Optimization Services Instance Language (OSiL)

Part I

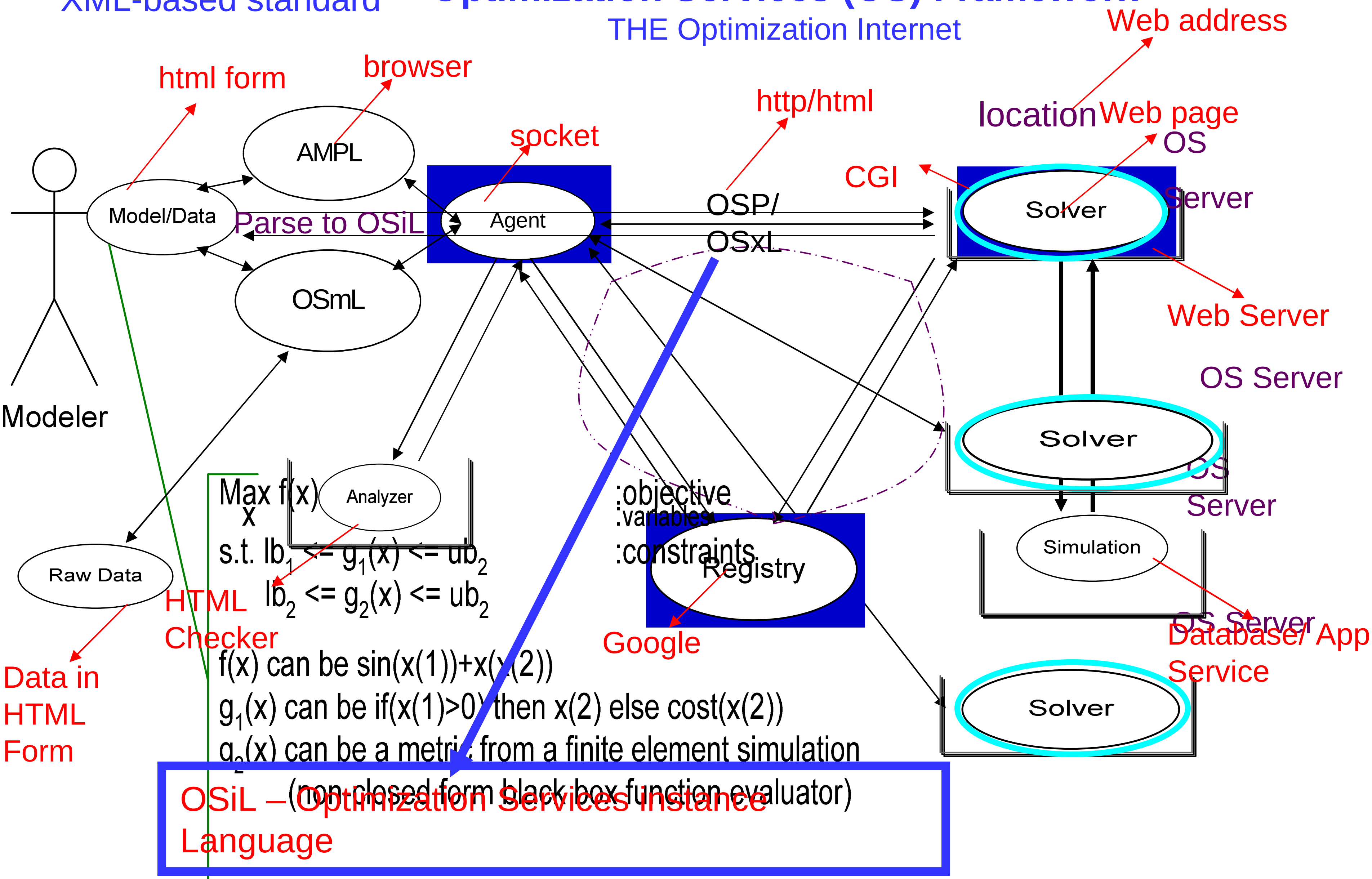
Robert Fourer
Jun Ma
Northwestern University
Kipp Martin
University of Chicago

Jun Ma
maj@northwestern.edu
Industrial Engineering and Management Sciences, Northwestern University
11/22/2004

XML-based standard

Optimization Services (OS) Framework

THE Optimization Internet



OUTLINE

1. Introduction and motivation

2. Model versus Instance

3. Why XML?

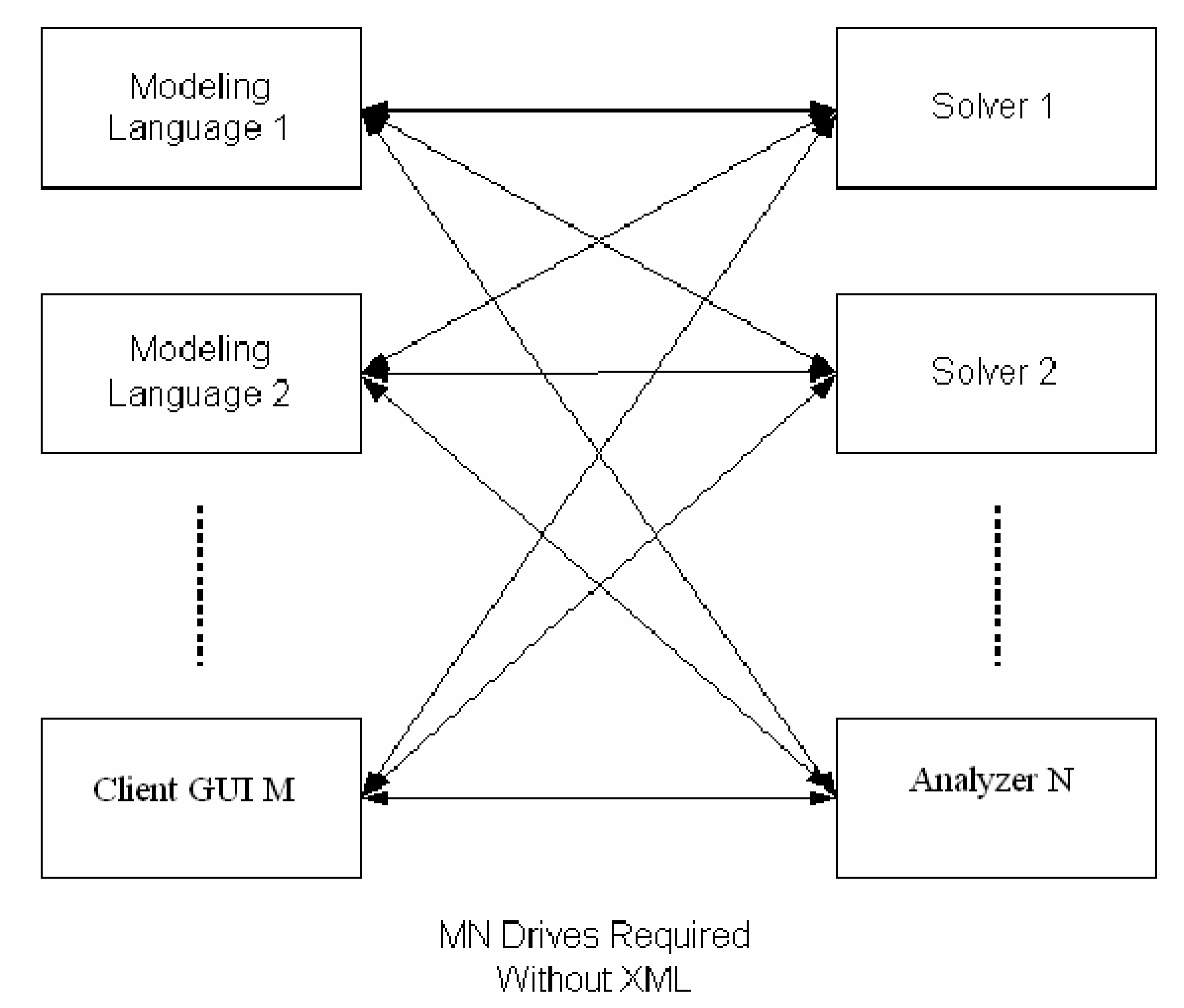
4. The OSiL Schema

5. Conclusion

There is a proliferation of modeling languages and solvers

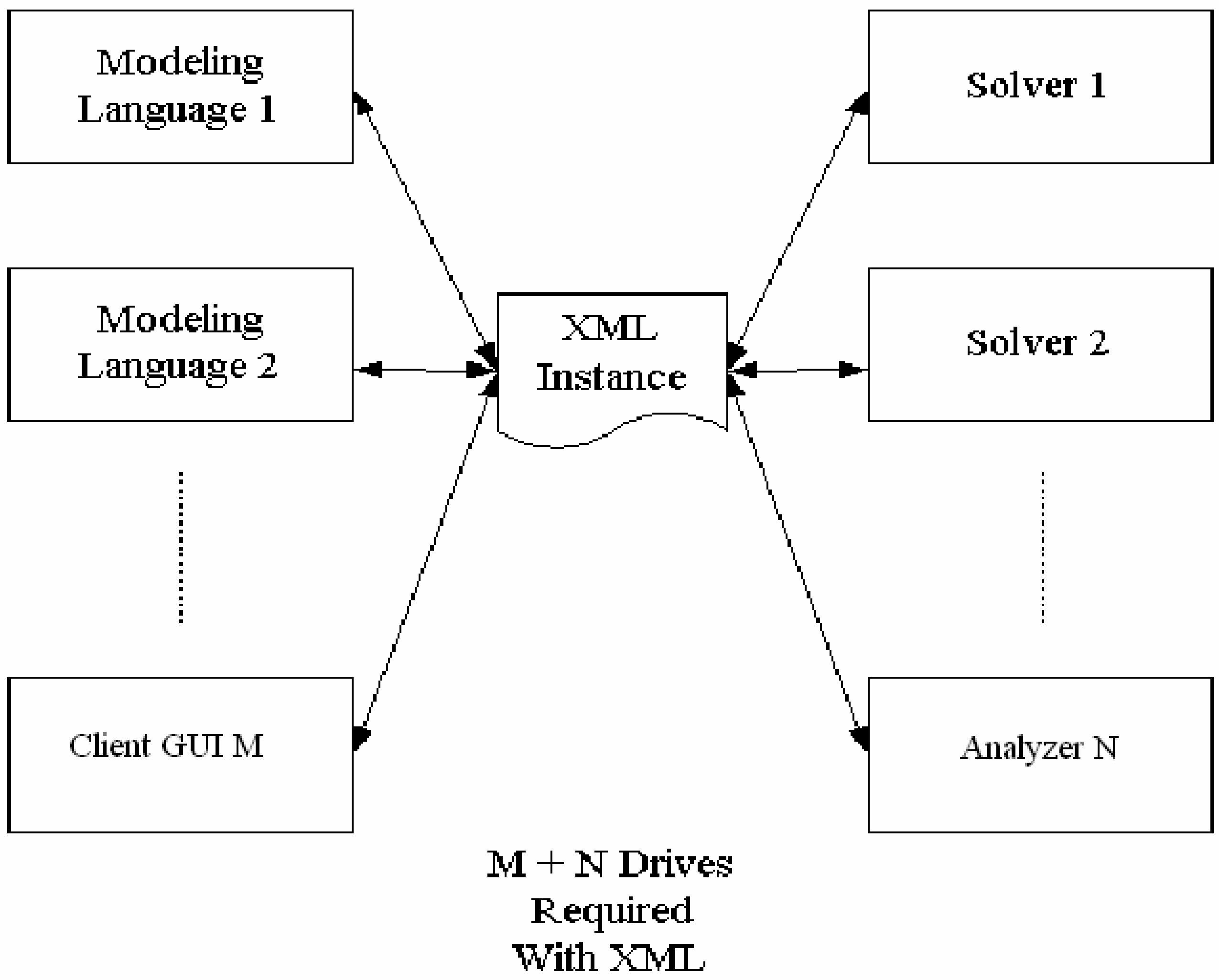
AIMMS	CLP
AMPL	CPLEX
GAMS	GLPK
LINGO	LINDO
OSmL	MINOS
MPL	MOSEK
OPL	Xpress-MP

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



Robert Fourer, Jun Ma, Kipp Martin

It would be nice to have an instance representation language.



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];
```

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;

Why not MPS?

```
NAME        PRODMIX
ROWS
 N  TPROFIT
 L  HRSCUT
 L  HRSSEW
 L  HRSFIN
 L  HRSINS
COLUMNS
      MAKESTD   TPROFIT   10
      MAKESTD   HRSCUT    0.7        HRSSEW    0.5
      MAKESTD   HRSFIN    1          HRSINS    0.1
      MAKEDEL   TPROFIT   9
      MAKEDEL   HRSCUT    1          HRSSEW    0.8333
      MAKEDEL   HRSFIN    0.6667     HRSINS    0.25
RHS
    RHS1     HRSCUT    630
    RHS1     HRSSEW    600
    RHS1     HRSFIN    708
    RHS1     HRSINS    135
ENDATA
```

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

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.

```
<rows>
  <row rowName=“cutanddye" rowUB="630"/>
  <row rowName="sewing" rowUB="600"/>
  <row rowName="finishing" rowUB="708"/>
  <row rowName="inspectandpack" rowUB="135"/>
</rows>
```

ELEMENT

ATTRIBUTE

XML Schema Concepts – a Row Class

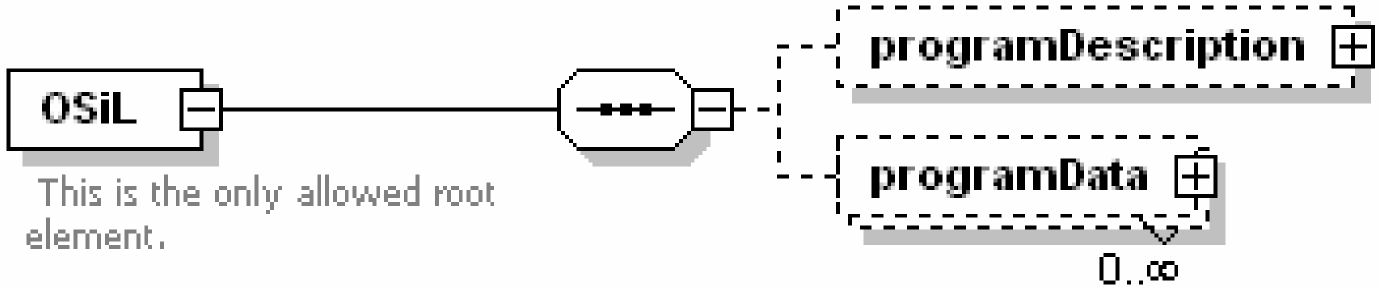
```
<xs:element name="rows">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="row" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="rowName" type="xs:string" use="optional"/>
          <xs:attribute name="rowUB" type="xs:double" use="optional"/>
          <xs:attribute name="rowLB" type="xs:double" use="optional"/>
          <xs:attribute name="mult" type="xs:int" use="optional"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Concepts – a Row Object

```
<rows>
   <row rowName=“cutanddye" rowUB="630"/>
   <row rowName="sewing" rowUB="600"/>
   <row rowName="finishing" rowUB="708"/>
   <row rowName="inspectandpack" rowUB="135"/>
</rows>
```

OSiL Schema

1. General Architecture



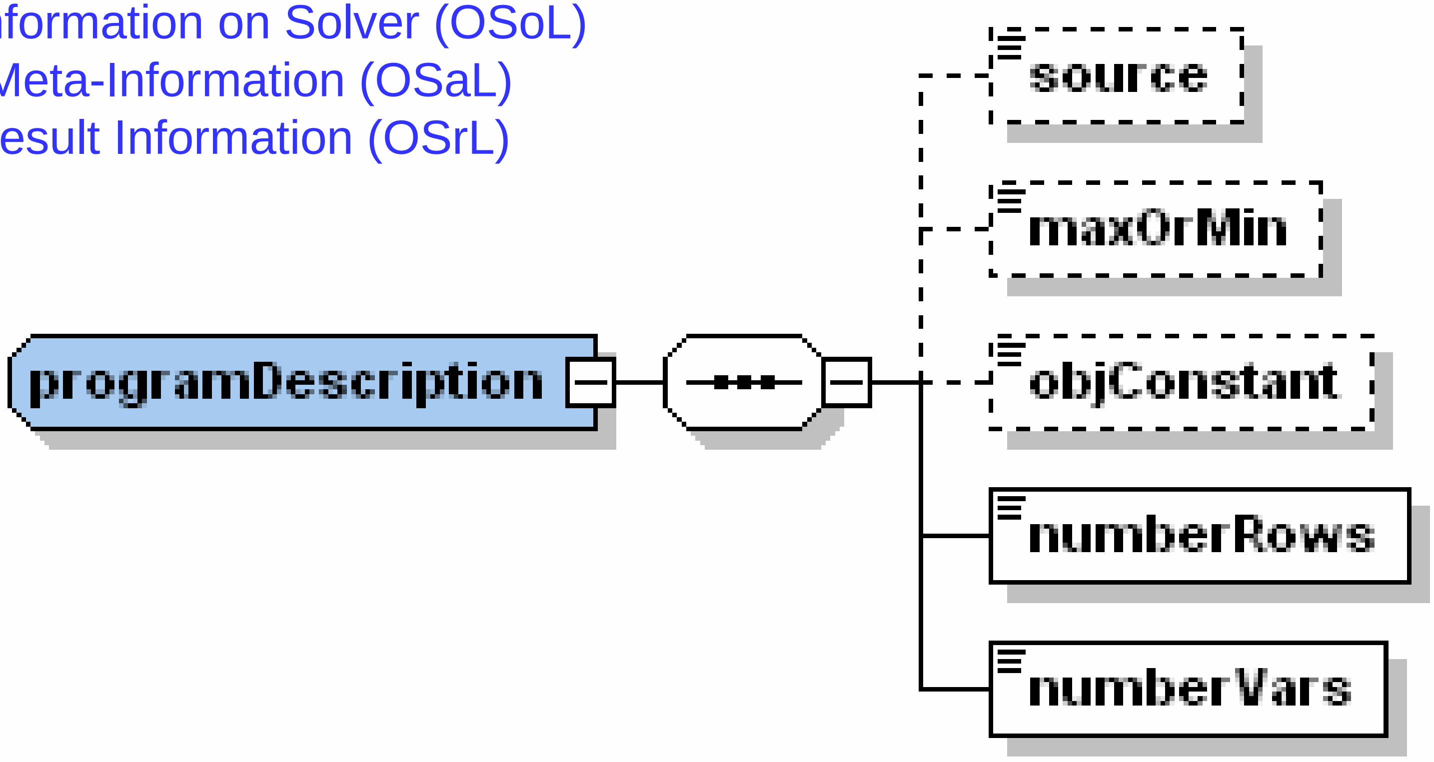
OSiL Schema

2. Information about the instance

No Information on Solver (OSoL)

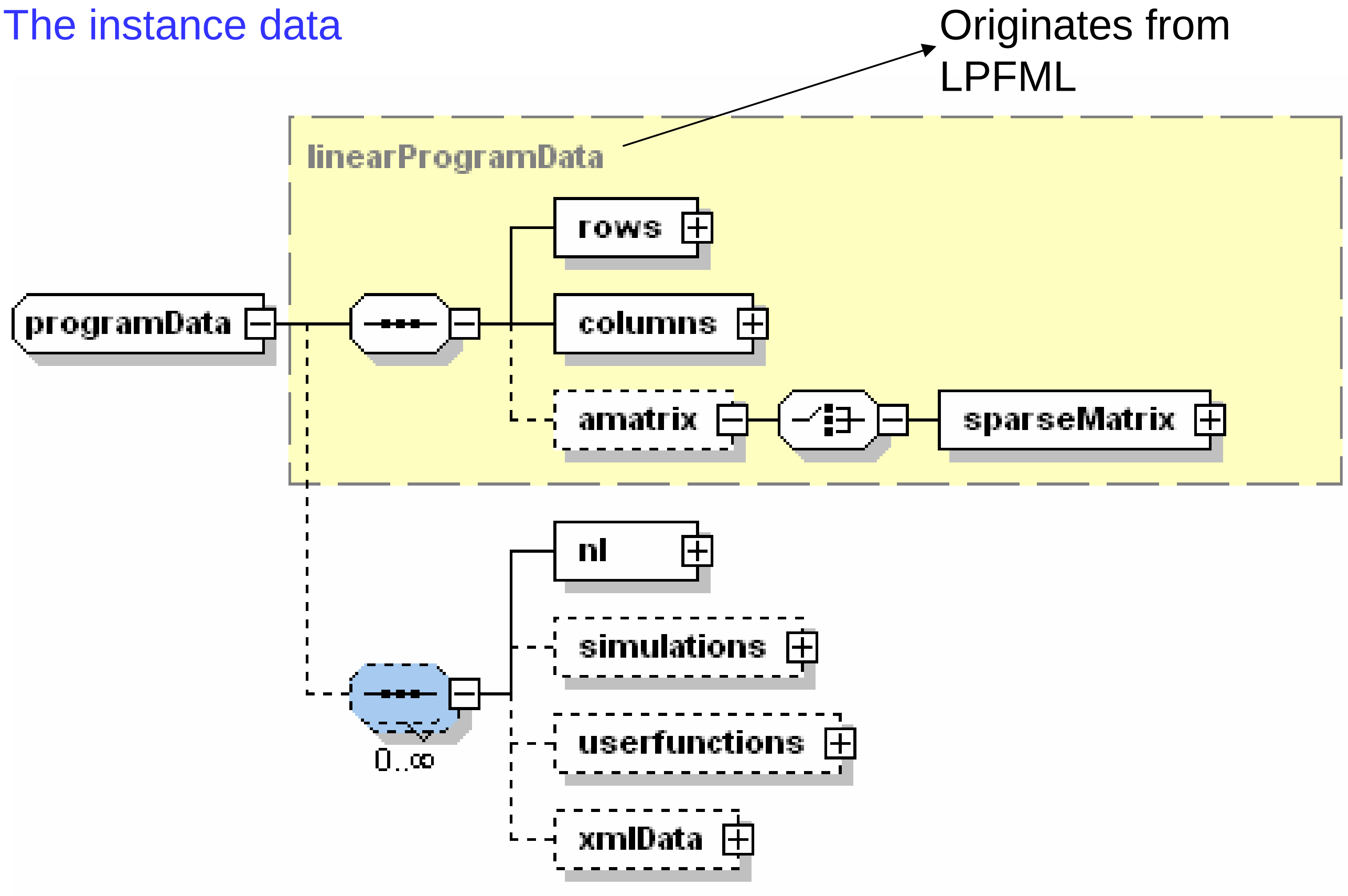
*No Meta-Information (OSaL)

No Result Information (OSrL)



OSiL Schema

2. The instance data



OSiL Schema

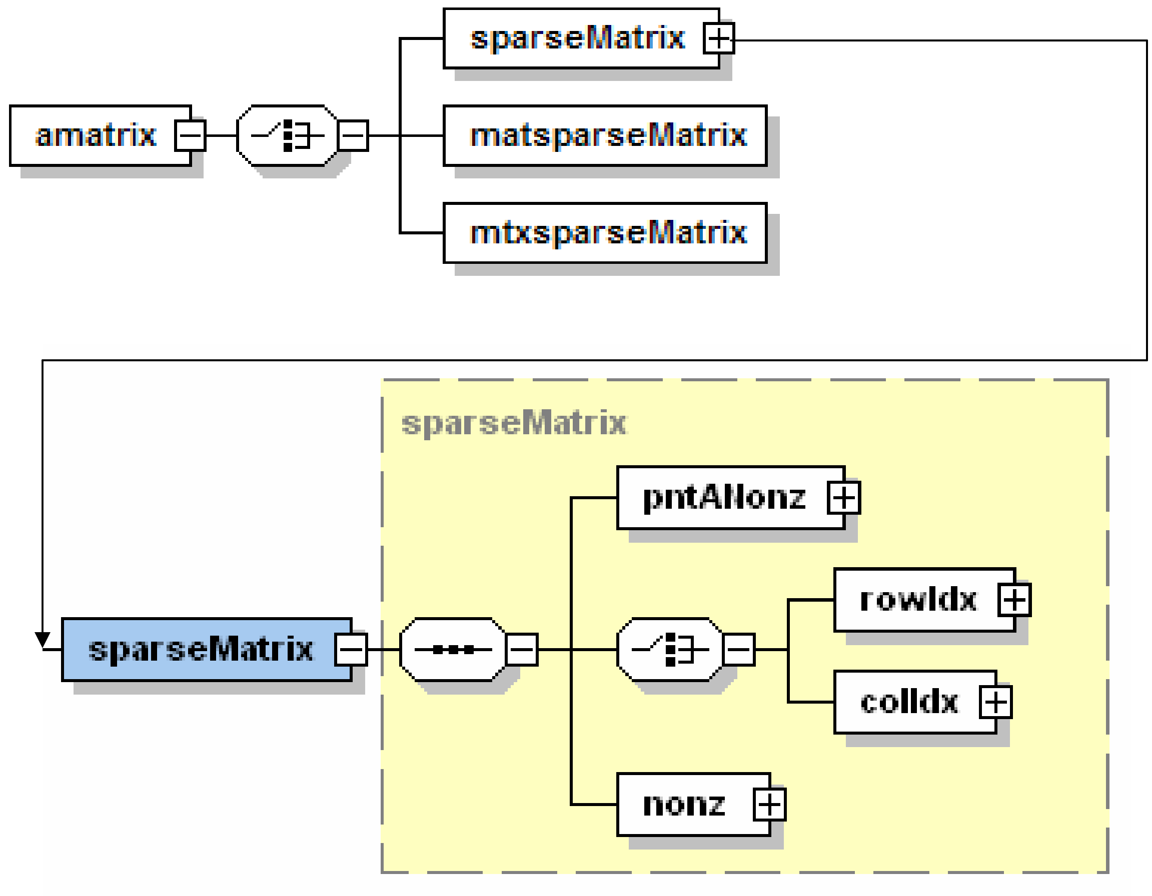
2. The instance data (rows and columns)

```
<rows>
  <row rowName="HoursAvailable[‘cutanddye’]" rowUB="630"/>
  <row rowName="HoursAvailable[‘sewing’]" rowUB="600"/>
  <row rowName="HoursAvailable[‘finishing’]" rowUB="708"/>
  <row rowName="HoursAvailable[‘inspectandpack’]" rowUB="135"/>
</rows>

 <columns>
   <col objVal="10" colName="Make[‘std’]" colType="C" colLB="0.0"/>
   <col objVal="9" colName="Make[‘del’]" colType="C" colLB="0.0"/>
 </columns>
```

OSiL Schema

2. The instance data (the A matrix)



OSiL Schema

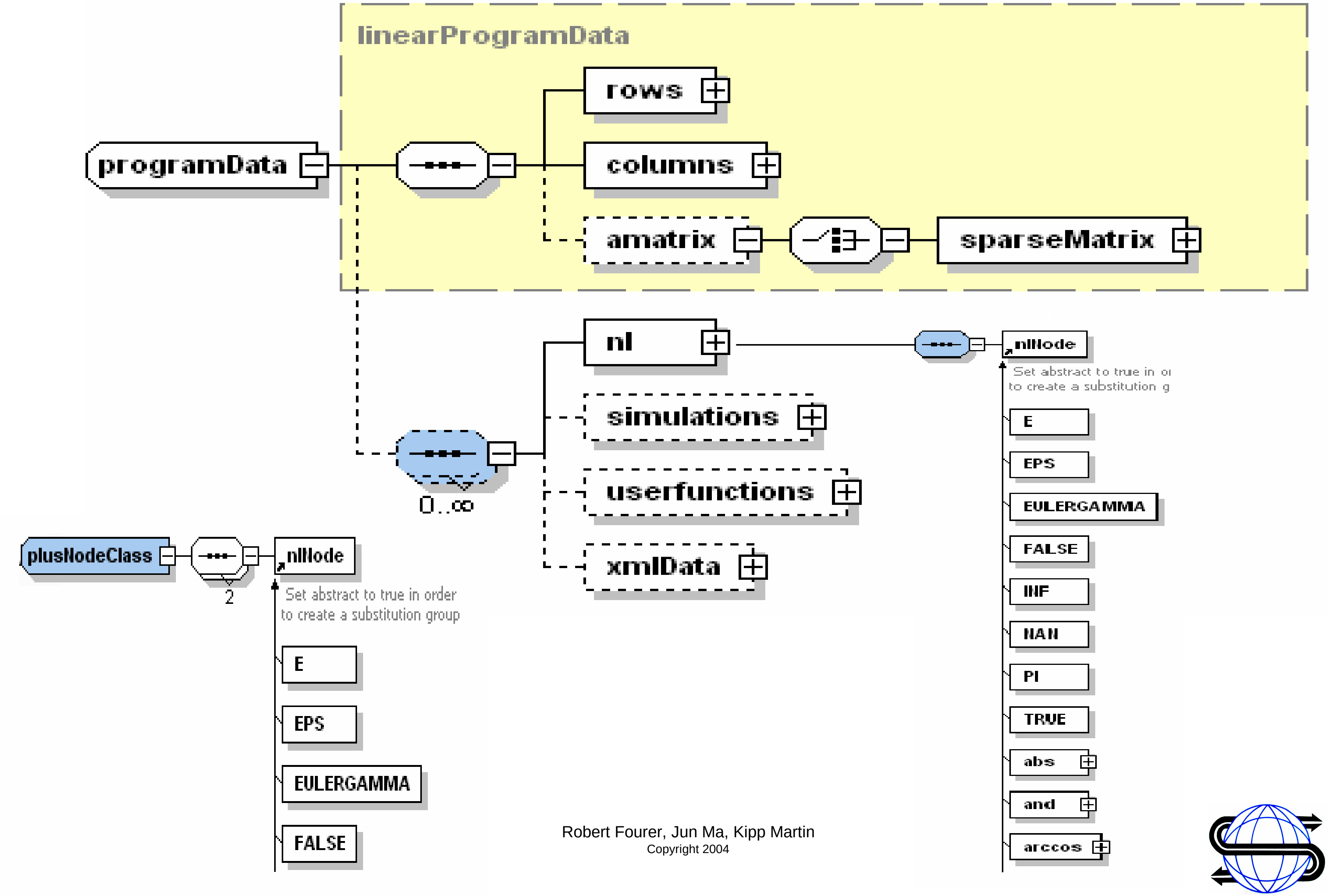
2. The instance data (the A matrix)

```
<sparseMatrix>
  <pntANonz>
    <el>2</el><el>4</el>
  </pntANonz>
  <rowIdx>
    <el>1</el><el>2</el>
    <el>0</el><el>1</el>
  </rowIdx>
  <nonz>
    <el>1</el><el>2</el>
    <el>-3</el><el>4</el>
  </nonz>
</sparseMatrix>
```

$$\begin{pmatrix} 0 & -3 \\ 1 & 4 \\ 2 & 0 \end{pmatrix}$$

OSiL Schema

3. Extensions



Conclusion and Extension

1. The libraries are to be open source. Check

http://www.optimizationservices.org

2. Libraries to be licensed under a non-copyleft license

3. Extensions include Networks, SDP, Quadratic, Complementarity, Stochastic Programming, Constraint/Logic/Combinatorial, General Nonlinear