Optimization Services Modeling Language (OSmL)

Jun Ma
Northwestern University

Kipp Martin
University of Chicago

November 15, 2005
Outline

Introduction and Motivation

The OSmL Philosophy

OSmL Syntax

Data and XML

Hybrid Approached

OSmL GUI
Introduction and Motivation

The Objective: A native XML modeling language

- It should be able to act as an agent and send OSiL files to a server with a solver that implements Optimization Services
Introduction and Motivation

The Objective: A native XML modeling language

- It should be able to act as an agent and send OSiL files to a server with a solver that implements Optimization Services
- It should be a true algebraic modeling language
  1. take a general infix notation
  2. support sets and subscripts
  3. have looping capability
  4. support logical conditions
  5. allow for user-defined functions
  6. allow for sparse sets, union, intersection, etc.
Introduction and Motivation

The Objective: A native XML modeling language

- It should be able to act as an agent and send OSiL files to a server with a solver that implements Optimization Services
- It should be a true algebraic modeling language
  1. take a general infix notation
  2. support sets and subscripts
  3. have looping capability
  4. support logical conditions
  5. allow for user-defined functions
  6. allow for sparse sets, union, intersection, etc.
- Store model instances internally as an OSInstance object
Introduction and Motivation

The Objective: A native XML modeling language

- It should be able to act as an agent and send OSiL files to a server with a solver that implements Optimization Services
- It should be a true algebraic modeling language
  1. take a general infix notation
  2. support sets and subscripts
  3. have looping capability
  4. support logical conditions
  5. allow for user-defined functions
  6. allow for sparse sets, union, intersection, etc.
- Store model instances internally as an OSInstance object
- Also function as a matrix generator
XML is a key technology in industry

- XML is rapidly becoming an accepted format for transferring/storing data. This is where the data is! Think Willie Sutton and Sam Savage.

- People in IT use XML. Perhaps OR people should use IT tools, rather than having IT people use OR tools.

- Numerous open-source tools exist for manipulating XML files
Introduction and Motivation

There are four ways to incorporate XML in the mathematical modeling process:

- Use XML to represent the instance of a mathematical program
- Develop an XML modeling language dialect
- Enhance modeling languages with XML features such as XPath
- Use XML technologies to transform XML data into a problem instance
**Strategy 1:** Use XML to represent the instance of a mathematical model: e.g. LPFML and OSiL (Fourer, Kristjansson, Lopes, Ma, Martin, etc.).

If there are \( N \) modeling languages and \( M \) drivers you can go from \( M \times N \) drivers to \( M + N \) drivers.

**Strategy 2:** Use XML to represent the mathematical model, e.g. Ezechukwu and Maros (AML Algebraic Markup Language)

- With this approach we use XML tags to represent the algebraic model NOT the instance.
- This is a high level approach.
- Have tags for model constructs such as sets, variables, parameters, etc.
Strategic 2 (Continued): Use XML to represent the mathematical model, e.g. Ezechukwu and Maros (AML Algebraic Markup Language)

Potential Problems:

- How do we get everyone to agree? Witness the proliferation of modeling languages.
- XML is wordy and would lead to a very verbose language.
Introduction and Motivation

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<lotSizeData>
  <product productId="1" holdCost="1" prodCost="7" fixedCost="150">
    <period periodID="1">
      <demand>60</demand>
    </period>
    <period periodID="2">
      <demand>100</demand>
    </period>
    <period periodID="3">
      <demand>140</demand>
    </period>
    <period periodID="4">
      <demand>77.77</demand>
    </period>
  </product>
  <product productId="2" holdCost="2" prodCost="4" fixedCost="100">
    <period periodID="1">
      <demand>1</demand>
    </period>
    <period periodID="2">
      <demand>2</demand>
    </period>
    + <period periodID="3">
    + <period periodID="4">
  </product>
  <periodCapacity>
    <capacity periodID="1">200</capacity>
    <capacity periodID="2">200</capacity>
    <capacity periodID="3">200</capacity>
    <capacity periodID="4">200</capacity>
  </periodCapacity>
</lotSizeData>
```
Dynamic Lot Size Model:

\[
\min = \sum_{i=1}^{N} \sum_{t=1}^{T} (h_{it}l_{it} + f_{it}y_{it})
\]

\[
l_{i,t-1} + x_{it} - l_{it} = d_{it}, \quad i = 1, \ldots, N, \quad t = 1, \ldots, T
\]

\[
\sum_{i=1}^{N} x_{it} \leq c_t, \quad t = 1, \ldots, T
\]

\[
x_{it} \leq c_t y_{it}, \quad i = 1, \ldots, N, \quad t = 1, \ldots, T
\]
Introduction and Motivation

**Strategy 3:** Enhance current modeling languages with XML features such as XPath.

With **XPath** we can query an XML file and return a node set as an ordered sequence.

In AMPL we declare sets such as:

``` ampl
set PROD;
set LINKS = {PROD, 1..numPeriods};
param HC {PROD} ;
param FXC {PROD} ;
param CAP {1..numPeriods} ;
param DEM {LINKS};
```

Let's look at equivalent in XPath.
Introduction and Motivation

**Strategy 3:** Enhance current modeling languages with XML features such as XPath.

```plaintext
set PROD;
set LINKS = {PROD, 1..numPeriods};
param HC {PROD} ;
param FXC {PROD} ;
param CAP {1..numPeriods} ;
param DEM {LINKS};
```

Key Analogy: Create a built-in XPath Handler much like ODBC

table FXC IN XPath lotsizedata.xml
```
/lotSizeData/product/@fixedCost
```
The OSmL Philosophy: All X all the time!

**Key Premise**: OSmL is based on XQuery. Think of XQuery as a much more powerful SQL applied to XML data rather than relational data.

**SQL**:
- SELECT
- FROM
- WHERE

**XQuery (FLWOR flower)**:
- For
- Where
- Let
- Order by
- Return
The OSmL Philosophy: All X all the time!

**Key Premise**: XQuery (an extension of XPath) is a very powerful modeling language for mathematical optimization. It is (See Fourer 1983):

- symbolic
- general
- concise
- understandable

We can build a modeling language using existing W3C standards!
The OSmL Philosophy
Advantages of using XQuery:

- XQuery and XPath have very powerful algebraic modeling features, e.g. sets, for loops, if-then, union, intersection, library modules
- These are already accepted W3C standards
- Allow for concise model representation
- Lots of open source software tools are being written
- XQuery and XPath very amenable to distributed computing
- Easy problem analysis on OSrL
First Requirement: Take a model in infix format without any set notation.

return
<mathProgram>
<obj maxOrMin="min" name="Rosenbrock">
100*(x2 - x1^2)^2 + (1 - x1)^2
</obj>
<constraints>
<con>
x1 + x2 <= 100
</con>
</constraints>
</mathProgram>
Of course work with sets:

Here is some AMPL

```AMPL
param HC {PROD} ;
param FXC {PROD} ;
param CAP {1..numPeriods} ;
```

Here is some XQuery

```XQuery
let $HC := $products/@holdCost
let $FXC := $products/@fixedCost
let $CAP := $time/text()
```
OSmL Syntax

We can point to any number of data sets:

let \( \$products := \)
\[
\text{doc("/Users/kmartin/temp/osml/lotsizedata.xml")}
\]
\[
/\text{lotSizeData/product[ (1, 2, 5, 11, 17)]}
\]

let \( \$products := \)
\[
\text{doc("http://128.135.124.10/Users/kmartin/ ...")}
\]

We can also define variables:

let \( \$N := \text{count(\$products)} \)
let \( \$T := \text{count(\$time)} \)
We can do looping:

Here is AMPL

subject to demand \{i in PROD, t in 1..numPeriods\}:
   \[ X[i, t] + I[i, t - 1] - I[i, t] = DEM[i, t]; \]

Here is XQuery

for $i in PROD, $t in (1 to $T)
let $demand :=
$products[$i]/period[@periodID=$t]/demand/text()
return
<con name="demand[{$i},{$t}] > X({$i},{$t }) + I({$i},{$t - 1}) - I({$i},{$t}) =
{$demand}
</con>
OSmL Syntax

An AMPL and XQuery analogy:

AMPL: subject to
XQuery: <con>

AMPL: demand
XQuery: name="demand[{$i},${$t }]"
AMPL: {i in PROD, t in 1..numPeriods}
XQuery: for $i in (1 to $N), $t in (1 to $T)
let $demand :=
data($products[$i]/period[@periodID=$t]/demand)

AMPL: X[i, t] + I[i, t - 1] - I[i, t] = DEM[i, t]
XQuery: X({$i},{$t }) + I({$i},{$t - 1}) - I({$i},{$t}) = {$demand}
OSmL Syntax

XQuery evaluates what is in `{...}` and the $ tells XQuery you have a variable.

```xquery
{ for $i in $PROD, $t in (1 to $T)
  let $demand :=
  ($products[$i]/period[@periodID=$t]/demand/text())
  return
  <con name="demand[{$i},{$t}]"> X[{$i},{$t}] + I[{$i},{$t - 1}] - I[{$i},{$t}] =
  {$demand}
  </con> }
```
The XQuery results is:

<!-- DEMAND CONSTRAINTS -->
<con name=demand[1,1]>
    X(1,1) + I(1,0) - I(1,1) = 60
</con>
<con name=demand[2,1]>
    X(2,1) + I(2,0) - I(2,1) = 1
</con>

Etc

We then parse this and transform into OSiL.
OSmL Syntax

Other features:

- Use if-then logic
- We can use built-in Java functions. For example:
  
  ```
  declare namespace math="java:java.lang.Math";
  ```
- We can define our own functions
- Use XQuery and XPath to display results
- Define sparse sets, intersection, union

```xml
let $products := /lotSizeData/product[ (1, 2, 11, 19)]
for (1 to 100)[mod 2 eq 0]
```
Data and XML

Point 1: It is getting easy to get data in XML format from traditional sources

- Can export to XML from desktop software (Microsoft Office)
- Can query an enterprise database in SQL and get result as XML

Point 2: There is even a trend toward native XML databases

- Total XML  Cincom
- Tamino  Software AG
- Apache  Xindice
- Cognetic Systems’ solutions
- Ipedo
Hybrid Approaches

Possibilities:

- Make XQuery/XPath equivalent to ODBC/SQL
- Introduce the concept of a node set (as an alternative) to a table in algebraic modeling languages
- What about adding XQuery syntax to the an algebraic modeling language?

Perhaps all algebraic modeling languages could have a common underlying syntax based upon XQuery/XPath?
The current implementation of OSmL is in OSmL GUI. It can be used in three ways:

- A simple agent to send OSiL to a Web server.
- Use XQuery and our parser to turn OSmL into OSiL
- With the OSInstance class as a matrix generator
The OSInstance class is used to access the problem data or create/modify the problem. For example, accessing a problem for the solver

```cpp
m_mdVarLB = osinstance->getVariableLowerBounds();
m_mdVarUB = osinstance->getVariableUpperBounds();
solver->assignProblem(m_, m_mdVarLB, m_mdVarUB,
m_mmdObjDenseCoefValue, m_mdConLB, m_mdConUB);
```

or creating a problem

```cpp
instanceData.linearConstraintCoefficients.start.el
= A_colstarts;
instanceData.linearConstraintCoefficients.value.el
= A_vals;
instanceData.linearConstraintCoefficients.rowIdx.el
= A_rownos;
```

**Key Idea:** It maps to the OSiL Schema.