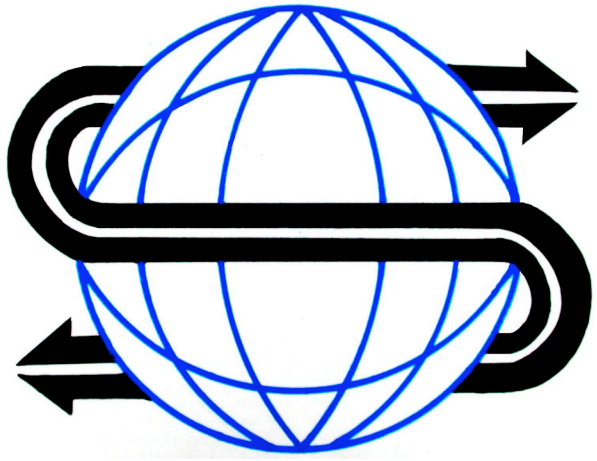


Optimization Services Instance Language (OSiL)

Robert Fourer
Jun Ma
Northwestern University
Kipp Martin
University of Chicago

Kipp Martin
University of Chicago
kipp.martin@chicagogsb.edu

Outline

1. Background and motivation for an instance standard

2. Why use XML for the instance standard

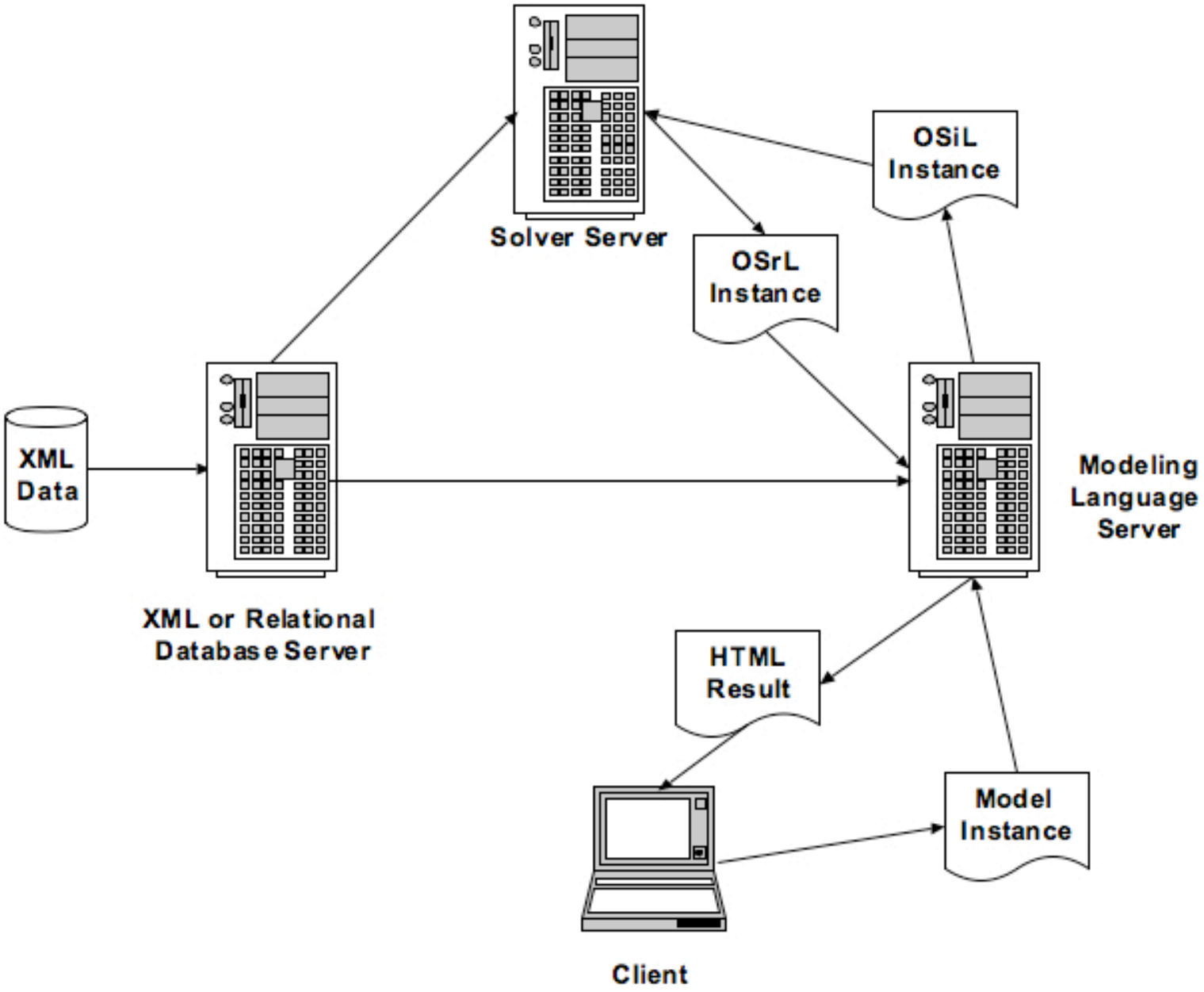
3. Optimization Services instance Language (OSiL) - this replaces LPFML for the linear case and can be used to represent a wide variety of optimization problems.

4. The OSiL schema

5. The OSExpressTree data structure

6. Real time data and user defined functions

7. Extensions


Solver Server
OSiL Instance
OSrL Instance
XML Data
XML or Relational Database Server
Modeling Language Server
HTML Result
Model Instance
Client

A Distributed Modeling Environment

In a loosely coupled setting we have a separation of the modeling language process and solver process.

Key idea: model versus instance

The solver wants an instance as opposed to a model

A MODEL

```
set PROD;  # products
set DEP;   # processing departments

param hours {DEP};      # time available in each department
param rate {DEP,PROD}; # hours used in each dept per product unit made
param profit {PROD};    # profit per unit of each product made

var Make {PROD} >= 0;   # number of units of each product to be made

maximize TotalProfit:
sum {j in PROD} profit[j] * Make[j];

subject to HoursAvailable {i in DEP}:
sum {j in PROD} rate[i,j] * Make[j] <= hours[i];
```

This is a **model**. It is *symbolic*, *general*, *concise*, and *understandable* (Fourer, 1983).

DATA

```
param: PROD: profit :=
           std    10
           del     9 ;

param: DEP:                hours :=
             cutanddye          630
             sewing             600
             finishing          708
             inspectandpack     135 ;

param: rate:                   std   del :=
             cutanddye         0.7   1.0
             sewing            0.5   0.8333
             finishing         1.0   0.6667
             inspectandpack    0.1   0.25  ;
```


MODEL + DATA = INSTANCE

maximize TotalProfit:
10*Make[‘std’] + 9*Make[‘del’];

subject to HoursAvailable[‘cutanddye’]:
0.7*Make[‘std’] + Make[‘del’] <= 630;

subject to HoursAvailable[‘sewing’]:
0.5*Make[‘std’] + 0.8333*Make[‘del’] <= 600;

subject to HoursAvailable[‘finishing’]:
Make[‘std’] + 0.6667*Make[‘del’] <= 708;

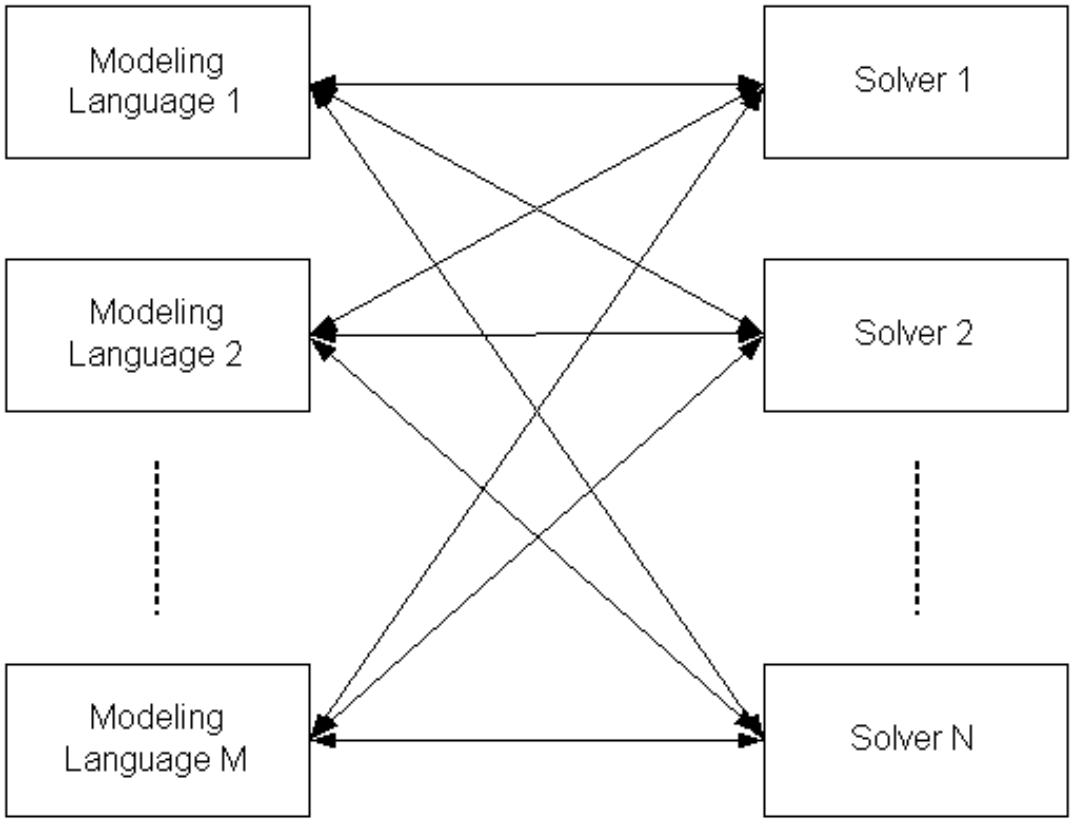
subject to HoursAvailable[‘inspectandpack’]:
0.1*Make[‘std’] + 0.25*Make[‘del’] <= 135;

Objective: represent a model instance using XML.

There is a proliferation of modeling languages and solvers

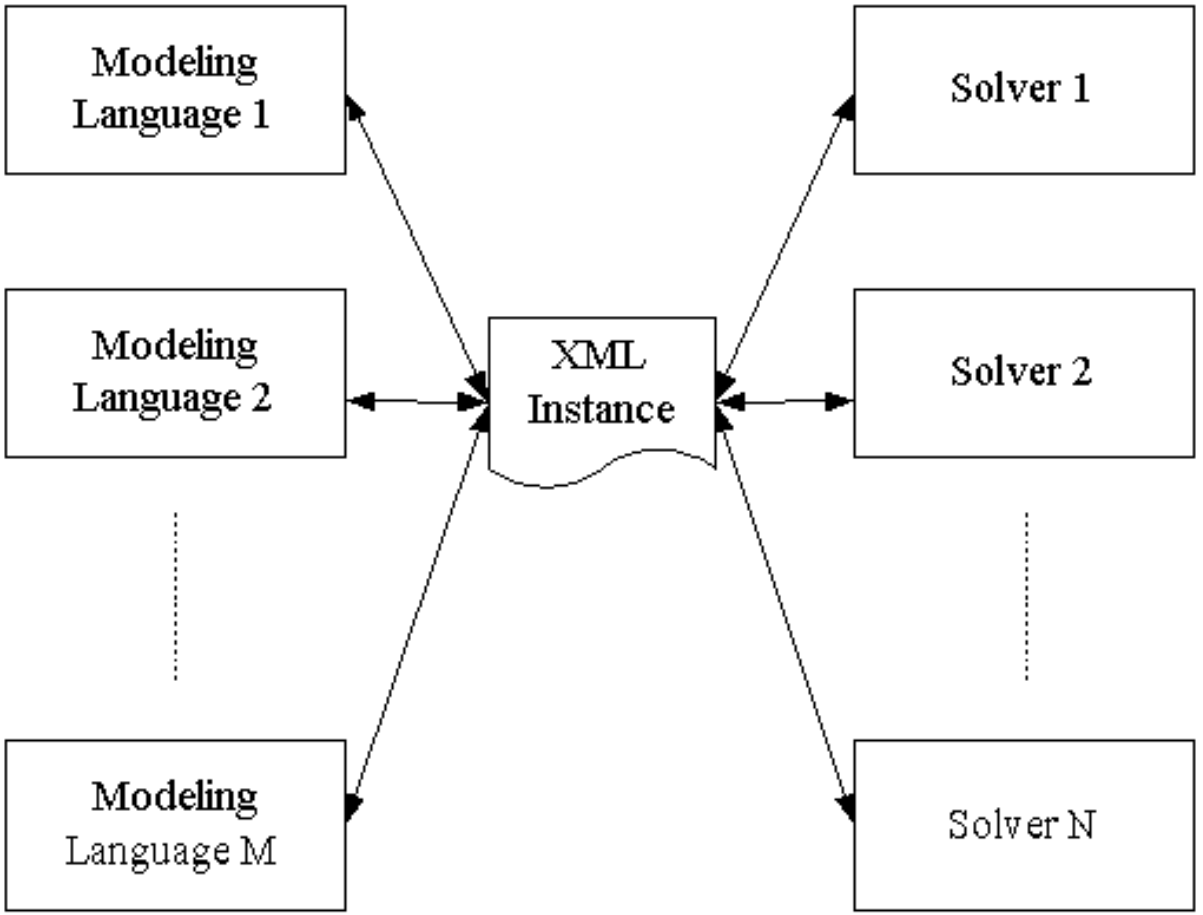
AIMMS	CLP
AMPL	CPLEX
GAMS	GLPK
LINGO	LINDO
Mosel	MINOS
MPL	MOSEK
OPL	Xpress-MP

Consequence: a lot of drivers are need for every modeling language to talk to every solver



MN Drives Required Without XML

It would be nice to have an instance representation language.



M + N Drives Required With XML

The Case for XML

1. Validation against a schema provides for error checking

2. Validation against a schema promotes stability of a standard

3. The schema can restrict data values to appropriate types, e.g. row names to **string**, indices to **integer**, coefficients to **double**

4. The schema can define keys to insure, for example, no row or column name is used more than once.

5. The schema can be extended to include new constraint types or solver directives

6. There is a lot of open source software to make parsing easy.

XML and Optimization Systems

1. When instances are stored in XML format, optimization technology solutions are more readily integrated into broader IT infrastructures

2. XML is used for Web Services – important for distributed computing

3. The XML format lends itself well to compression – more on this later

4. The XML format can be combined with other technologies, e.g. XSLT to present results in human readable formats

5. Encryption standards are emerging for XML – possibly important in a commercial setting.

XML Concepts

XML (Extensible Markup Language) – an XML file contains both data and Markup (Elements (tags) and Attributes)

The tags are organized in a **tree like** structure. The closing tag of a child element preceding the closing tag of its parent.

```
<constraints>
  <con name="cutanddye" ub="630"/>
  <con name="sewing" ub="600"/>
  <con name="finishing" ub="708"/>
  <con name="inspectandpack" ub="135"/>
</constraints>
```

ELEMENT

ATTRIBUTE

OSiL Instance Representation

$$\min 100(x_1 - x_0^2)^2 + (1 - x_0)^2 + 9 * x_1$$

$$x_0 + 3 * x_0 * x_1 + x_1^2 \leq 10$$

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

$$x_0, x_1 \geq 0$$

OSiL Instance Representation

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>
```

-OR-

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


OSiL Instance Representation

The objective functions: $\min 9 * x_1$

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


OSiL Instance Representation

The constraints:

$$\leq 10$$
$$\geq 10$$

```
<constraints number="2">
    <con name="row0" ub="10.0"/>
    <con name="row1" lb="10.0"/>
</constraints>
```


OSiL Instance Representation

The linear constraint terms:

$$x_0$$

$$7 * x_0 + 5 * x_1$$

```
<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>
```


OSiL Instance Representation

The nonlinear terms:

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

$$3 * x_0 * x_1 + x_1^2$$

$$\ln(x_0 x_1)$$

```
<nl idx="1">
   <ln>
      <times>
         <var idx="0"/>
         <var idx="1"/>
      </times>
   </ln>
</nl>
```

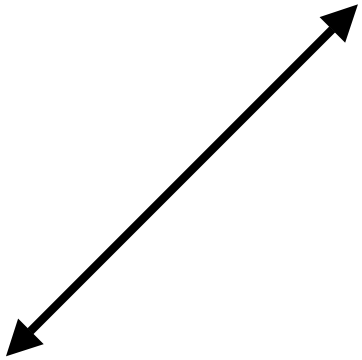

OSiL Instance Representation

The nonlinear terms:

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

$$3 * x_0 * x_1 + x_1^2$$

$$\ln(x_0 x_1)$$



```
<nl idx="0">
   <quadratic>
      <qpTerm idxOne="0" idxTwo="1" coef="3"/>
      <qpTerm idxOne="0" idxTwo=”0" coef=”1"/>
   </quadratic>
</nl>
```


XML Schema

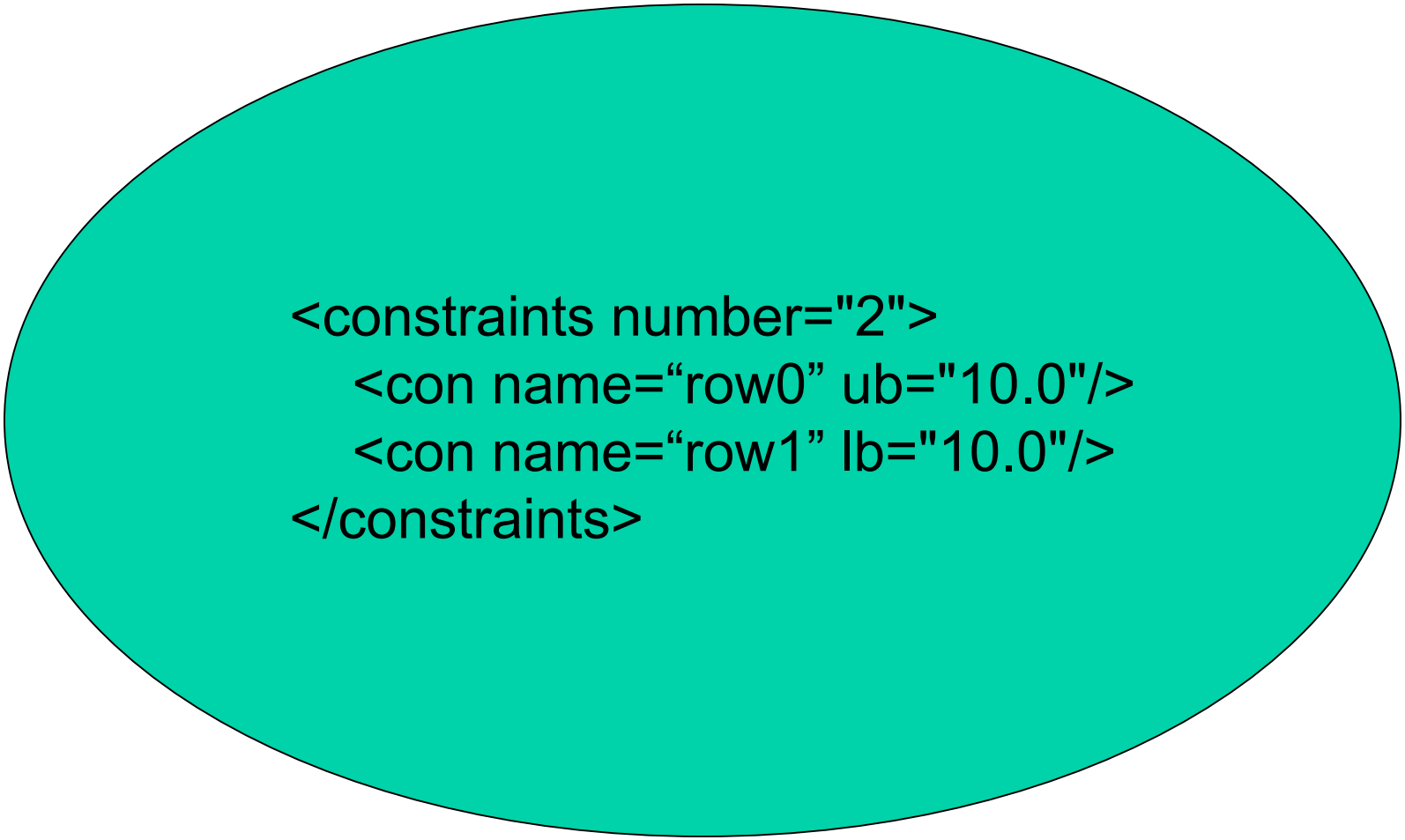
Key idea – a **schema**. Similar to the concept of a class in object orient programming. Critical for parsing!

Schema  **Class**

XML File  **Object**

We need a schema to represent an instance.

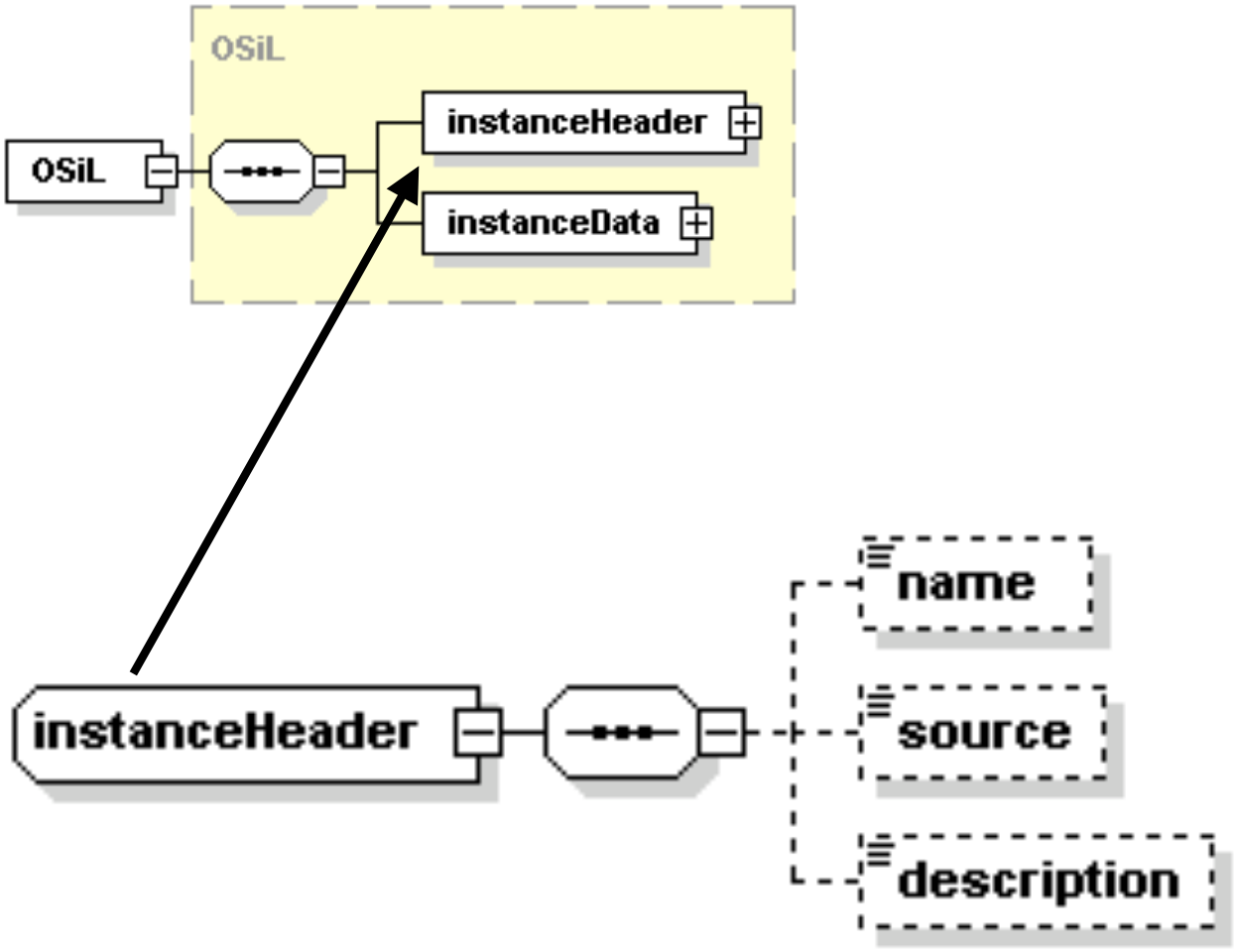
Schema – a Constraints Object



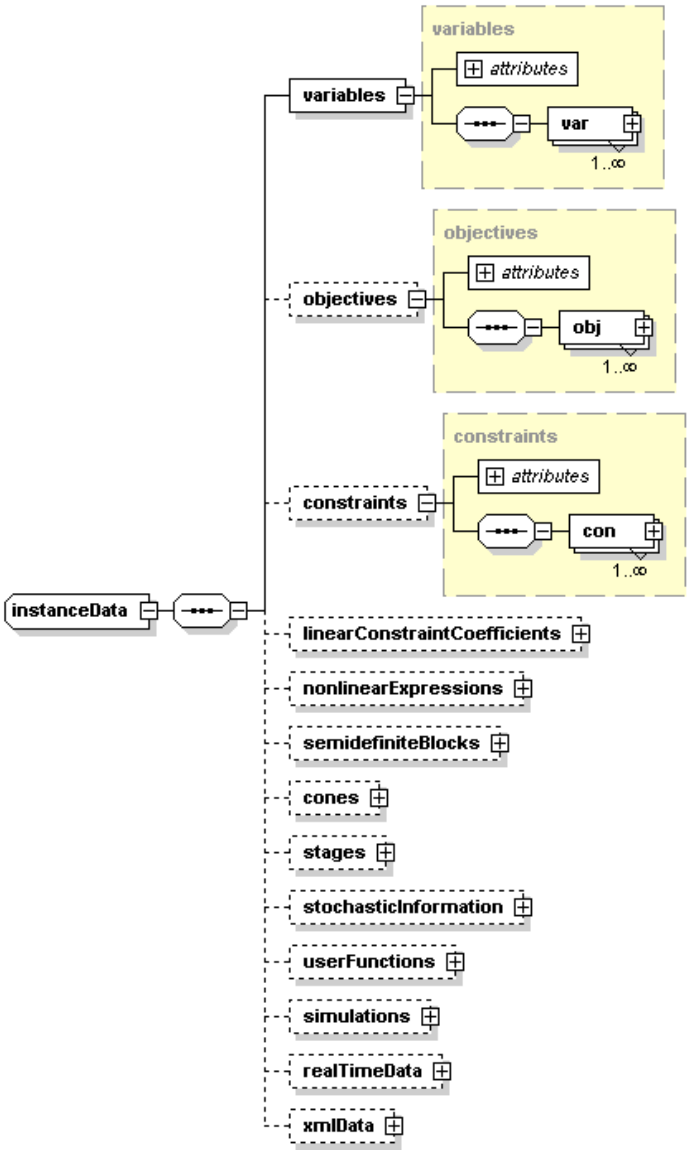
Schema – a Constraints and Con Class

```
<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 OSiL Schema



OSnLNode

Problem: parsing, doing function and gradient evaluations, etc. a real PAIN with numerous operators and operands.

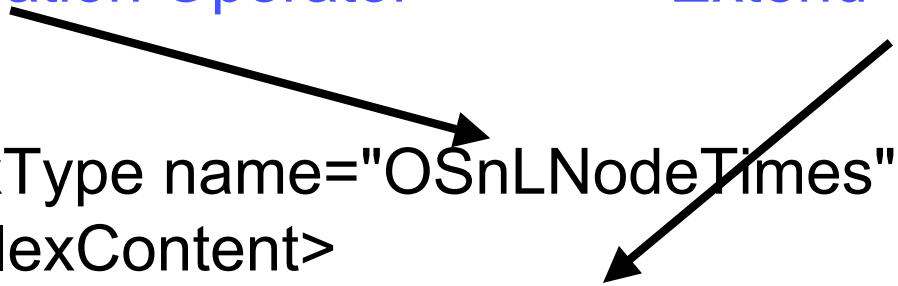
We avoid this by having EVERY nonlinear node an OSnLNode instance.

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


OSnLNode

The Multiplication Operator

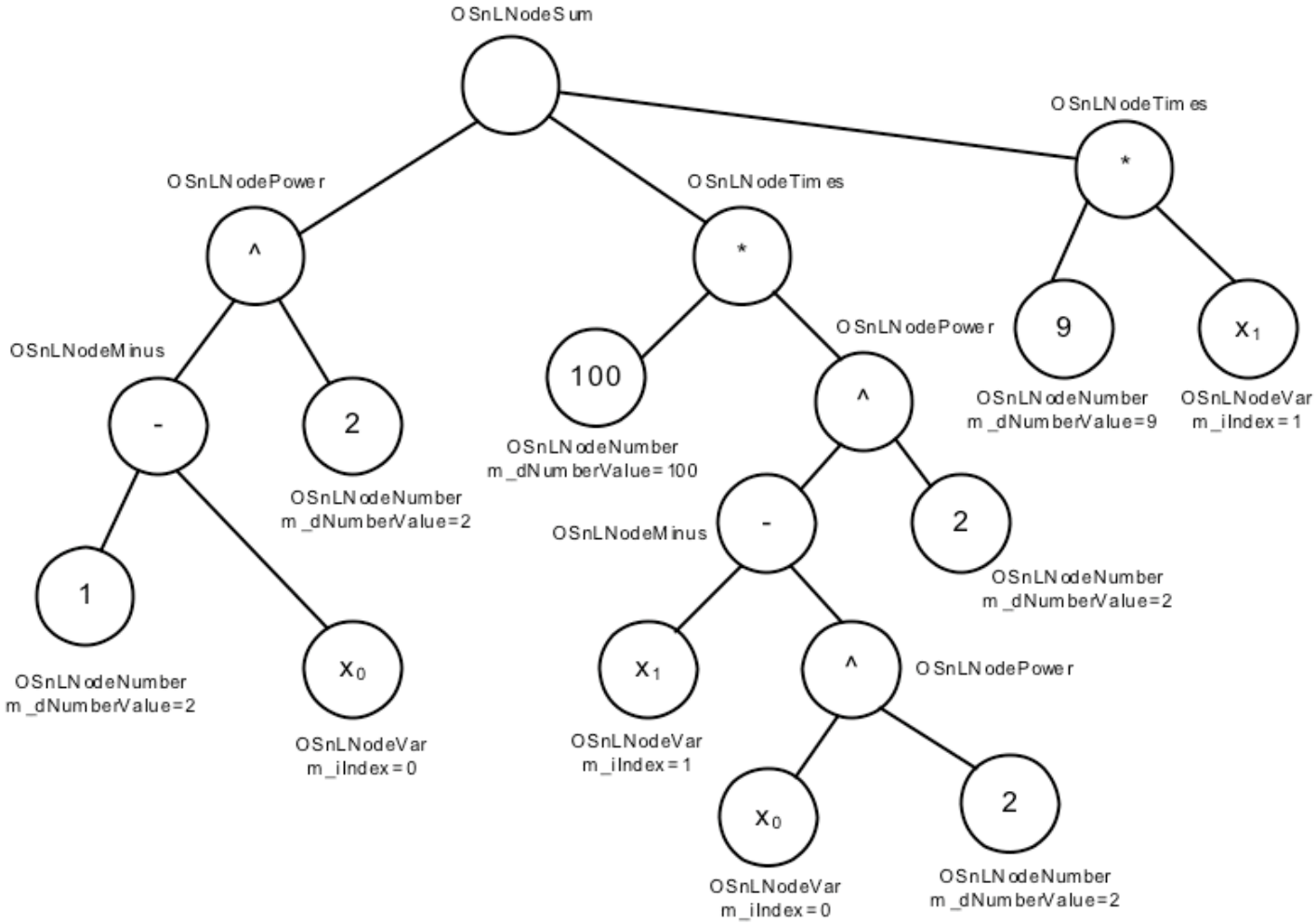
Extend OSnLNode



```
<xs:complexType name="OSnLNodeTimes">
   <xs:complexContent>
      <xs:extension base="OSnLNode">
         <xs:sequence minOccurs="2" maxOccurs="2">
            <xs:element ref="OSnLNode"/>
         </xs:sequence>
      </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:element name="times" type="OSnLNodeTimes"
substitutionGroup="OSnLNode"/>
```


OSExpressionTree

$$100(x_1 - x_0^2)^2 + (1 - x_0)^2 + 9 * x_1$$



OSExpressionTee (Parsing)

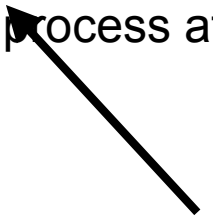
A traditional C approach for parsing. Construct *e a pointer to root of expression tree. Then process the expression tree.

```
double evaluate\_function (expr *e, double x[]){
    ...
    opnum = e->op
    switch(opnum){
        case PLUS_opno: ...
        case MINU_opno: ...
        ...
    }
}
```

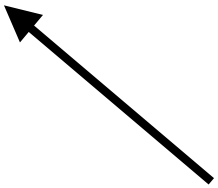

OSExpressionTree (Parsing)

We take an object oriented approach, every node in the expression tree is an instance in the OSnLNode class

```
OSnLNode nlNode = null;
String sNodeName = "";
try{
    sNodeName = ele.getLocalName();
    String sNlNodeClass = m\_sPackageName + "." + m\_sNlNodeStartString +]
        sNodeName.substring(0, 1).toUpperCase() + sNodeName.substring(1);
    Class nlNodeClass  = Class.forName(sNlNodeClass);
    nlNode = (OSnLNode)nlNodeClass.newInstance();
} // now process attributes
```



An instance of OSnLNode which is an OSnLNodeTimes



“OSnLNodeTimes”

Real Time Data

In many cases the instance generated by the solver contains time sensitive data. For example, in many financial models.

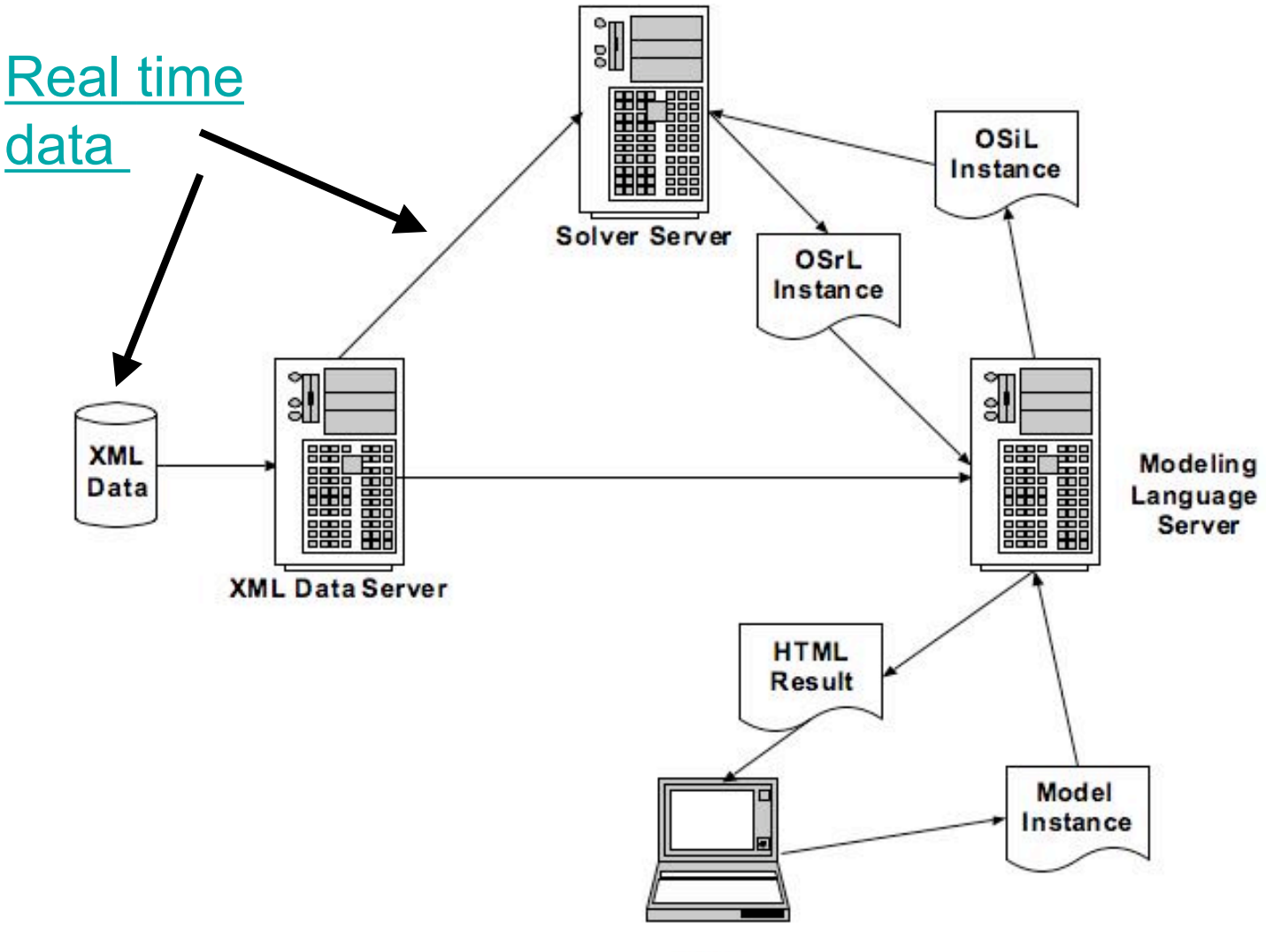
Before solving we can:

1. Repeat entire modeling process and have modeling language generate a new model from scratch.

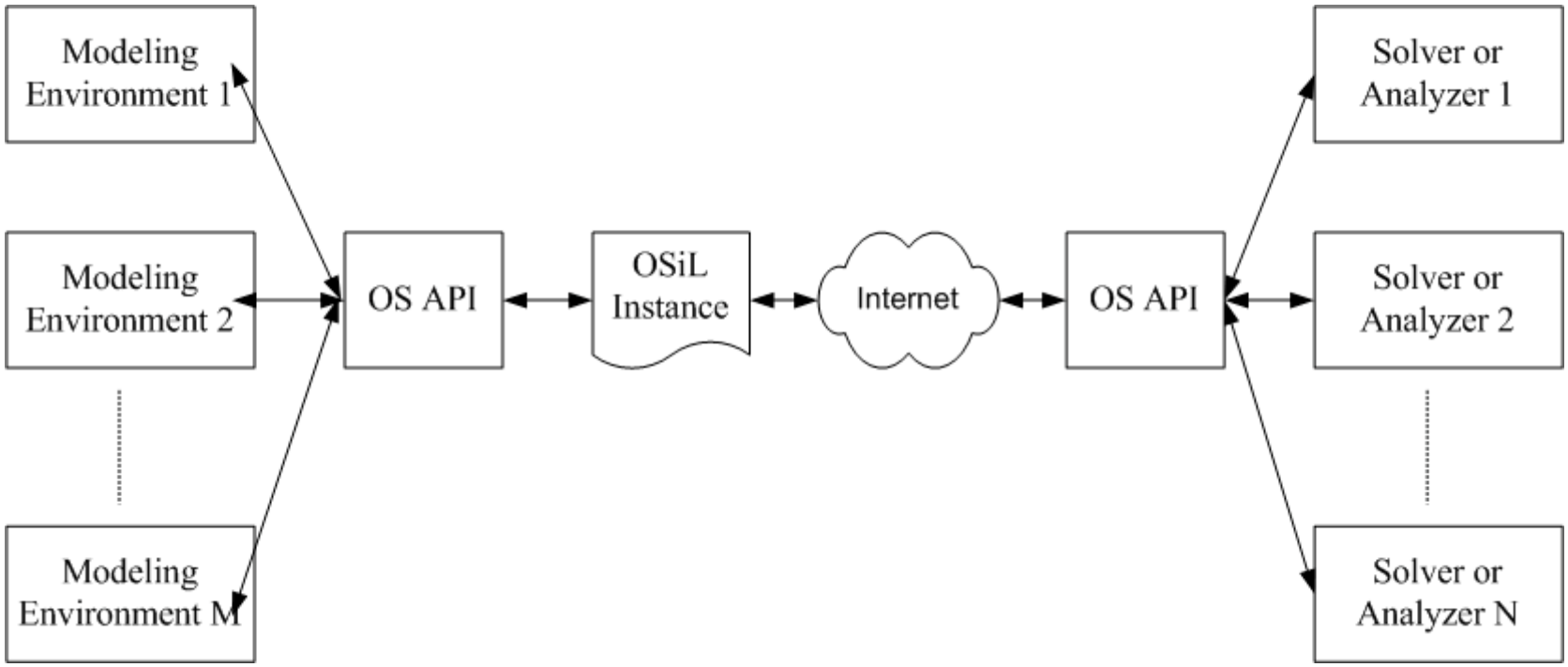
OR

2. Have the “reader” library update only the necessary data before sending it to the solver.

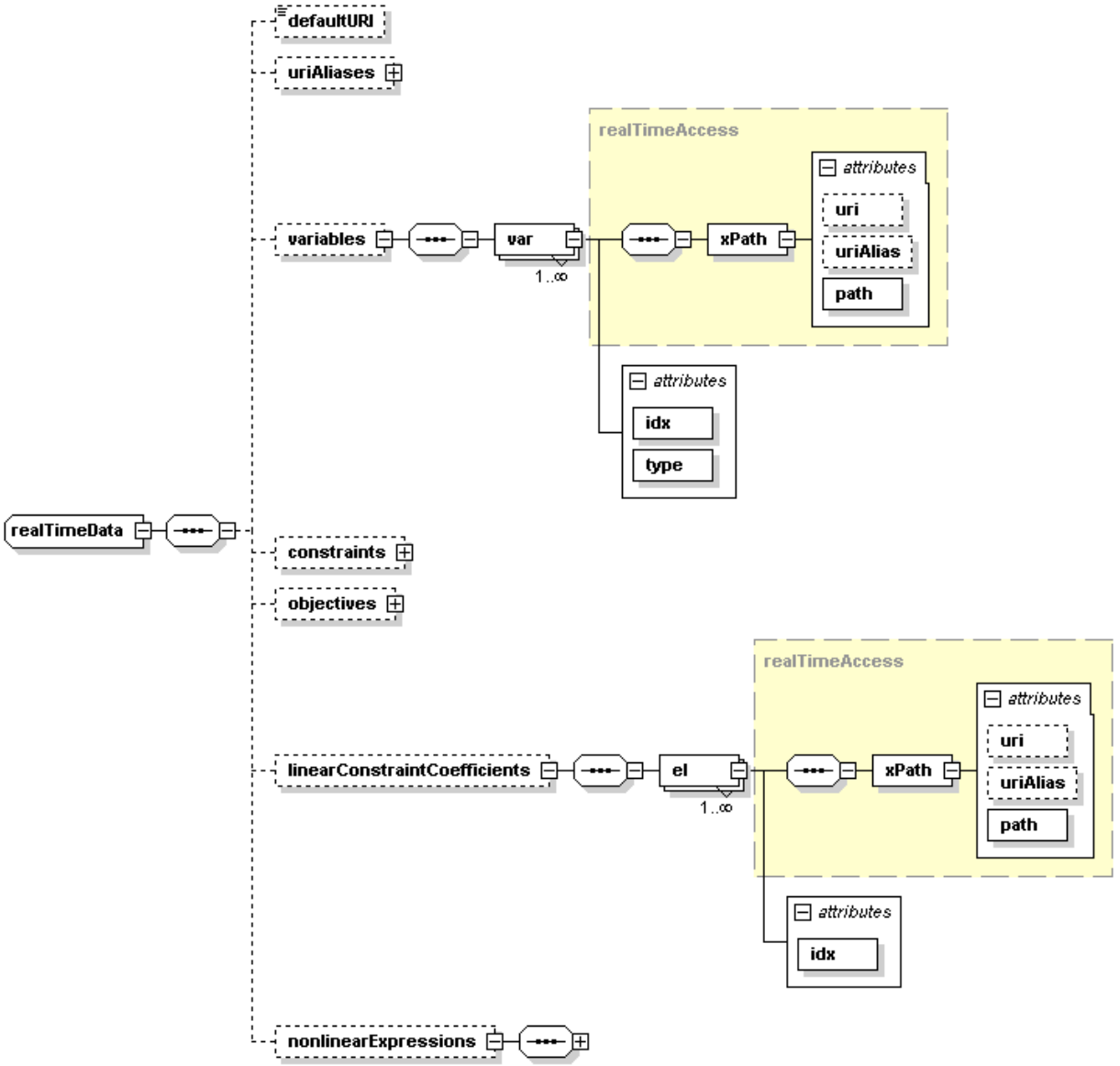
Real Time Data



Real Time Data



Real Time Data



Markowitz Example

$$\min \sum_{i=1}^{3} p_s (R - R_s)^2$$

$$x[msft] + x[pg] + x[ge] = 1$$

$$\bar{R} \geq r$$

$$r[s, msft]x[msft] + r[s, pg]x[pg] + r[s, ge]x[ge] = R_s$$

$$\sum_{s=1}^{3} p_s R_s = \bar{R}$$

$$x[msft], x[pg], x[ge] \geq 0$$

Markowitz and Real Time Data

$$\bar{R} \geq r$$

```
<variables number="4">
   <var name="msft" lb="0.0" ub=".75"/>
   <var name="pg" lb="0.0" ub=".75"/>
   <var name="ge" lb="0.0" ub=".75"/>
   <var name="RBAR" lb=".05"/>
</variables>
```

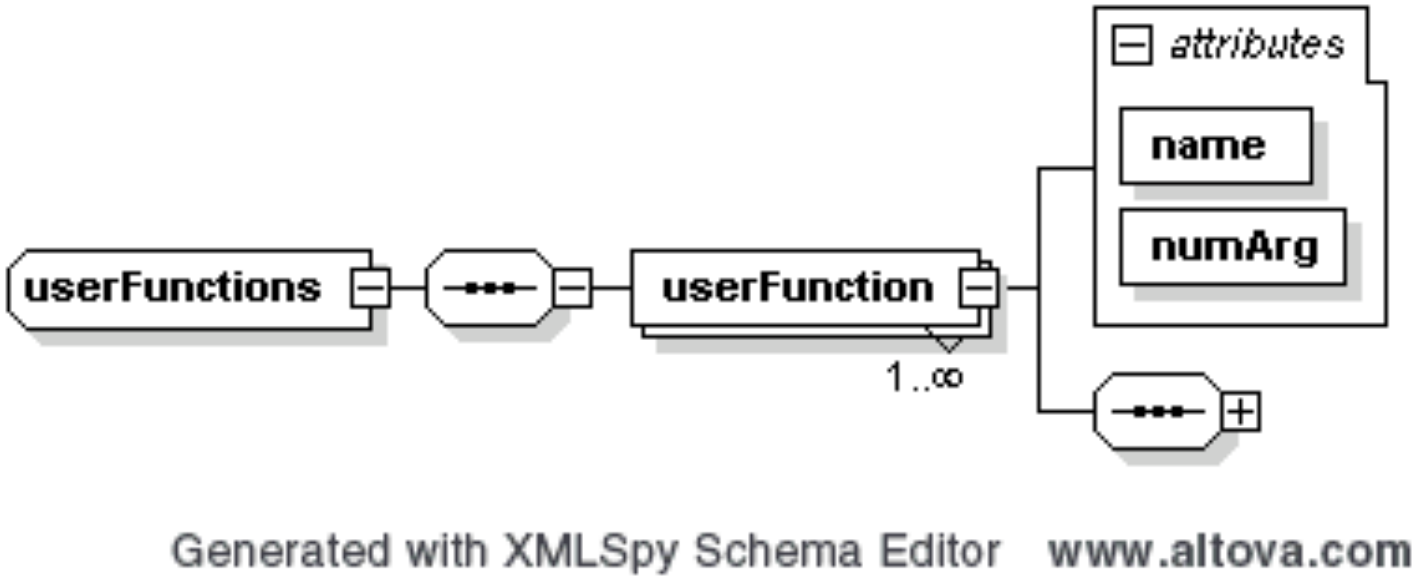
```
<realTimeData>
   <defaultURI>http://www.stockdata.com/stockdata.xml</defaultURI>
   <variables>
      <var idx="3" type="lb">
         <xPath path="/xmlData/portfolioReturn/text()" />
      </var>
   </variables>
</realTimeData>
```


User Defined Functions

Many istances often:

1. Contain terms repeated many times, either verbatim or With small systematic changes

2. Contain definitional variables



User Defined Functions

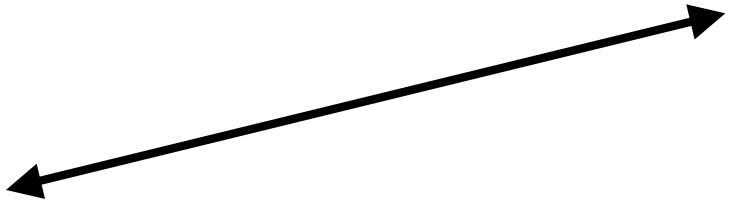
$$r[s,msft]x[msft] + r[s,pg]x[pg] + r[s,ge]x[ge] = R_s$$

```
<userFunction name="stockRet" numArg="2">
   <xPath path="//scenario[@id=$scenario]/stock[@name=$name]/@return">
      <xPathIndex indexName="scenario">
         <arg idx="0"/>
      </xPathIndex>
      <xPathIndex indexName="name">
         <arg idx="1"/>
      </xPathIndex>
   </xPath>
</userFunction>
```

Idx=0 idx = 1

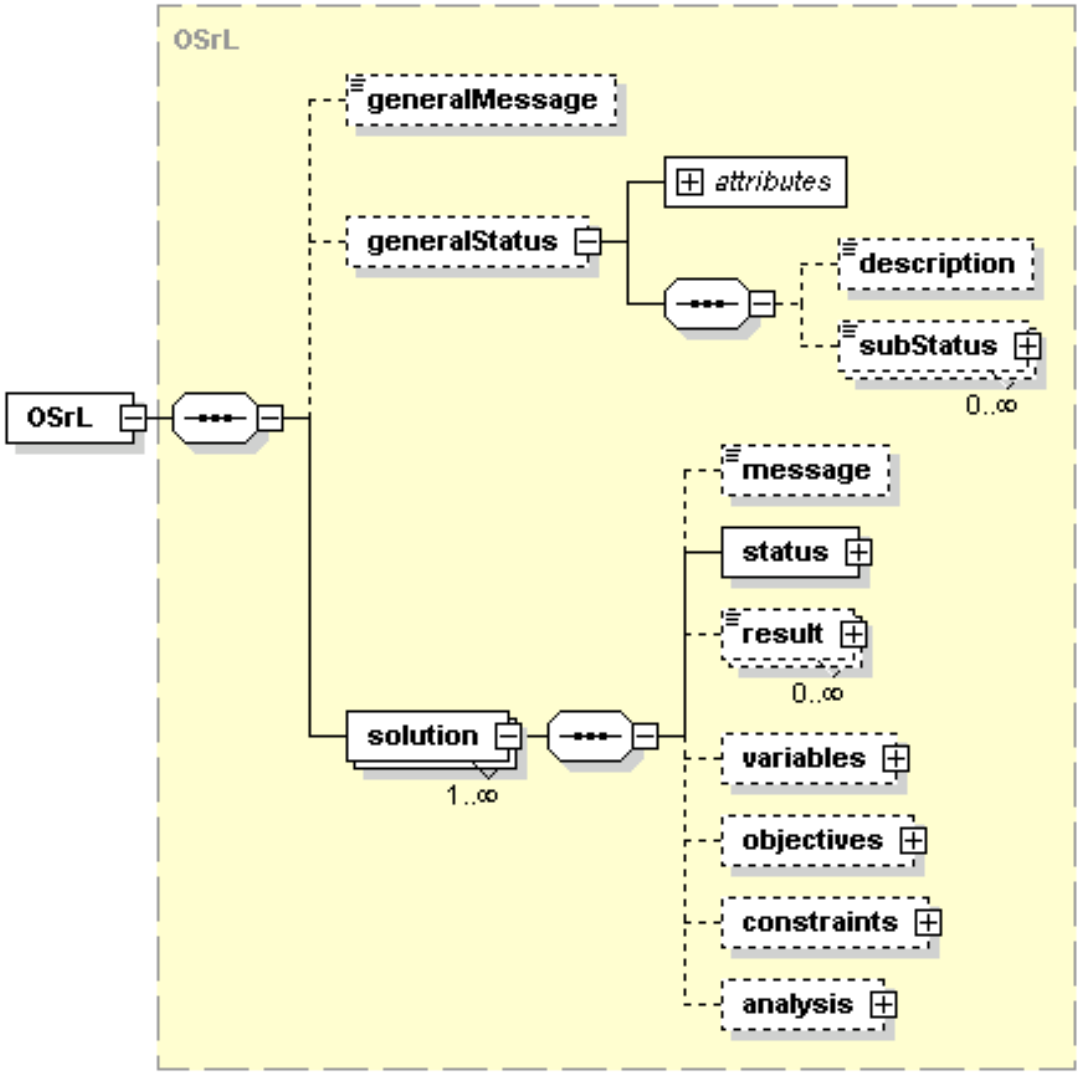
User Defined Functions

$$r[s,msft]x[msft] + r[s,pg]x[pg] + r[s,ge]x[ge] = R_s$$



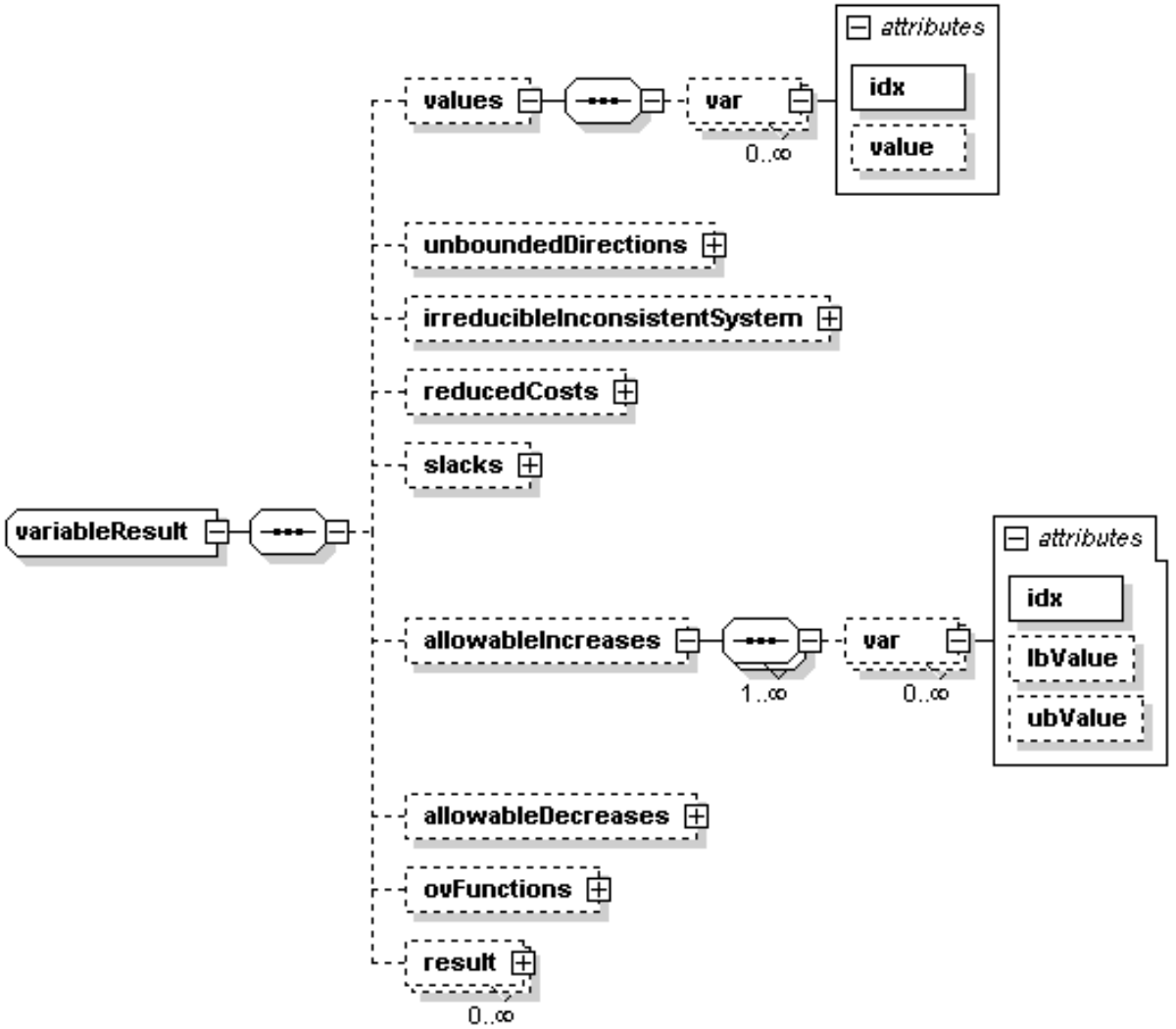
```
<userFunction name = "scenarioRet" numArg="1">
   <sum>
      <times>
        <userF name="stockRet">
           <arg idx="0"/>
           <string value="msft"/>
        </userF>
        <var idx="0"/>
      </times>
        ⇕
   </sum>
</userFunction>
```


OSrL - Optimization Services Result Language

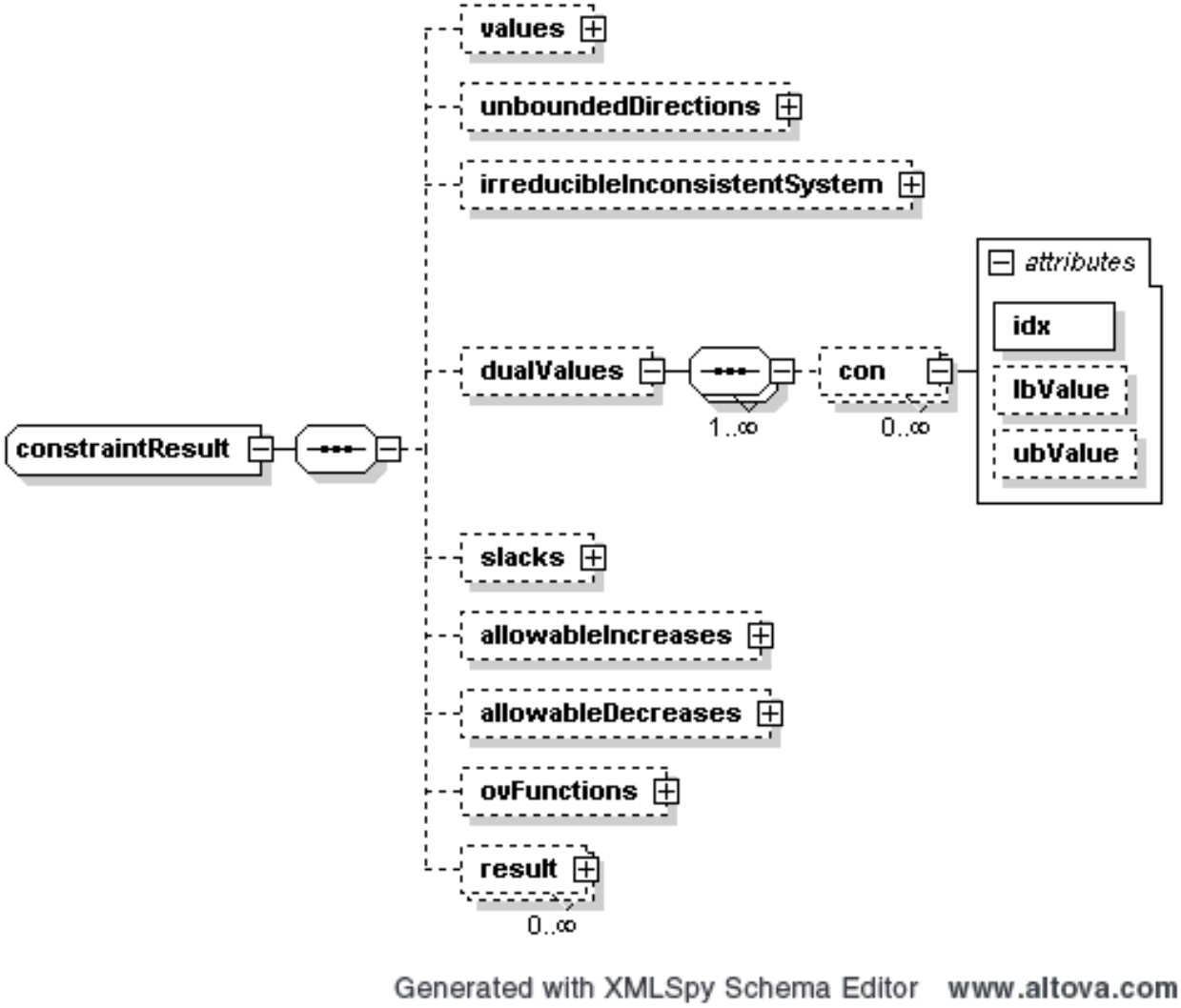


Generated with XMLSpy Schema Editor www.altova.com

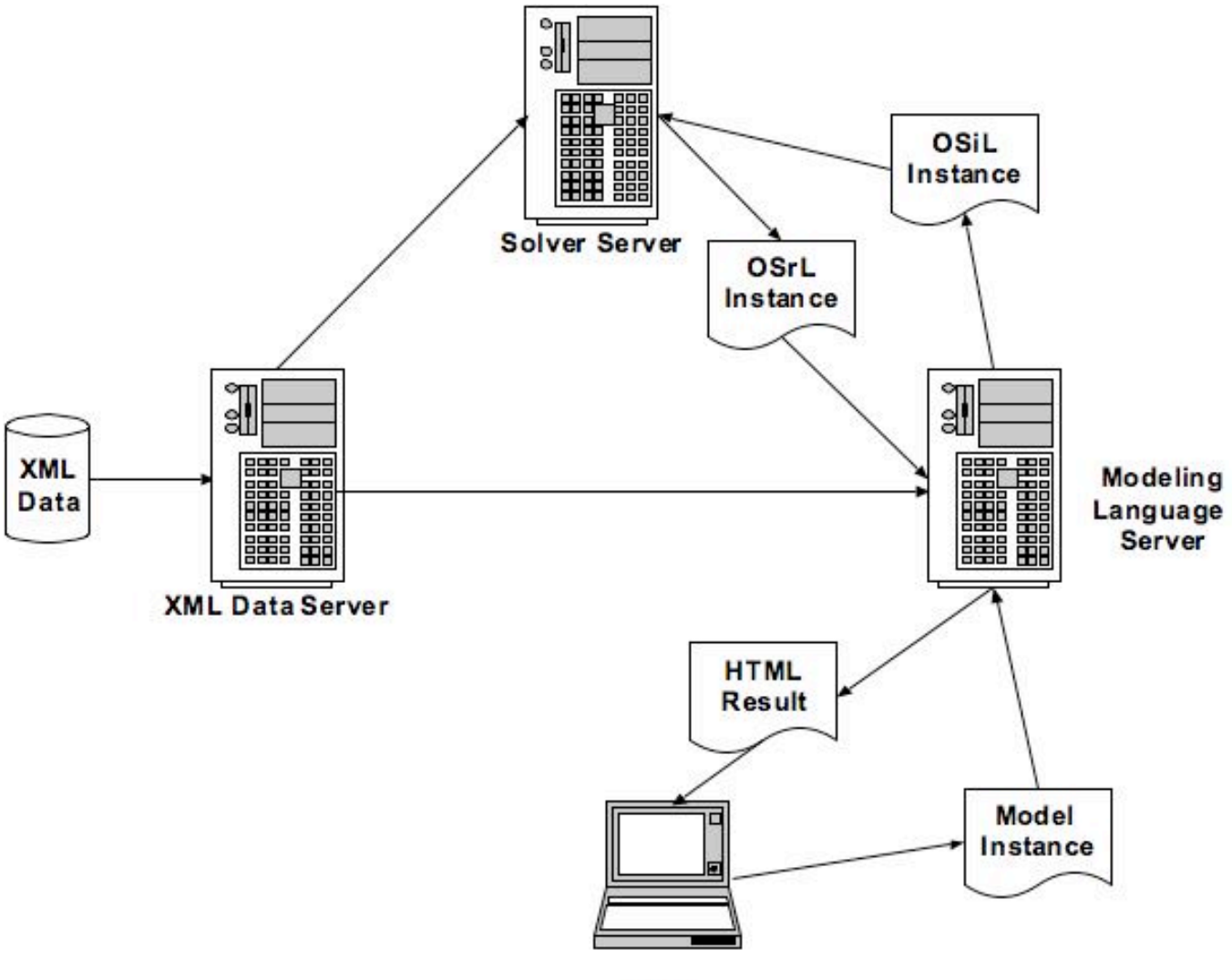
OSrL - Optimization Services Result Language



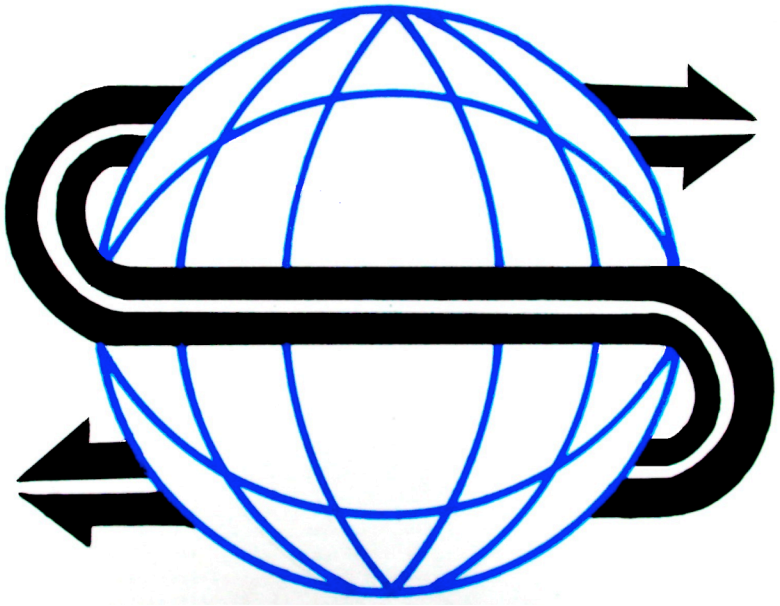
OSrL - Optimization Services Result Language



A Distributed Modeling Environment



QUESTIONS?



http://www.optimizationservices.org