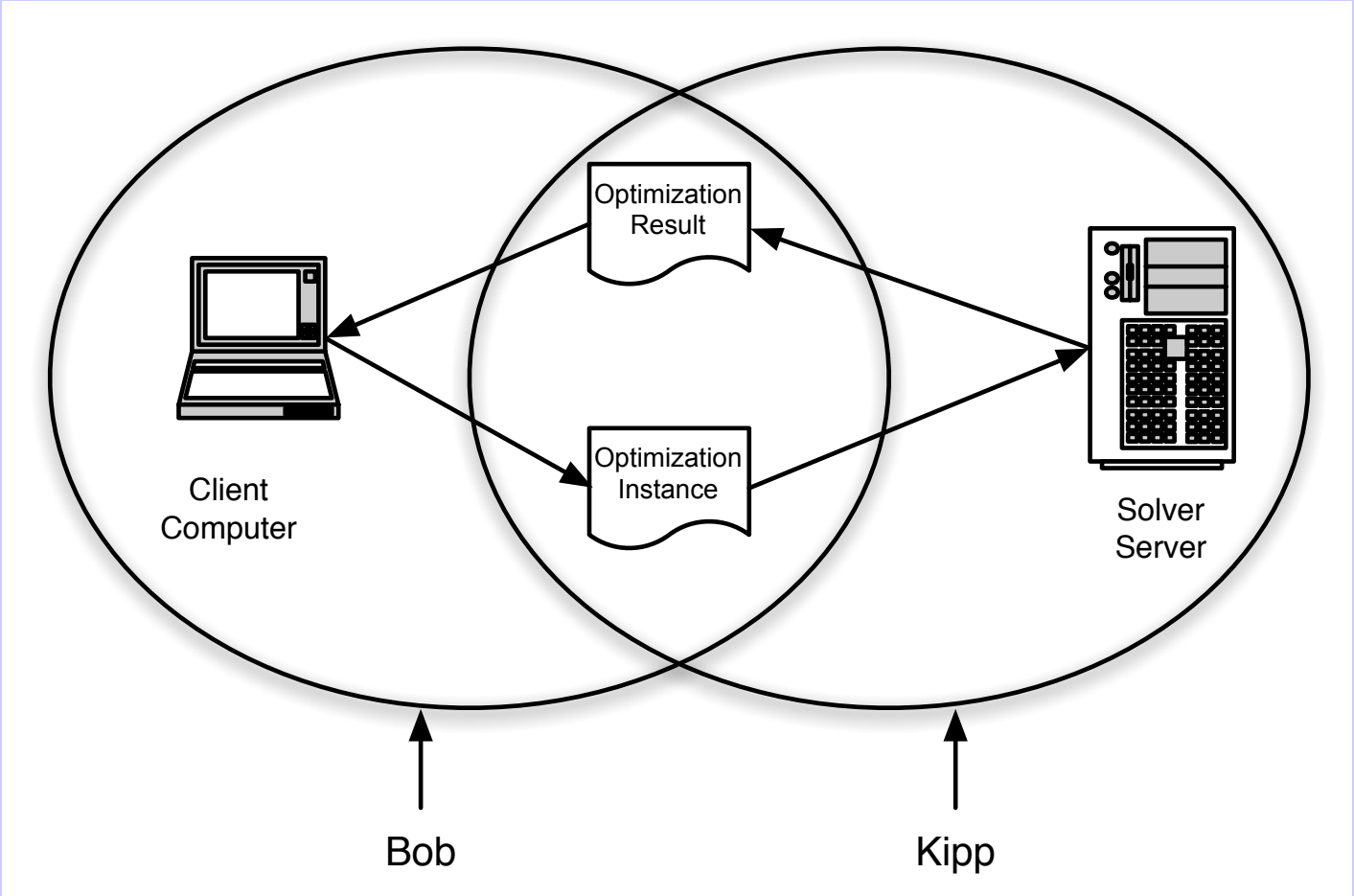
Kipp Martin
University of Chicago

July 19, 2006

Outline

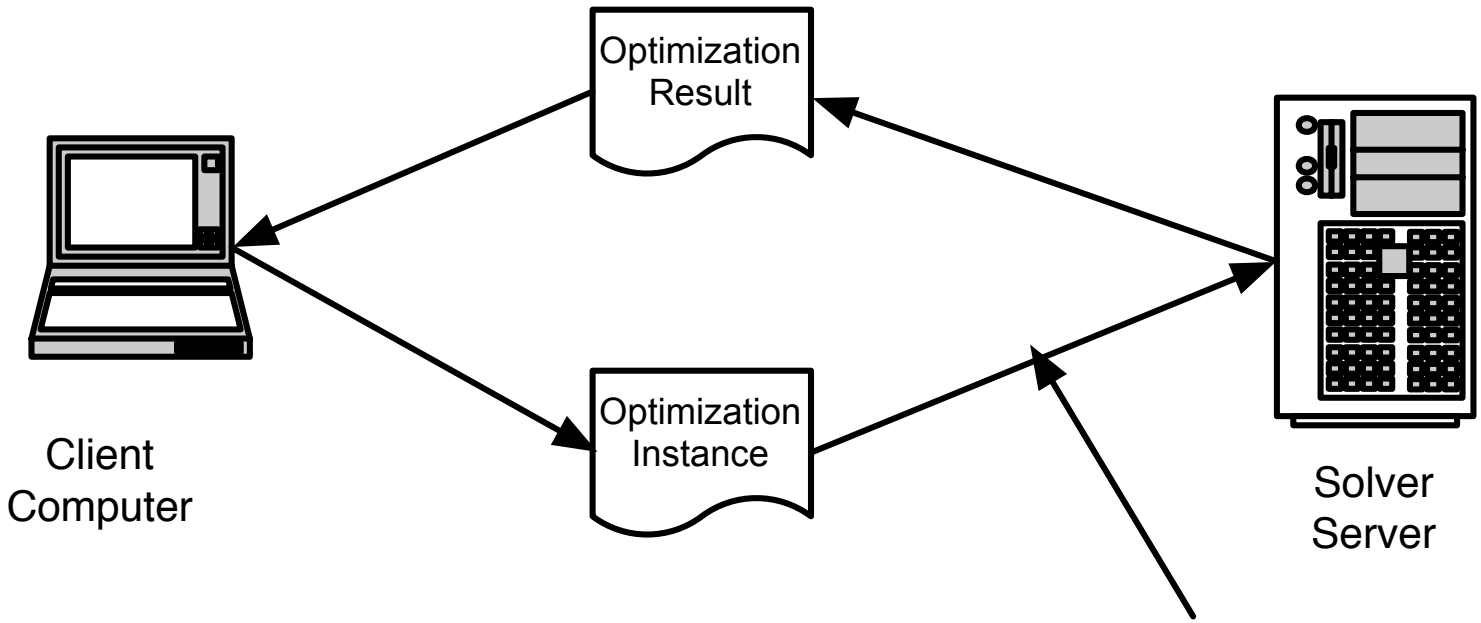
Motivation

A simple scenario:



Take advantage of a faster machine (the server), code not on the client, a better license deal, open source software, etc. Maintaining code on a single machine is just easier.

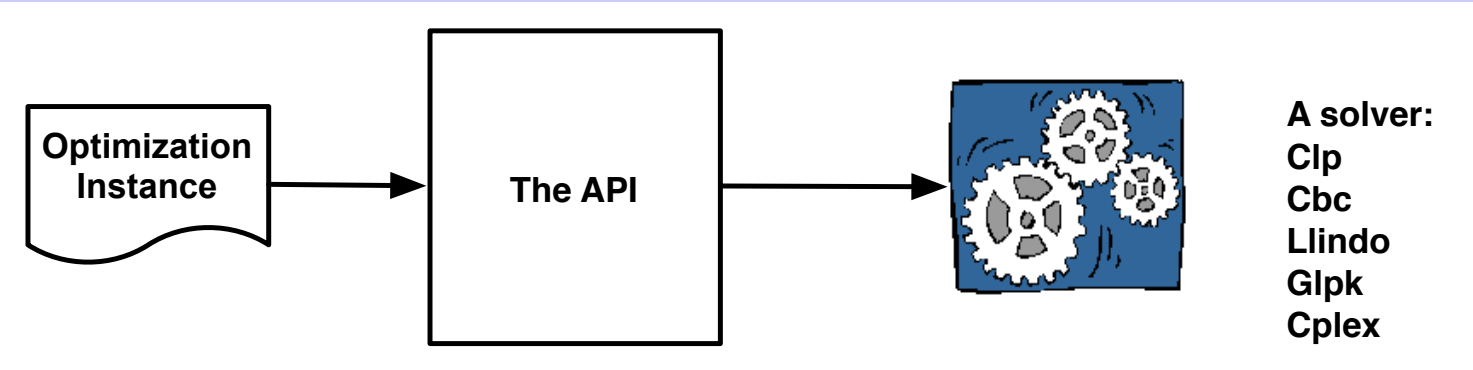
Motivation



BIG PROBLEM!

Unless the solver can directly read the optimization instance an API is needed.

Motivation



The COIN OSI is an example of such an API.

```
m_CoinPackedMatrix = new CoinPackedMatrix(...);
m_OsiSolver->loadProblem(*m_CoinPackedMatrix, ...)
m_OsiSolver->branchAndBound();
m_OsiSolver->initialSolve();
```

Motivation

KEY OBSERVATIONS:

1. The current COIN OSI is **BOTH** a solver interface **AND** a problem instance interface.
2. The current COIN OSI does not handle nonlinear problems.

OUR OBJECTIVE: A robust representation and **instance interface** for very general optimization problems but **NOT** a **solver interface**.

A solver interface is not really practical or necessary! Instead use an instance interface and an option interface.

Motivation

From Coin-discuss:

```
On Sat, 8 Jul 2006, Matthew Galati wrote:

 Hi,

Several of the LP solvers in Osi have interior point
methods. Can this be added as an option to solve
with interior vs simplex? I guess the OSI2 design -
where model and algorithm are split would fix this...
but, is OSI2 still going to happen?
and, if there is no real timeline for OSI2,
can this be added to OSI?

Matt
```

Also, degeneracy issue raised by Ojas Parekh in PICO-CLP talk.

Motivation

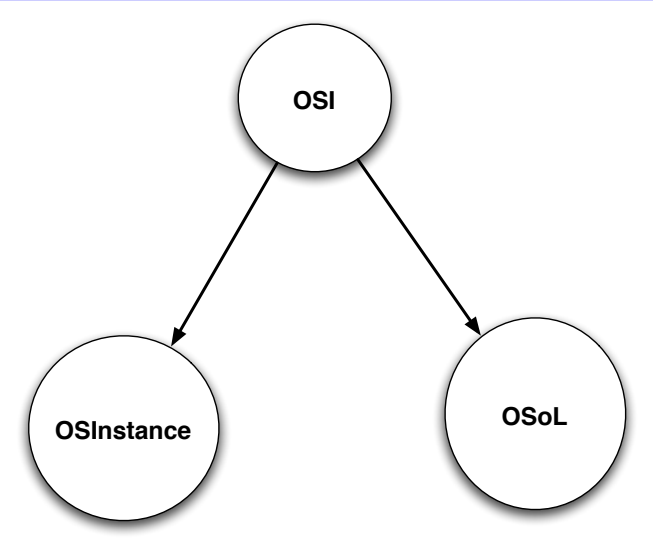
An ideal world – no solver interface!

If the instance interface and option interface are robust enough then we have:

```
load(problem_instance);
solve(options_list);
```

It is up to the solver to implement the problem_instance and interpret what is in the options_list

Motivation




```
<osol>
  <general>
      <optimization>
          <other name="solverType">LP</other>
          <other name="solverAlg">barrier</other>
          <other name="degenLevel">3</other>
      </optimization>
   </general>
</osol>
```

Motivation

```
OSiLReader *osilReader;
OSoLReader  *osolReader;
OsiSolverInterface  *coinSolver;


osilReader = new OSiLReader();
osolReader = new OSoLReader();
coinSolver = new OsiCbcSolverInterface();


coinSolver->osinstance = osilReader->readOSiL(osilString);
coinSolver->osoption = osolReader->readOSoL(osolString);
coinSolver->solve();
```

OSiL – A File-Based Representation

$$\text{Minimize} \quad (1 - x_0)^2 + 100(x_1 - x_0^2)^2 + 9x_1$$

$$\text{Subject to} \quad x_0 + 10x_0^2 + 11x_1^2 + 3x_0x_1 \leq 25$$

$$\ln(x_0x_1) + 7x_0 + 5x_1 \geq 10$$

$$x_0, x_1 \geq 0$$

OS Protocols: OSiL

The variables: $x_0, x_1 \geq 0$

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

The objective function: $\text{minimize} \quad 9x_1$

```
<objectives number="1">
     <obj maxOrMin="min" name="minCost">]
          <coef idx="1">9</coef>
     </obj>
</objectives>
```

OS Protocols: OSiL

The linear terms are stored using a sparse storage scheme

$$\begin{aligned} x_0 + 10x_0^2 + 11x_1^2 + 3x_0x_1 &\leq 25 \\ 7x_0 + 5x_1 + \ln(x_0x_1) + &\geq 10 \end{aligned}$$

```
<linearConstraintCoefficients>
     <start>
          <el>0</el><el>2</el><el>3</el>
     </start>
     <rowIdx>
          <el>0</el><el>1</el><el>1</el>
     </rowIdx>
     <value>
          <el>1.0</el><el>7.0</el><el>5.0</el>
     </value>
</linearConstraintCoefficients>
```

OS Protocols: OSiL

Representing quadratic and general nonlinear terms

$$\begin{aligned} x_0 + 10x_0^2 + 11x_1^2 + 3x_0x_1 &\leq 25 \\ 7x_0 + 5x_1 + \ln(x_0x_1) + &\geq 10 \end{aligned}$$

```
<quadraticCoefficients numberOfQuadraticTerms="3">
     <qTerm idx="0" idxOne="0" idxTwo="0" coef="10"/>
     <qTerm idx="0" idxOne="1" idxTwo="1" coef="11"/>
     <qTerm idx="0" idxOne="0" idxTwo="1" coef="3"/>
</quadraticCoefficients>

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

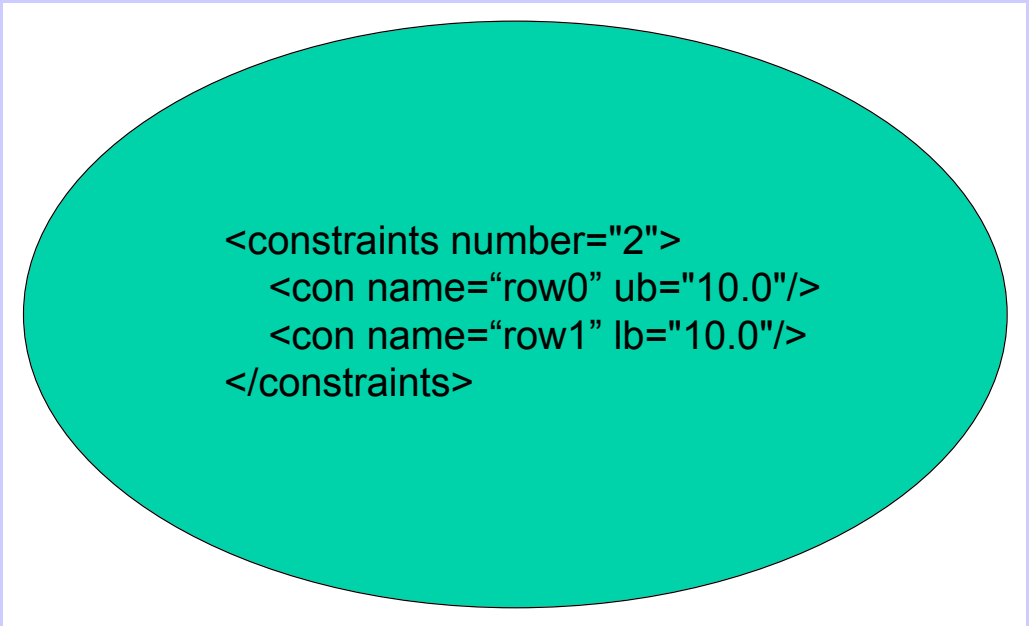
OS Protocols: OSiL

Key idea a **schema**. How do we know how to write proper OSiL? Similar to the concept of a class in object orient programming. Critical for parsing!

Schema $\Longleftrightarrow$ Class

XML File $\Longleftrightarrow$ Object

We need a schema to define the OSiL instance language.



OS Protocols: OSiL

Schema a Constraints and Con Class

```
<xs:complexType name="constraints">
        <xs:sequence>
                <xs:element name="con" type="con" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="number" type="xs:nonNegativeInteger" use="required"/>
</xs:complexType>
<xs:complexType name="con">
        <xs:attribute name="name" type="xs:string" use="optional"/>
        <xs:attribute name="lb" type="xs:double" use="optional" default="-INF"/>
        <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>
        <xs:attribute name="mult" type="xs:positiveInteger" use="optional" default="1"/>
</xs:complexType>
```

The OSiL Schema

The schema is used to **validate** the XML document. Think of validation as an error check.

The schema defines an XML vocabulary, language, or dialect. Examples include:

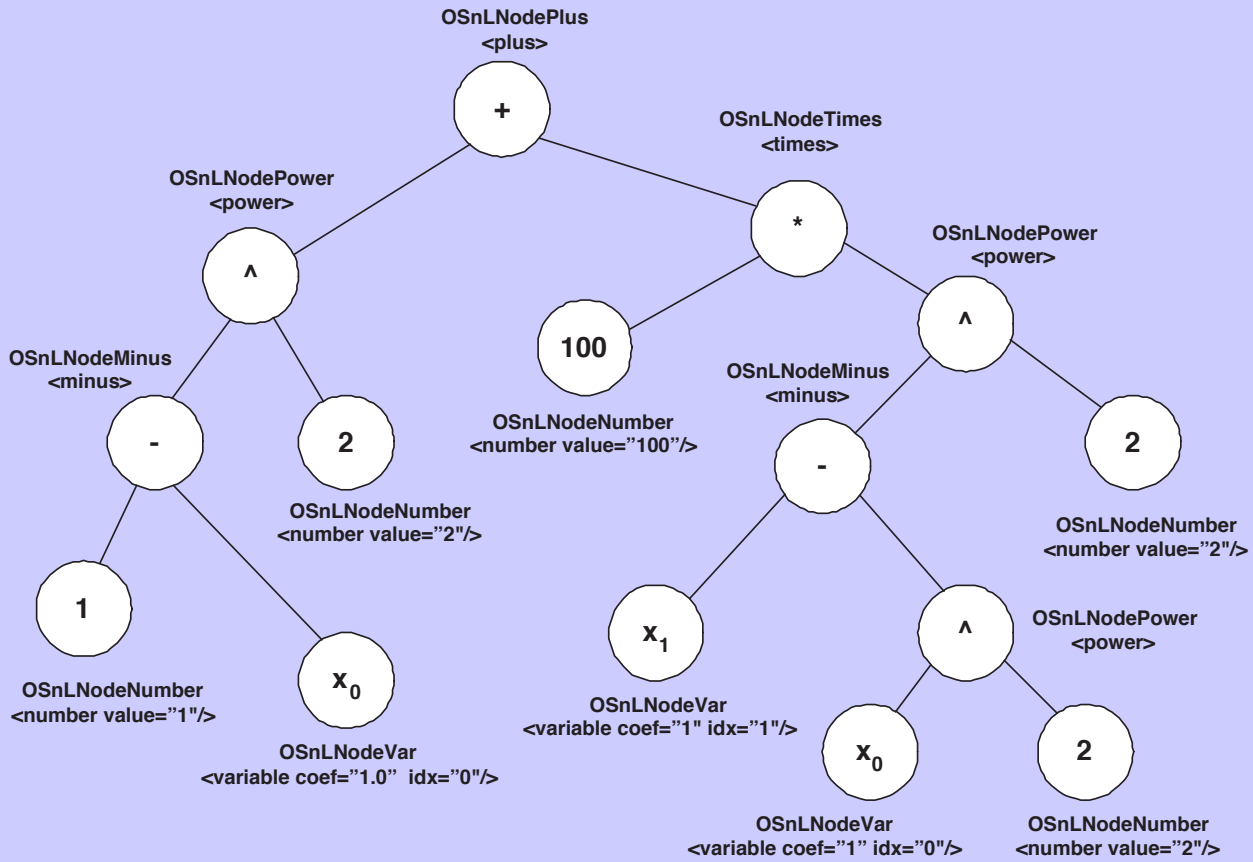
- XHTML – the markup language for Web documents
- FpML– Financial products Markup Language
- WordProcessingML and SpreadsheetML for Microsoft Office
- XBRL– eXtensible Business Reporting Language
- MathML– a format for representing math on Web pages
- AnatML– Anatomical Markup Language
- RSS – Really Simple Syndication for news feeds

OSiL – the markup language for optimization instances

The OSiL Schema

$$(1 - x_0)^2 + 100(x_1 - x_0^2)^2$$

How do we validate this? Designing the schema is a huge problem!



The OSiL Schema

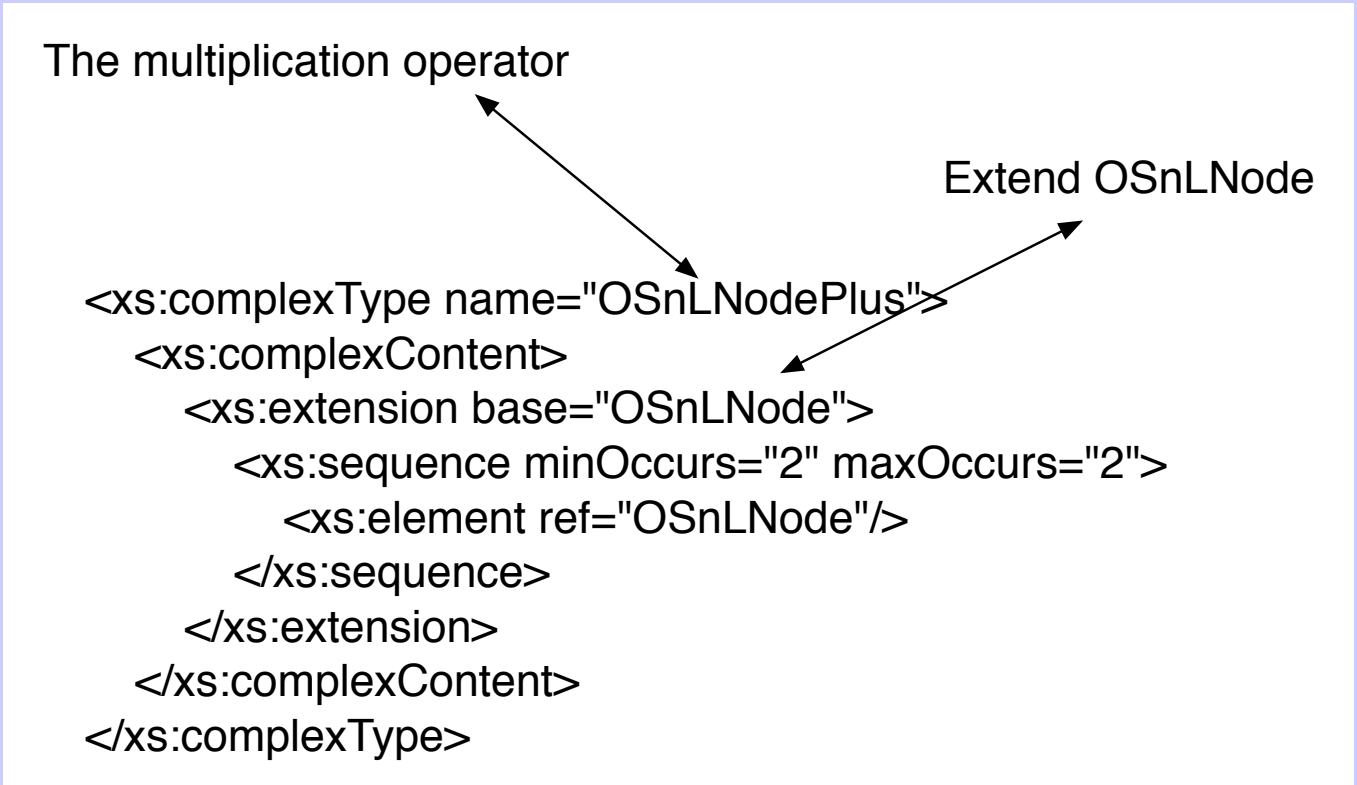
Design Goal: *represent a comprehensive collection of optimization problems while keeping parsing relatively simple.* Not easy!!!

- For purposes of validation, any schema needs an explicit description of the children allowed in a `<operator>` element
- It is clearly inefficient to list every possible nonlinear operator or nonlinear function allowed as a child element. If there are n allowable nonlinear elements (functions and operators), listing every potential child element, of every potential nonlinear element, leads to $O(n^2)$ possible combinations.
- This is also a problem when doing function and gradient evaluations, etc. a real PAIN with numerous operators and operands.
- We avoid this by having EVERY nonlinear node an OSnLNode instance.

The OSiL Schema

Solution: *Use objected oriented features of the XML Schema standard.*

```
<xs:complexType name="OSnLNode" mixed="false"/>
<xs:element name="OSnLNode" type="OSnLNode"
        abstract="true">
```



In-Memory Object: OSInstance

- The code for implementing this is written in C++.
- The C++ code "mimics" the XML schema
- In C++ there is an abstract class **OSnLNode** with pure virtual functions for function and gradient calculation.
- There are operator classes such as **OSnLNodePlus** that inherit from **OSnLNode** and *do the right thing* using polymorphism.

The C++ code “mimics” the XML schema

Schema:

```
<xs:complexType name="Constraints">
<xs:sequence>
<xs:element name="con" type="Constraint"/>
</xs:sequence>
<xs:attribute name="number" type="xs:nonNegativeInteger"/>
</xs:complexType>
```

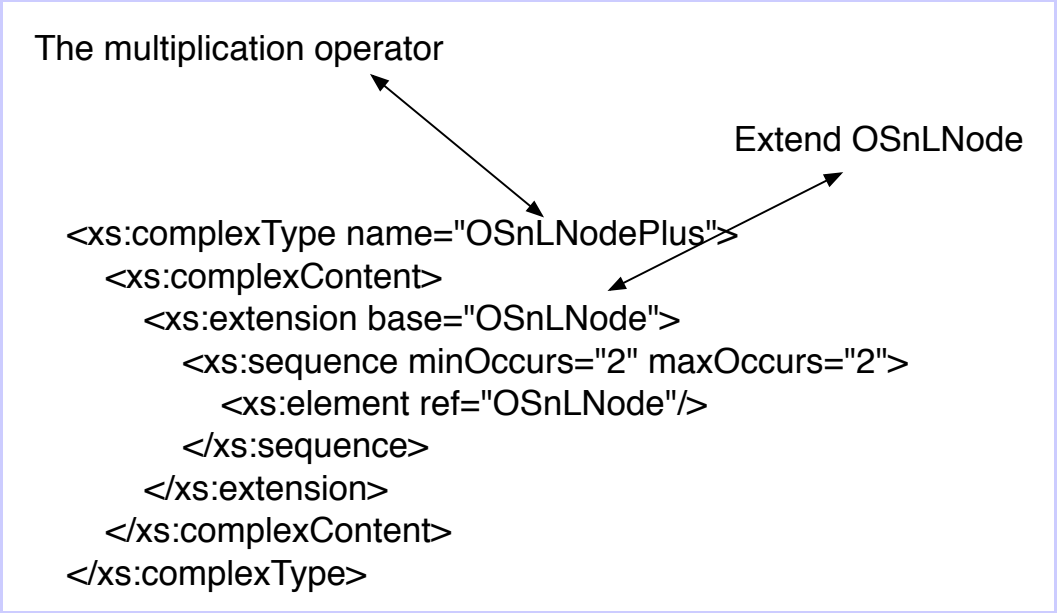
CPP Code:

```
class Constraints{
public:
     Constraints();
     ~Constraints();
     int numberOfConstraints;
     Constraint **con;
}; //class Constraints
```

The Mapping Rules

- Each XML schema complexType corresponds to a class in `OSInstance`. Elements in the actual XML file then correspond to objects in the `OSInstance` class.
- An attribute or element used in the definition of a `complexType` is a member of the corresponding in-memory class; moreover the type of the attribute or element matches the type of the member.
- A schema sequence corresponds to an array. For example, the complexType `Constraints` has a sequence of <con> elements that are of type `Constraint`.

In-Memory Object: OSInstance




```
OSnLNodePlus::OSnLNodePlus()
{
      snodeName = "plus";
      inumberOfChildren = 2;
      m_mChildren = new OSnLNode*[2];
      m_mChildren[ 0] = NULL;
      m_mChildren[ 1] = NULL;
      inodeInt = 1001;
}//end OSnLNodePlus
```

The OSnLNode Class

The `OSnLNode` class mimics the complexType `OSnLNode` in the schema. It is an **abstract class** with virtual functions,

```
virtual double calculateFunction(double *x) = 0;
```

Here is the implementation of the virtual function in the `OSnLNodePlus` class that is derived from the `OSnLNode` class.

```
double OSnLNodePlus::calculateFunction(double *x){
   m_dFunctionValue = m_mChildren[0]->calculateFunction(x)
   + m_mChildren[1]->calculateFunction(x);
   return m_dFunctionValue;
}// end OSnLNodePlus::calculate
```

The OSInstance API

The OSInstance is then 1) a data structure (including a nonlinear expression tree), 2) a set of `get()` methods, and 3) a set of `set()` methods.

Some `get()` methods:

- get instruction lists in postfix or prefix
- get a text version of the model in infix
- get function and gradient evaluations
- get information about constraints, variables, objective function, the A matrix, etc.
- get the root node of the OSExpression tree

Using the OSInstance API

How can a solver use the API:

- the solver can work directly with the OSInstance data structure
- the solver can use the $\mathrm{get}()$ methods to convert the OSInstance structure into its own data structure
- the solver can use OSInstance to perform function and gradient calculations.

Work Directly with OSInstance

Scatter column 0 of the A matrix into a dense vector:

```
OSiLReader *osilreader ;
osilreader = new OSiLReader();
OSInstance *osinstance;
osinstance = new OSInstance();
osinstance = osilreader->readOSiL( osil);
double *aColSparse;
aColSparse = osinstance->instanceData->linearConstraintCoefficients->value->el;
int *rowIdx;
rowIdx = osinstance->instanceData->linearConstraintCoefficients->rowIdx->el;
int *start;
start = osinstance->instanceData->linearConstraintCoefficients->start->el;
int numConstraints;
numConstraints = osinstance->instanceData->constraints->numberOfConstraints;
double *aColDense = new double[ numConstraints ];
for(int i = start[0]; i < start[ 1]; i++){
     aColDense[ rowIdx[ i] ] = aColSparse[ i];
}
```

Convert an OSInstance into Solver Data Structure

CoinSolver – take an OSInstance object and create an instance using the OSI.

```
osinstance->getVariableUpperBounds();
osinstance->getConstraintLowerBounds();
```

LindoSolver – take an OSInstance object and create an instance using the Lindo API.

```
allExpTrees = osinstance->getAllNonlinearExpressionTrees();
for(posTree = allExpTrees.begin(); posTree != allExpTrees.end();
++posTree){
postFixVec = posTree->second->getPostfixFromExpressionTree();
```

Use OSInstance for Callback Functions

Use the OSInstance object to:

Provide function evaluations.

Provide gradient evaluations (AD)

The COIN-OR OS Project