Optimization Services Instance Language (OSiL), Solvers, and Modeling Languages

Robert Fourer
Jun Ma
Northwestern University
Kipp Martin
University of Chicago

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

Outline

1. Motivation and problem description

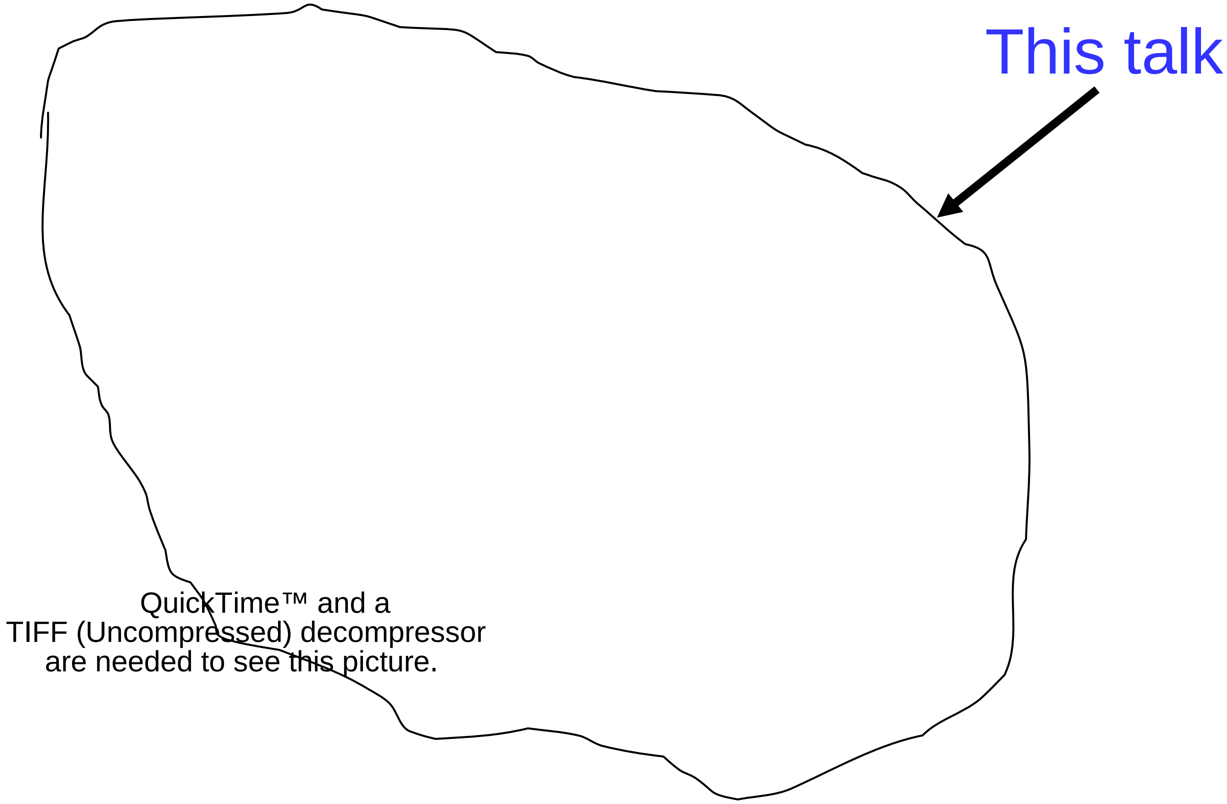
2. Instance and solver communication (APIs)

3. OSiLHandler and OSiLReader classes

4. Solver Interfaces

5. Concluding Remarks

The Problem


This talk
QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

The Problem

Brief Review:

OSiL: Optimization Services instance Language. This is an XML based format for representing a wide variety of optimization problems.

OSrL: Optimization Services result Language. An XML based format for representing the solution to optimization problems.

The Problem

We are in a loosely-coupled environment.

Solver and modeling language are separate process, possibly on separate machines.

There are lots of solvers, both linear and nonlinear.

Given a common instance format (OSiL), how do we communicate the instance format to solvers?

The Problem

How is communication done?

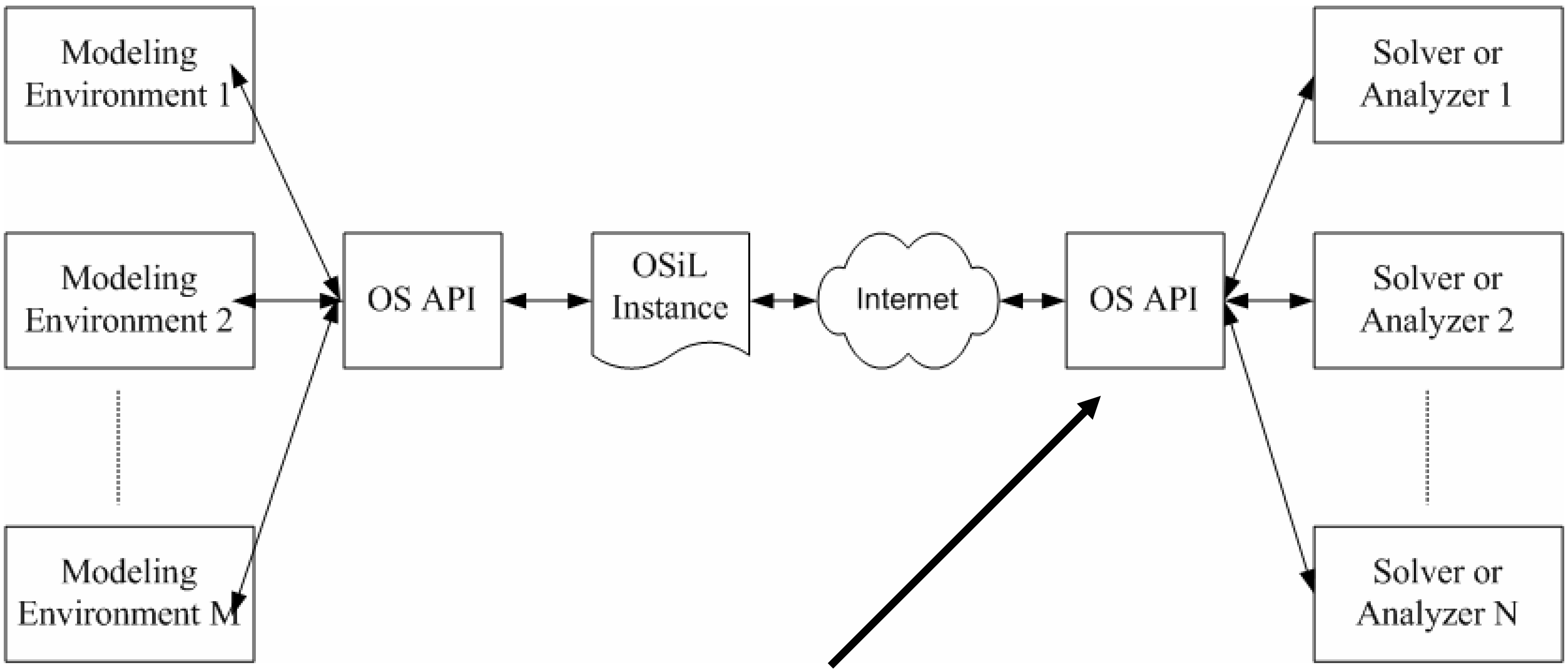
Through an Application Program Interface (API)

Think of an API as a specification for methods.

The methods then interact with an underlying data structure.

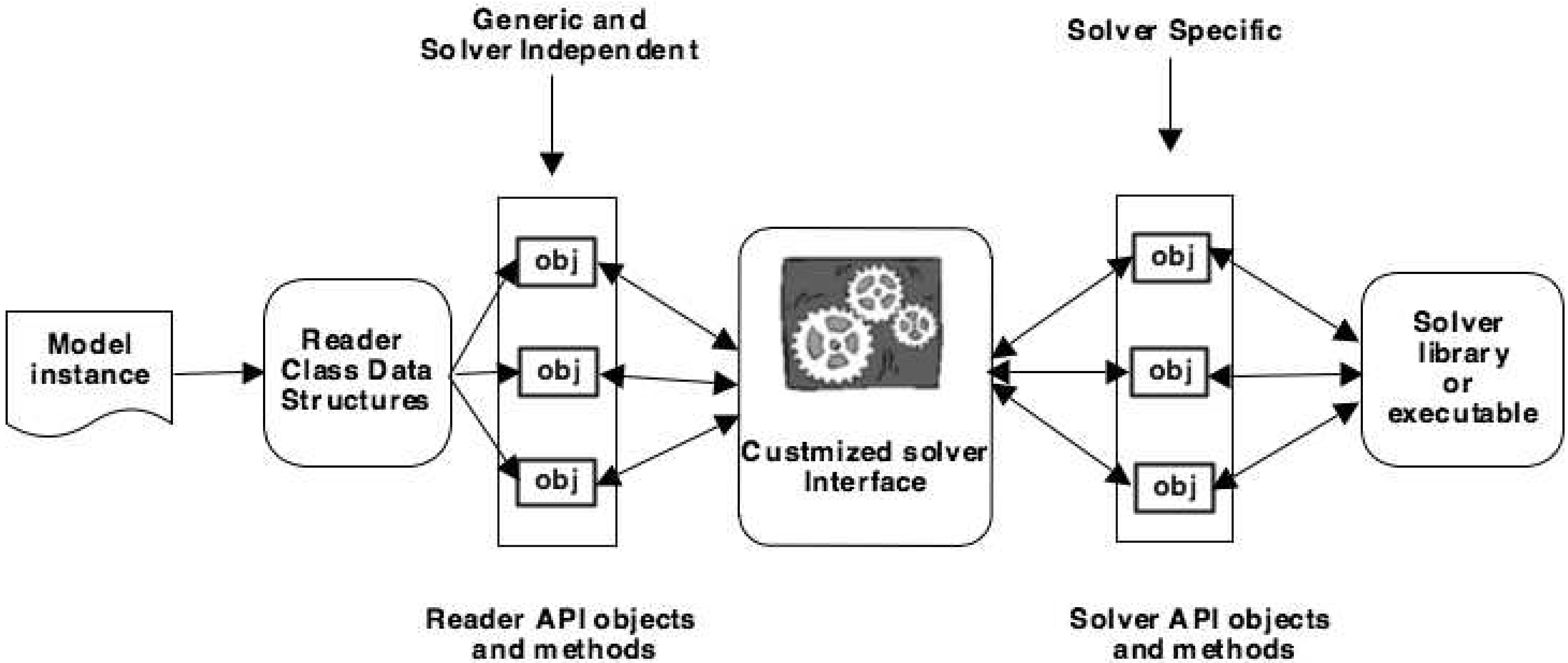
In the case of OSiL it is our OSExpressionTree

The Problem



Our Focus - the solver side API

Instance and Solver Communication



Instance and Solver Communication

Related work: "Simple C-API Windows DLL Implementation Of CLP, CBS, and CGL" by Bjarni Kristjannon (TD-14)

We also provide libraries with the following features:

1. Our libraries designed to read OSiL(replacing LPFML)

2. Not Windows based (Windows, Linux, and Mac)

3. Designed for a loosely-coupled environment

4. We provide libraries to read the instance and the solver specific interface libraries.

Key Library Components

OSiLHandler and **OSiLReader** are the key **solver independent** classes.

OSiLHandler -- a class that is designed to parse the XML OSiL file. The solver never sees this. The solver does not need to know anything about XML or OSiL.

OSiLReader -- a class that creates the necessary data structures and provides the API for the solver specific interface library.

The OSiLHandler Class

The OSiLHandler Class is designed to parse the XML.

There are two philosophies for this: SAX and DOM

SAX -- event based (data does not persist)

DOM -- tree based (data persists)

We have a C++ SAX based implementation for linear OSiL and a Java based DOM implementation for general OSiL

Our implementations uses the Apache Xerces libraries

The SAX OSiLHandler Class

SAX is event based. For example:

Reading the start of an XML element

Reading the end of an XML element

Reading character data in an XML element

Reading XML attributes

The Xerces parser has a default handler that
does nothing when these events are fired. The OSiLHadler
extends the Xerces base class and actually does something.

The SAX OSiLHandler Class

```
<variables number="2">
        <var  name="x1" type="C" lb="0.0"/>
        <var  name="x2" type="C" lb="0.0"/>
</variables>
```

```
void OSiLHandler::startElement( a bunch of parameters)

case var:
        processVar(attributes);
break;
```

processVar(attributes) get the information about the Variable, e.g. name, type, ub, lb and puts into a vector -- again the library user never sees this

After the last var element is read, <variables> processed

OSiLReader Class

Two key functions:

1. Use the OSiLHandler to create the data structures

2. Provide the methods that constitute the API

OSiLReader Data Structures

Linear part of model: arrays for the constraint matrix, e.g.

```
double* m_mdValueCoefMatrix;
int* m_miStartCoefMatrix;
int* m_miIdxCoefMatrix;
```

Nonlinear part of model: an expression tree

Arrays for variable and row information (e.g. lbs and ubs)

OSiLReader OSExpression Tree

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

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

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
```

"OSnLNodeTimes"

An instance of OSnLNode which is an OSnLNodeTimes

OSiLReader C++ API

The OSiLHandler instantiates an osilreader object and calls OSiLReader "on" methods when certain events "fire"

```
<variables number="2">
        <var  name="x1" type="C" lb="0.0"/>
        <var  name="x2" type="C" lb="0.0"/>
</variables>
```

```
void OSiLHandler::endElement( a bunch of parameters)

case variables:

osilreader_->onVariables(variables_,lb_,ub_,colDomain_)
```

OSiLReader API

The C++ OSiLReader is very flexible and provides two APIs

There are two strategies for the API:

Strategy I -- a pull strategy with get() methods

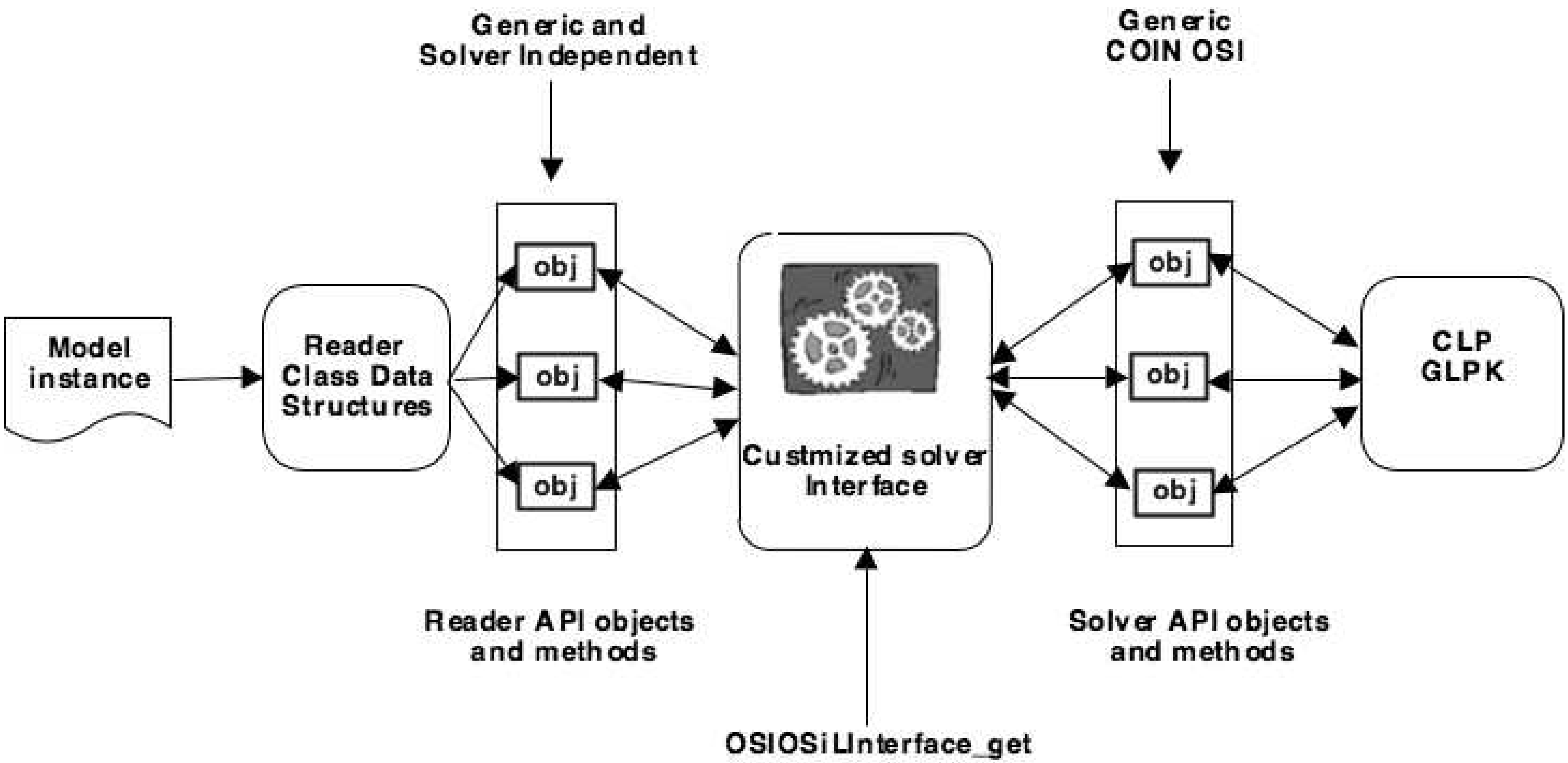
Strategy II -- an event based strategy that re-implements the base case "on" methods

OSiLReader C++ API Strategy 1 - Pull

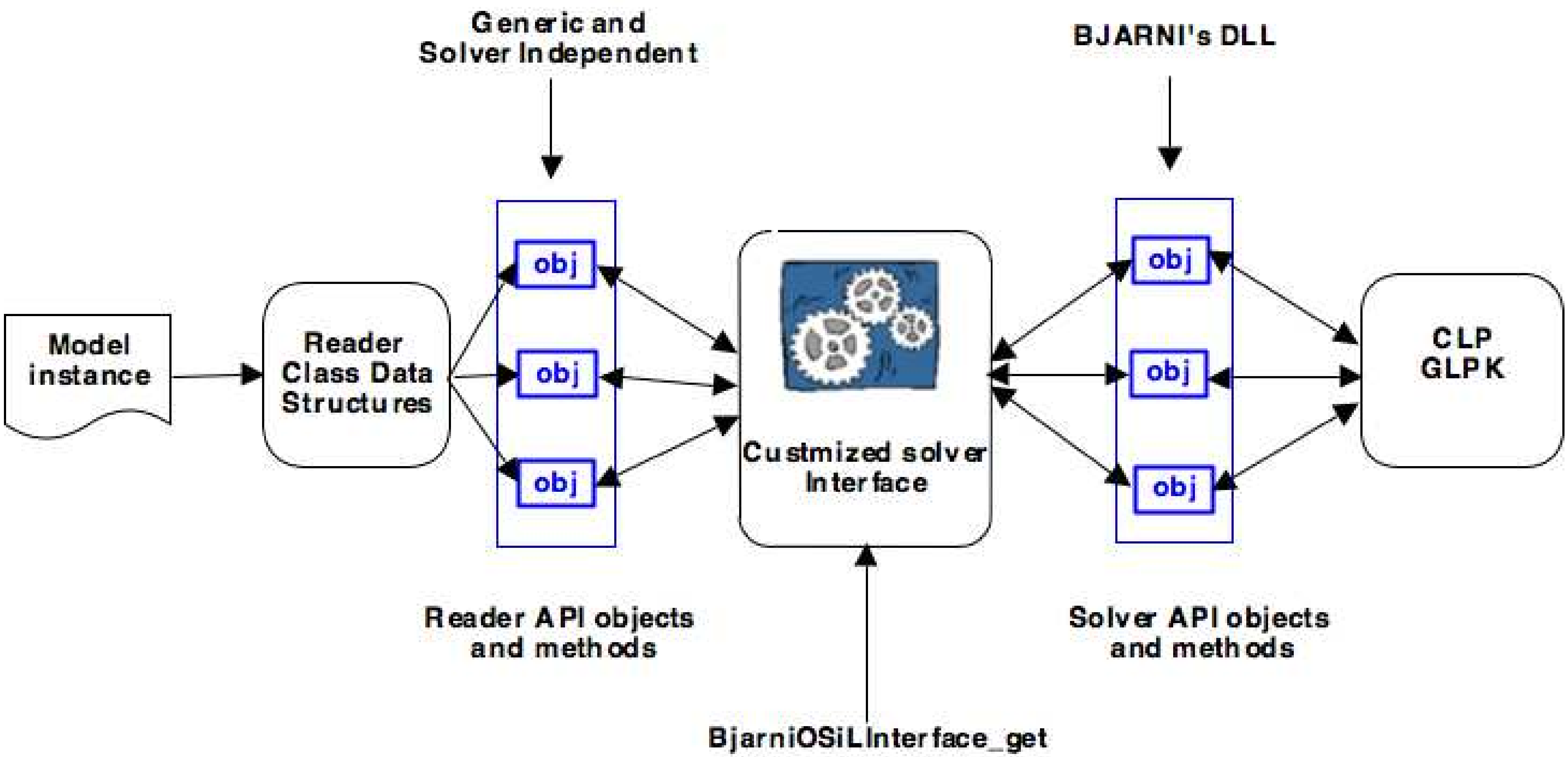
```
int OSiLReader::onVariables( parameters -- data from
OSiLHandler){

m_mdVarLB = new double[m_iNumberVariables];
m_mdVarUB = new double[m_iNumberVariables];
/* code to fill in the arrays */
}

double* OSiLReader::getVariableUBs(){
return m_mdVarUB;
}
```

So the API with Strategy I is a bunch of get() methods.
e.g., `getVariableLBs(), getConstraintUBs(), getMatrixNonzeroValues(),` etc

OSiLReader C++ API Strategy 1 - Pull



OSiLReader C++ API Strategy 1 - Pull



OSiLReader C++ API Strategy 1 - Pull

```
OSiLReader() osilreader;

m_mdVarLB = osilreader.getVariableLBs();
m_mdVarUB = osilreader.getVariableUBs();



solver_->assignProblem(m_, m_mdVarLB, m_mdVarUB,
          m_mmdObjDenseCoefValue, m_mdConLB, m_mdConUB);
```

Either a glpk or clp solver determined by the user at runtime.

We do a similar thing for the LINDO solver

OSiLReader C++ API Strategy 2 - Event

The on methods in OSiLReader are virtual. Define
A class that derives from this base clase with a new
Implementation of the on methods.

```
int OSiLOSIParser::onVariables( a bunch of parameters) {
lb_ = new double[nVars_];
ub_ = new double[nVars_];
std::copy(lb.begin(), lb.end(), lb_);
std::copy(ub.begin(), ub.end(), ub_);
```

```
solver_->assignProblem(m_, lb_, ub_, obj_, lhs_, rhs_);
```

OSiLReader Java Implementation Based on OSExpressionTree

This is pull oriented. A set of get() and calculate() methods.

getConstraintLBs()
getFirstObjectiveMaxOrMin()
getMatrixNonzeroIndexes()

getNonlinearPostfix(int rowIdx)
getNonlinearPrefix(int rowIdx)
getNonlinearInfix(int rowIdx)

calculateFunction(int rowIdx, double x[])
calculateNonlinearDerivatives(int rowIdx, double x[], boolean functionEvaluated)

OSiLReader Java API

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

OSiLReader Java API

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

```
<nl idx="1">
    <plus>
        <plus>
            <ln>
                <times>
                    <var coef="1.0" idx="1"/>
                    <var coef="1.0" idx="0"/>
                </times>
            </ln>
            <var coef="7.0" idx="1"/>
        </plus>
        <var coef="5.0" idx="0"/>
    </plus>
</nl>
```

OSiLReader Java API

Call getNonlinearPostfix(0)

This methods uses our OSExpressionTree data structure

postfix: [X1, X0, times, ln, 7.0, X1, times, plus, 5.0, X0, times, plus, 10, minus]

The Lindo Interface converts the postfix to a Lindo instruction list.

1063 1 1063 0 1003 1021 1062 16 1063 1 1003 1001 1062 17 1063 0 1003 1001

OSiLReader Java API

QuickTime™ and a
TIFF (Uncompressed) decompressor
are needed to see this picture.

Supported Platforms

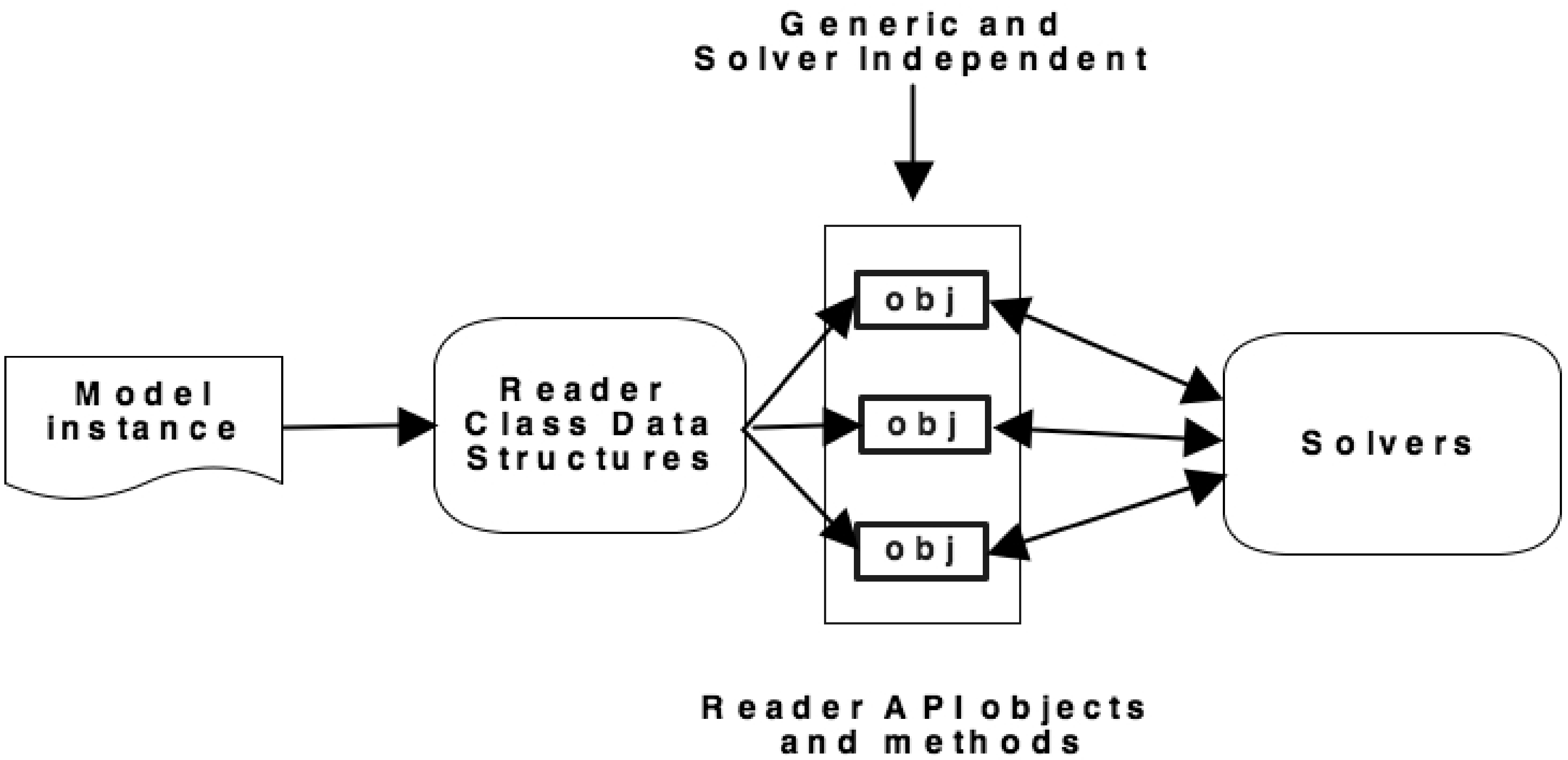
Modeling Languages that can generate OSiL (or LPFML):

AMPL (linear OSiL -- LPFML)
OSmL (native)
POAMS (native linear OSiL)

Solvers:

CLP - through COIN OSI
FORTMP - LPFML
GLPK - through COIN OSI
IMPACT - native support
KNITRO - using function callback
LINDO - using instruction list format

An Ideal World



QUESTIONS?

http://www.optimizationservices.org