LPFML: A W3C XML Schema for Linear and Integer Programming

Robert Fourer
Northwestern University
4er@iems.northwestern.edu

Leonardo Lopes
University of Arizona
leo@sie.arizona.edu

Kipp Martin
University of Chicago
kipp.martin@gsb.uchicago.edu

Denver INFORMS 2004

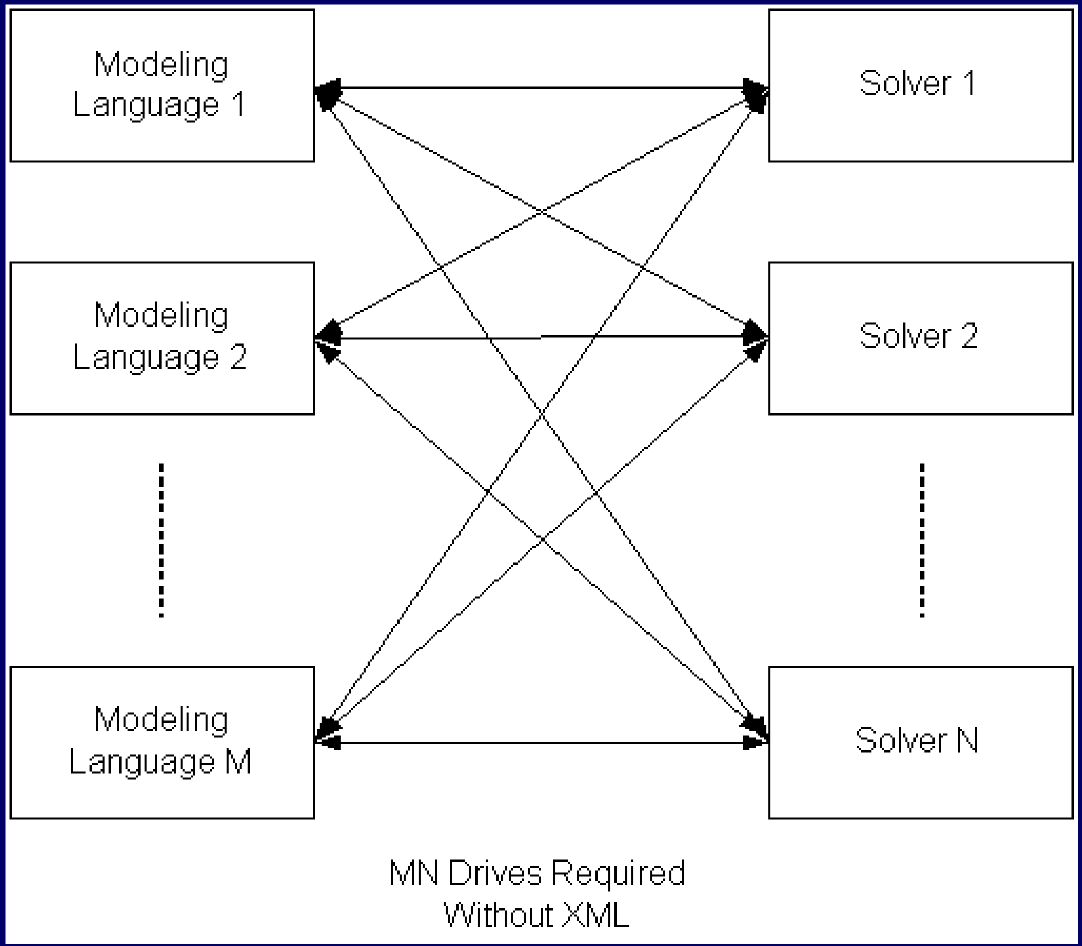
OUTLINE

1. Introduction and motivation
2. Model versus Instance
3. Why XML?
4. The LPFML Schema
5. Compression
6. The libraries
7. Testing under loosely and tightly coupled scenarios
8. The license

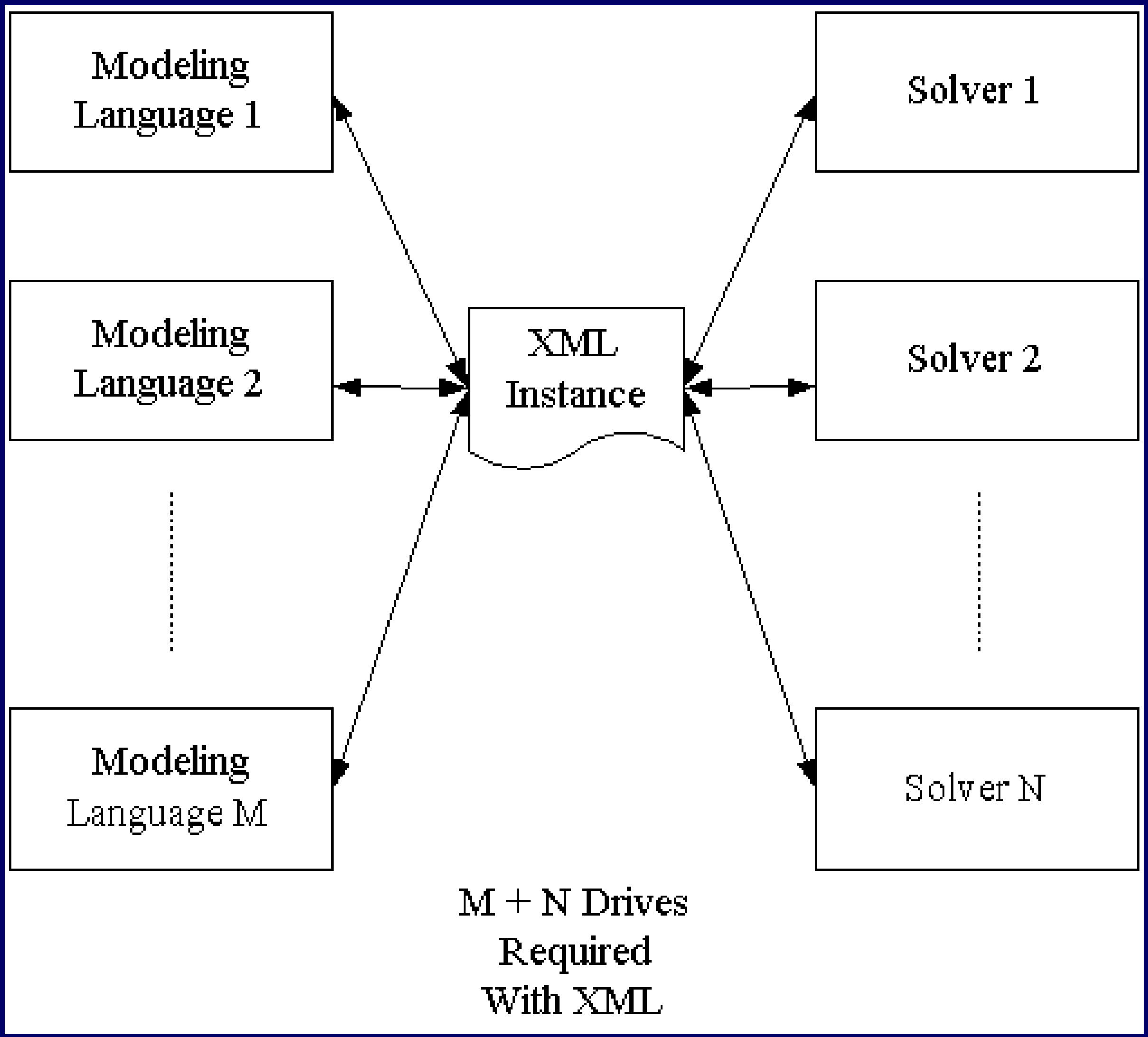
There is a proliferation of modeling languages and solvers

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



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];
```

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.

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

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.

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

ATTRIBUTE

ELEMENT

XML Concepts

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

Schema ⟷ **Class**

XML File ⟷ **Object**

We need a schema to represent an instance.

XML Concepts – a Row Class

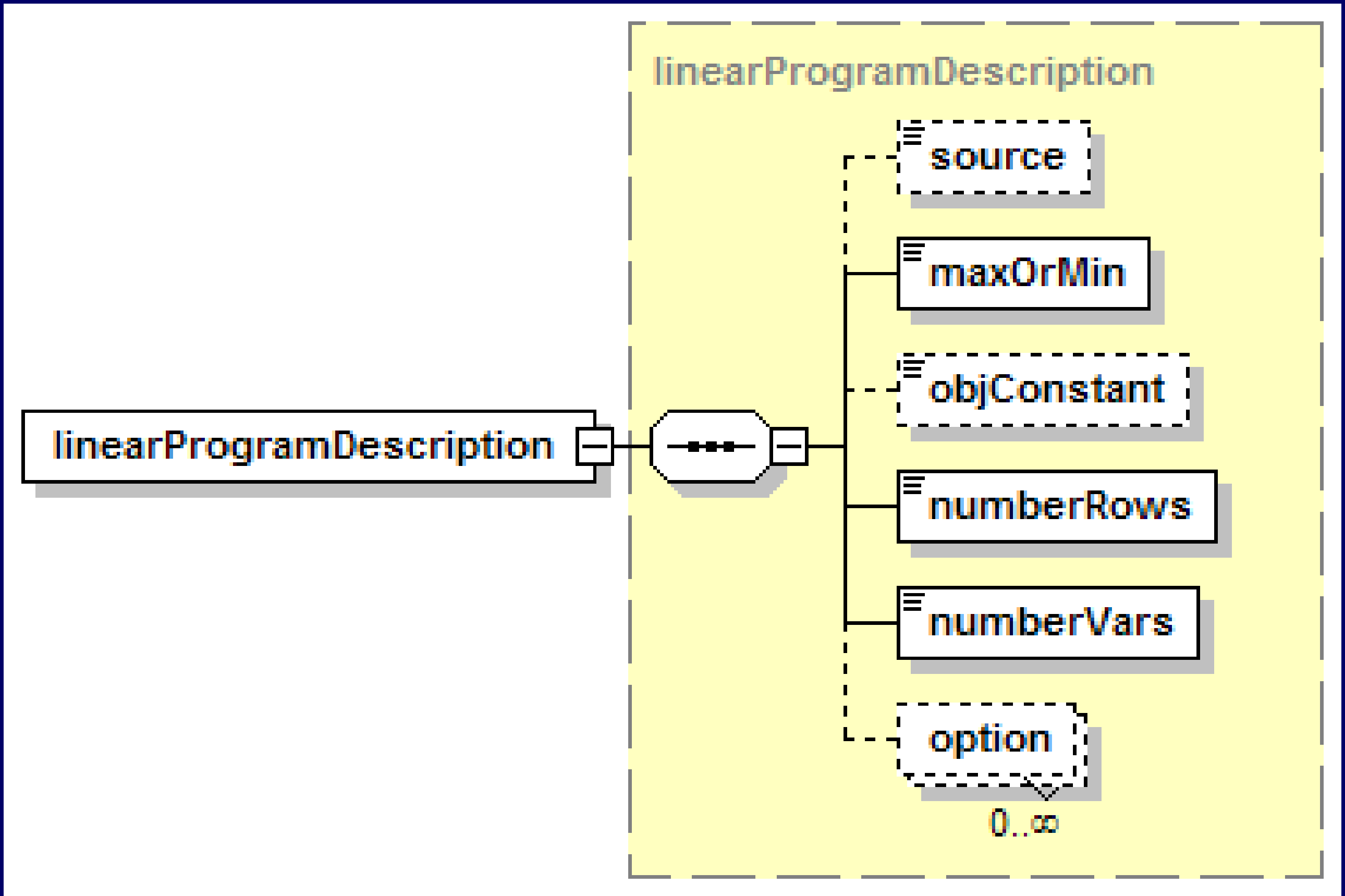
```
<xs:element name="rows">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="row" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="rowName" type="xs:string" use="optional"/>
          <xs:attribute name="rowUB" type="xs:double" use="optional"/>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

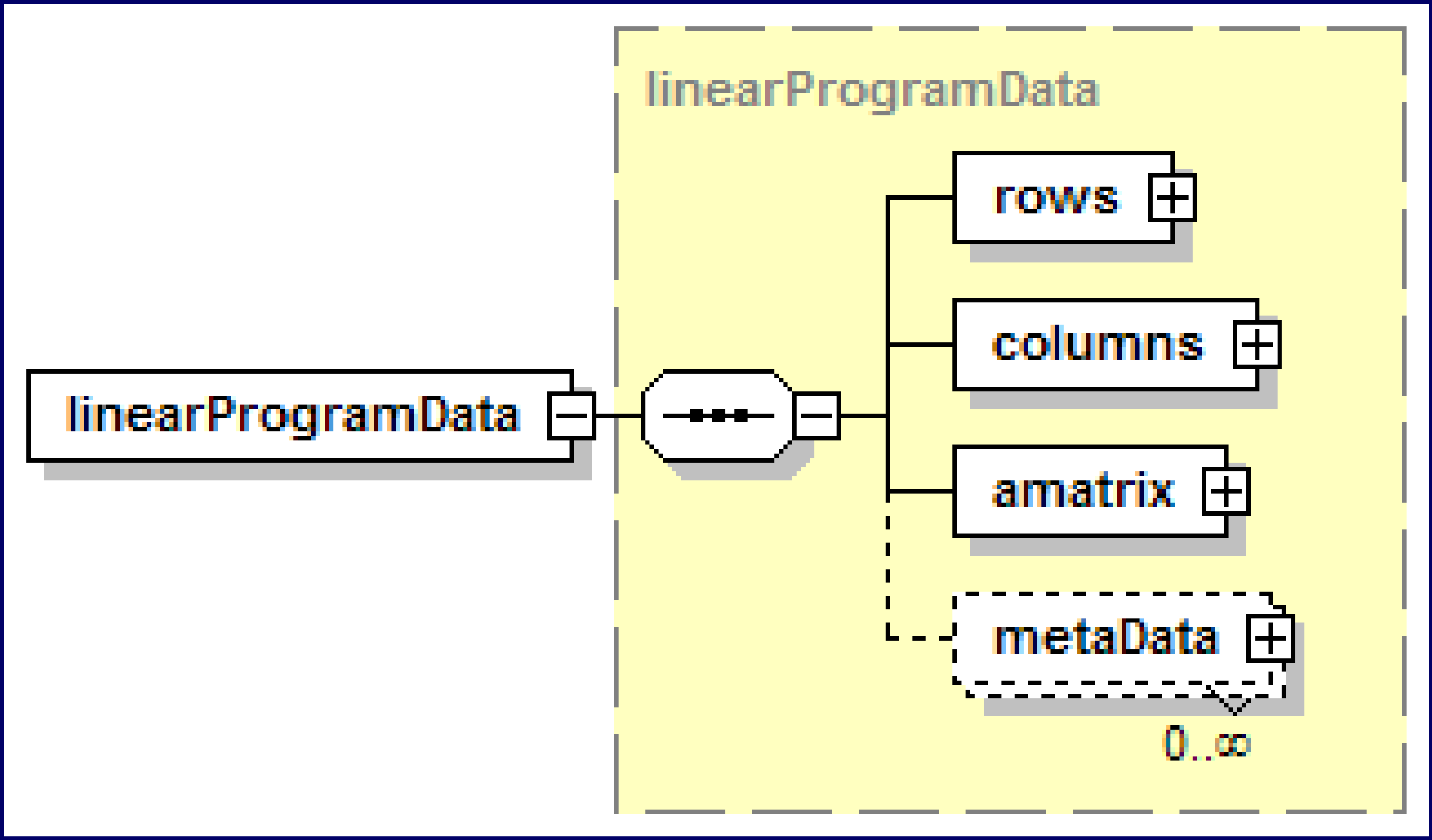
LPFML Schema

1. Information about the instance



LPFML Schema

2. The instance data



LPFML Schema

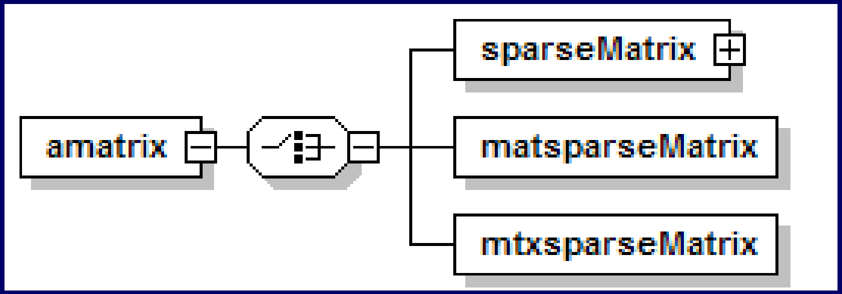
2. The instance data (rows and columns)

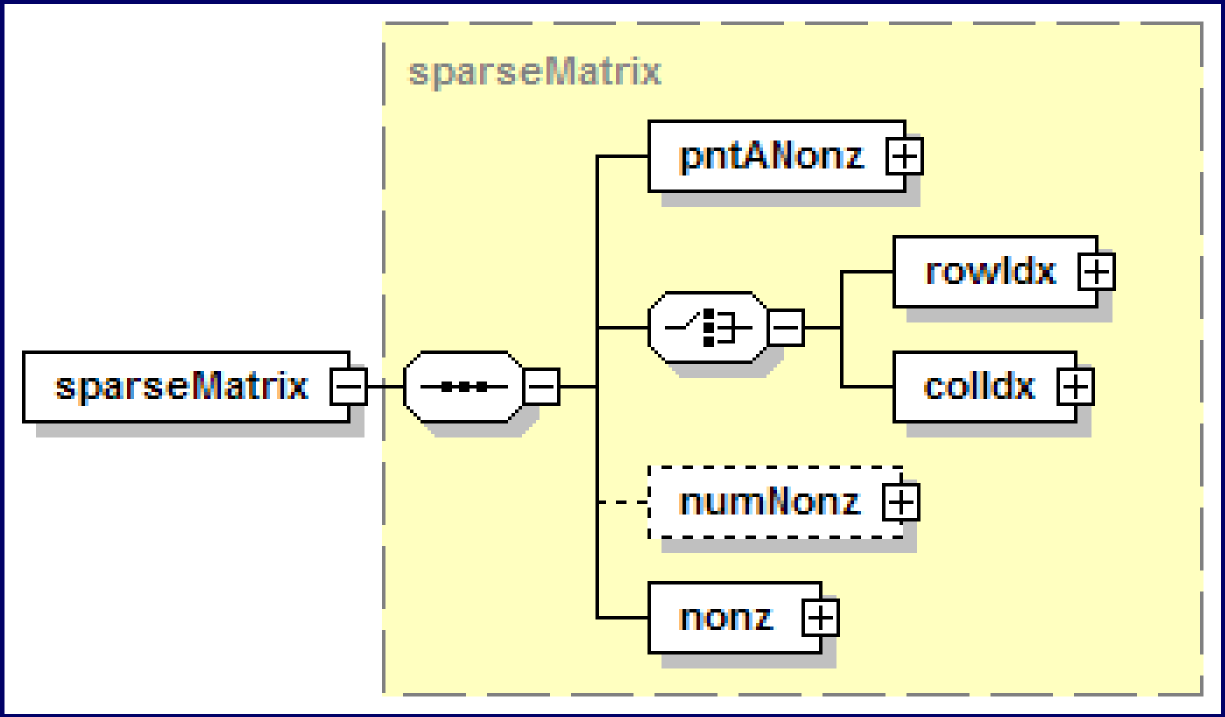
```
<rows>
  <row rowName="HoursAvailable['cutanddye']" rowUB="630"/>
  <row rowName="HoursAvailable['sewing']" rowUB="600"/>
  <row rowName="HoursAvailable['finishing']" rowUB="708"/>
  <row rowName="HoursAvailable['inspectandpack']" rowUB="135"/>
</rows>
```

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

LPFML Schema

2. The instance data (the A matrix)





LPFML Schema

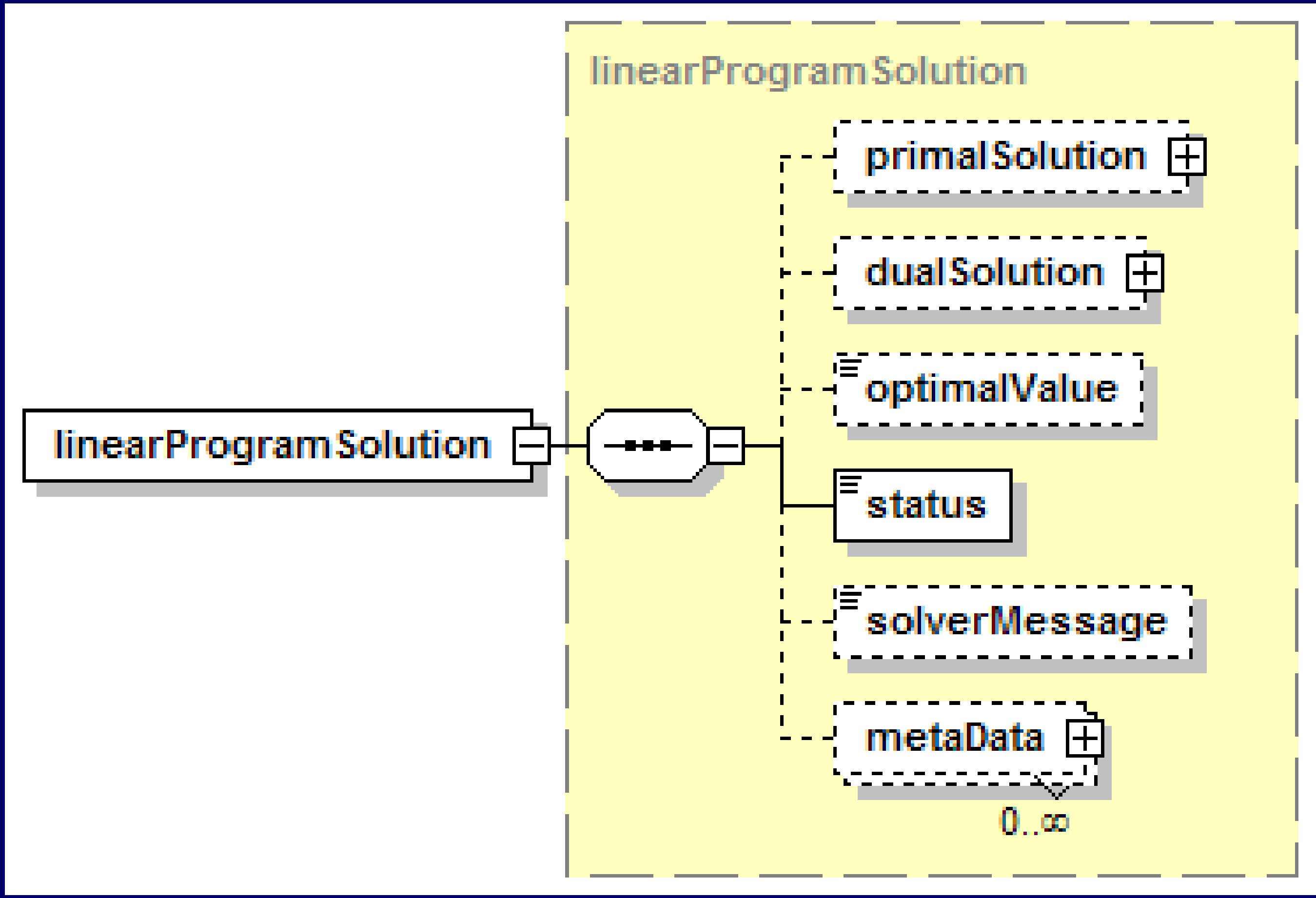
2. The instance data (the A matrix)

```
<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>
```

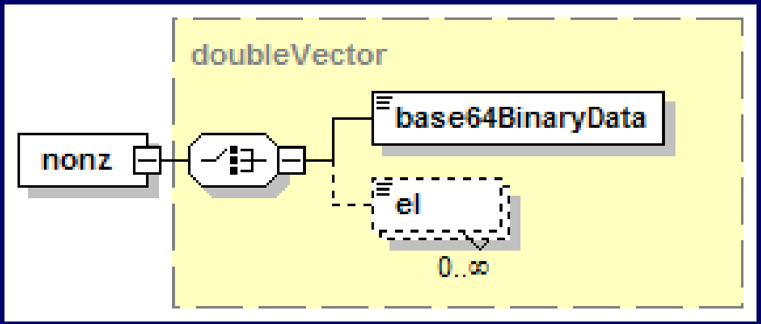
$$\begin{pmatrix} 0 & -3 \\ 1 & 4 \\ 2 & 0 \end{pmatrix}$$

LPFML Schema

3. Information about the solution



Compression – Base 64




```
<nonz>
  <el>.7</el><el>.5</el>
  <el>1.0</el><el>0.1</el>
  <el>1.0</el><el>0.8333</el>
  <el>0.6667</el><el>0.25</el>
</nonz>
```

```
<nonz>
<base64BinaryData numericType="double" sizeOf="8">
ZmZmZmZm5j8AAAAAAADgPwAAAAAAAPA/mpmZmZmZuT8AAAAAA
ADwP7U3+MJkquo/S8gHPZtV5T8AAAAAAADQPw==</base64BinaryData>
</nonz>
```

Compression – Structural

$$
\begin{array}{l}
\text{Min } x1 + x2 + x3 + x4 + x5 + x6 \\
\text{s.t.} \\
\begin{array}{rrrrrrl}
x1 & + x2 & & & & & \geq 1 \\
x1 & + x2 & & & & + x6 & \geq 1 \\
 & & + x3 & + x4 & & & \geq 1 \\
 & & + x3 & + x4 & + x5 & & \geq 1 \\
 & & & + x4 & + x5 & + x6 & \geq 1 \\
 & + x2 & & & + x5 & + x6 & \geq 1
\end{array}
\end{array}
$$

```
<nonz><el mult=“16”>1</el></nonz>
```

```
<rowIdx>
  <el>0</el><el>1</el><el>0</el><el>1</el>
  <el>5</el><el>2</el><el>3</el>
  <el mult="3" incr="1">2</el>
  <el mult="3" incr="1">3</el>
  <el>1</el><el>4</el><el>5</el>
</rowIdx>
```

The LPFML Libraries

A set of open source libraries for reading and writing LP instances in XML format.

Objective: hide all of the parsing and writing of XML in the library

Corollary: library users should only have to deal with optimization concepts such as objectives, constraints, etc.

Objective: allow changes and extensions to the schema without any solver or modeling language code to be rewritten

The LPFML Libraries - Parsing

Current C++ library based on SAX (Simple API for XML).

Our libraries use the Apache Xerces libraries.

Key classes are FMLHandler and FMLParser.

FMLHandler inherits from SAX2 ContentHandler.

The FMLHandler “gets the data” from the XML file when elements and attributes are read.

The LPFML Libraries - Parsing

FMLParser is solver independent. It instantiates an FMLHandler Object and passes to the constructor an FMLParser object.

handler = new FMLHandler(this, encodingName, expandNamespaces)

Provides numerous “dummy” methods:

onVariableCount(),
onObjectiveSense(),
onAMatrix(),
etc

The handler object calls these methods when reading elements and attributes.

The LPFML Libraries - Parsing

It is up to the user to implement their own solver specific FMLParser class. For example, in FMLLINDOParser we have

```
void FMLLINDOParser::onObjectiveSense(const bool isMin)
{
  if(isMin) nDir_ = LS_MIN;
  else nDir_ = LS_MAX;
  // make sure objConstant_ gets initialized
  objConstant_ = 0.;
}
```

There is no XML involved in writing the solver specific FMLParser that inherits from the FMLParser. It is hidden from the solver developer.

The LPFML Libraries - Parsing

Current open source FMLParses include:

OSI/COIN

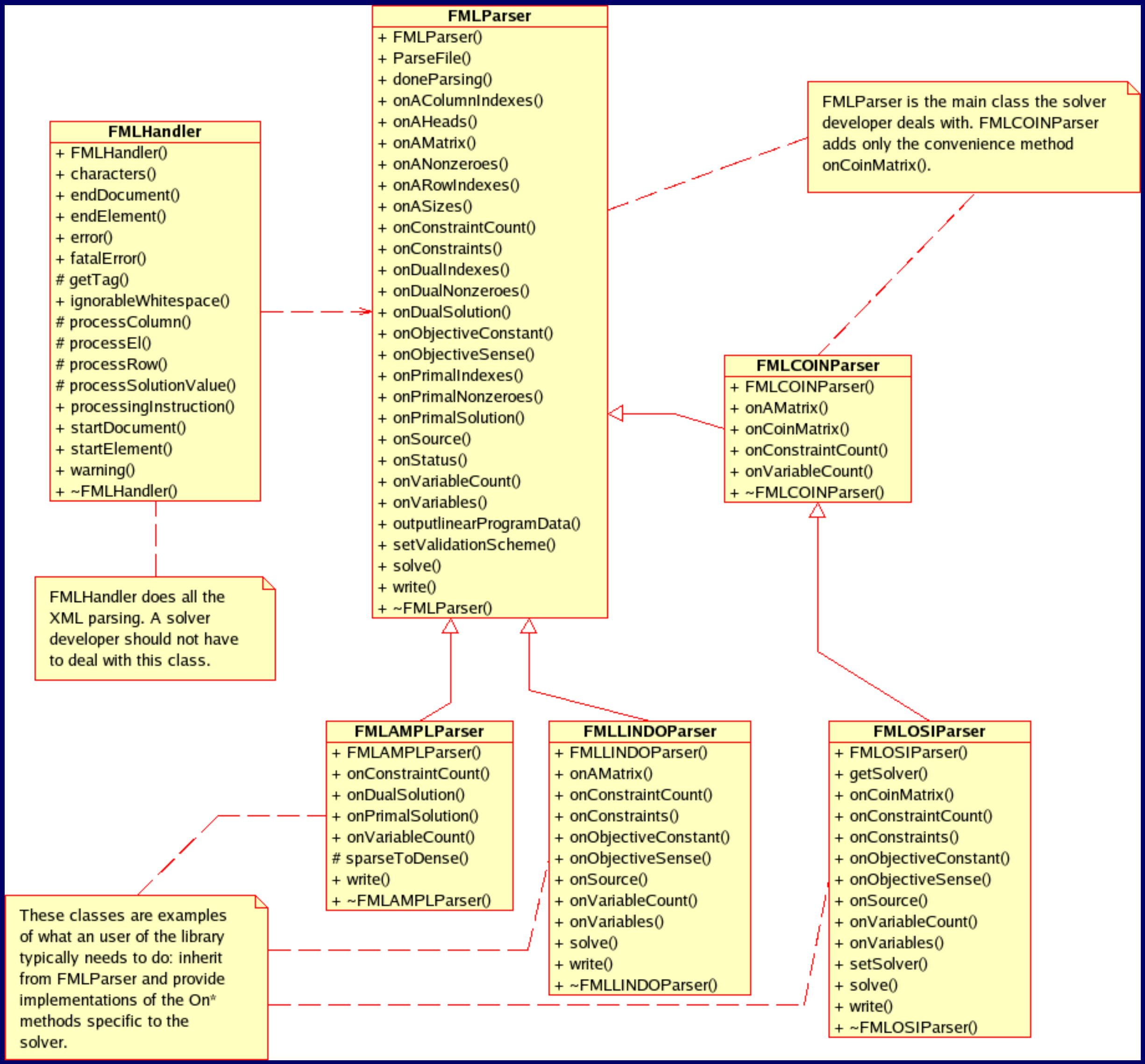
LINDO

FORTMP

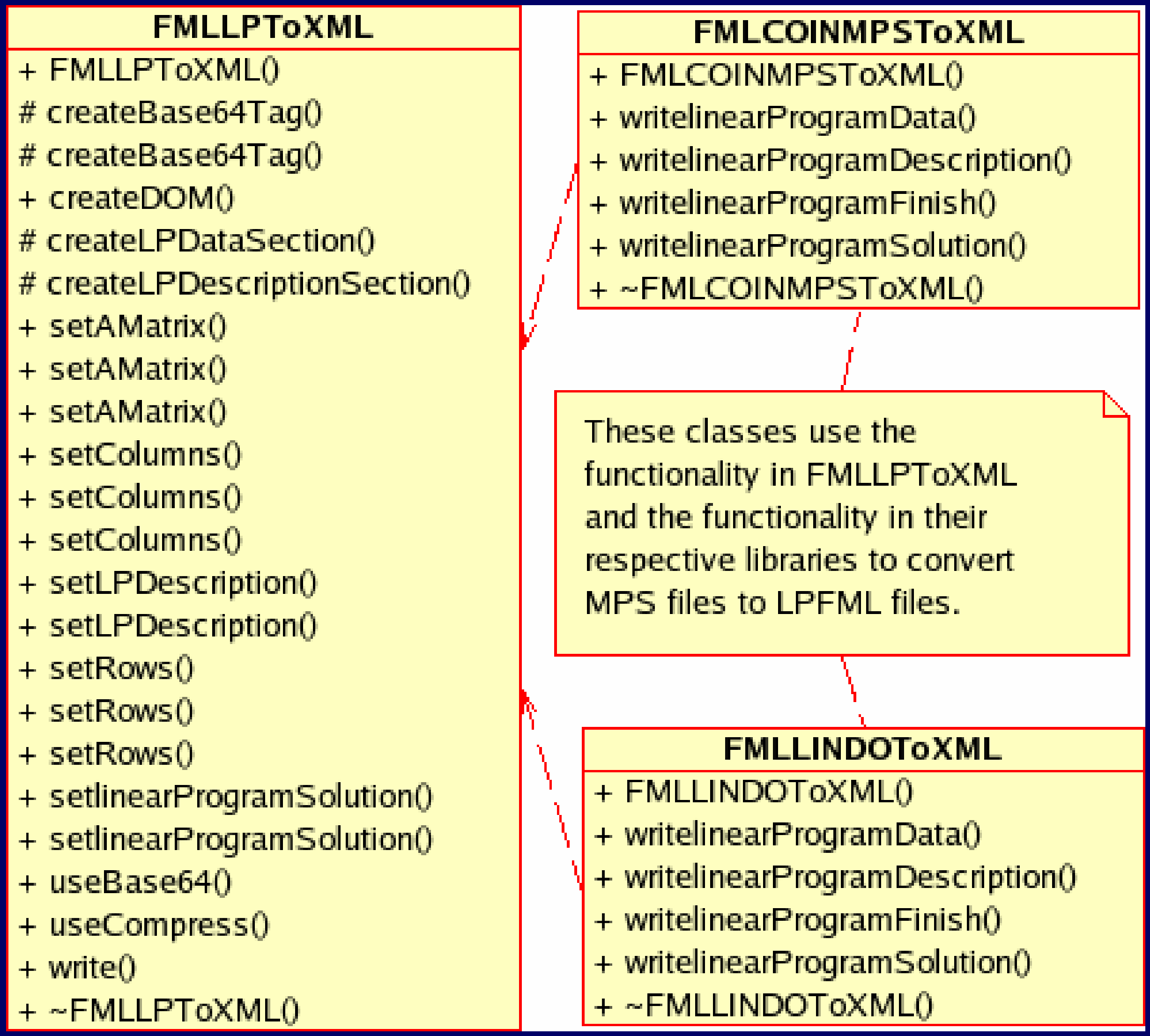
lp_solve (coming soon)

OSI/COIN – low level API for solvers. If a solver supports OSI/COIN they can read FML. For example, CLP and GLPK.

The LPFML Libraries - Parsing



The LPFML Libraries - Writing



LPFML - Testing

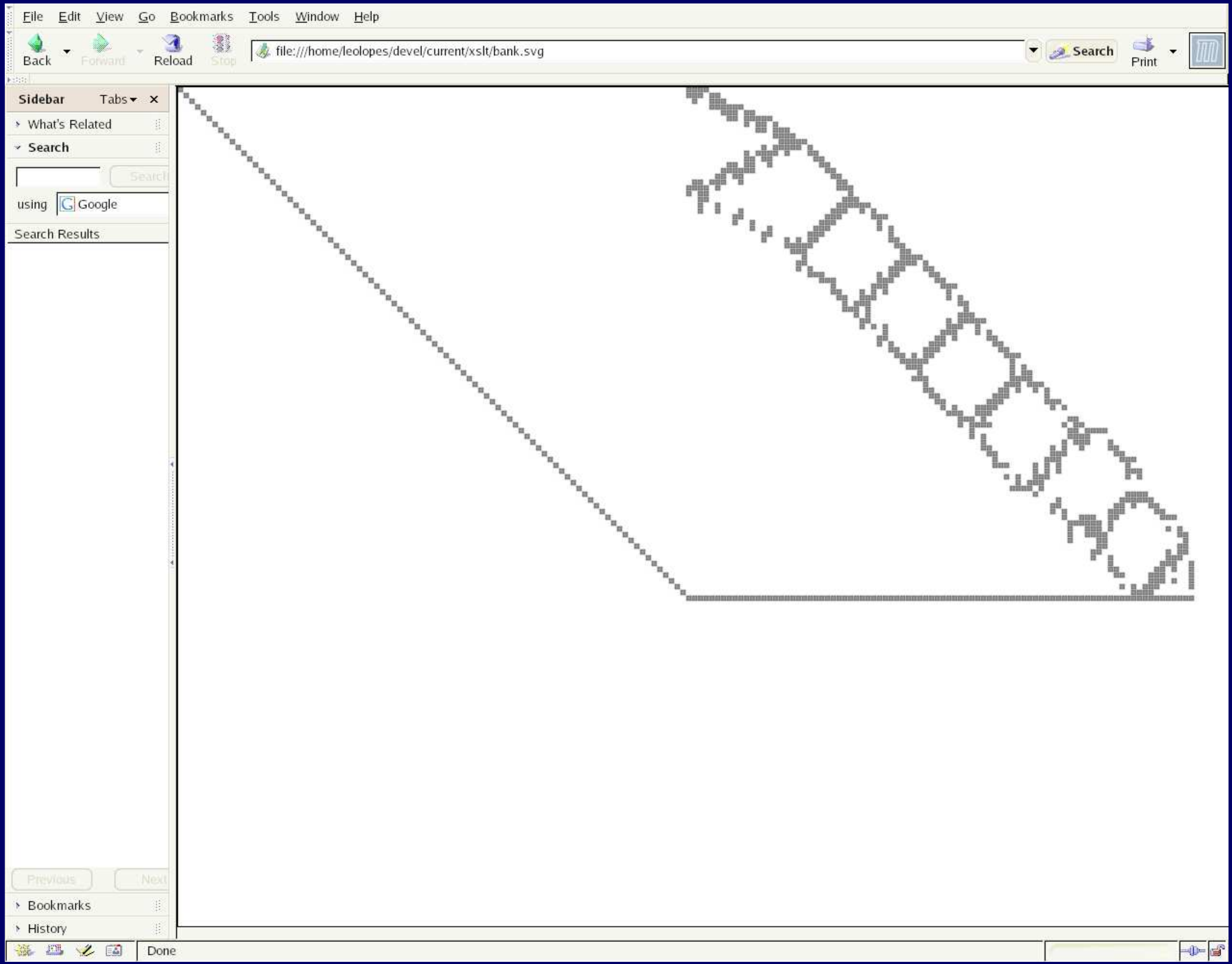
Basic scenarios:

1. Tightly coupled scenario where modeling system and solver communicate directly with each other on same machine. Performance important here.

2. Loosely coupled environments where modeling system and solver on different machines. File size important here.

3. Pre and post-processing environments. Flexibility key here.

Scenario – Pre and Post Processing



Scenario – Pre and Post Processing

Linear Program Solution

PRIMAL SOLUTION

Variable	Value
Make ['std']	539.984
Make ['del']	252.011

DUAL SOLUTION

Constraint	Dual Value
HoursAvailable['cutanddye']	4.37457
HoursAvailable['sewing']	0
HoursAvailable['finishing']	6.9378
HoursAvailable['inspectandpack']	0

Conclusion and Extension

1. The libraries are open source. See

http://gsbkip.chicagogsb.edu/fml/fml.html

2. Libraries licensed under a non-copyleft license

3. Obvious extensions in the linear area include networks, SOS, quadratic, and stochastic programming.

4. For more general extensions into nonlinear and optimizaton services framework see Jun Ma (SB16.4), “A Unified XML-Based Framework for Optimization Services.”