Optimization Services 1.0 User's Manual

Robert Fourer, Jun Ma, Kipp Martin

June 21, 2007

Abstract

This is the User's Manual for the Optimization Services (OS) project. The objective of (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site `www.optimizationservices.org` and the COIN-OR Trac page `projects.coin-or.org` for more information.

Contents

List of Figures

List of Tables

1 Introduction

The objective of Optimization Services (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. This COIN-OR project provides source code for libraries and executable programs that implement OS standards. See the Optimization Services (OS) Home Site `www.optimizationservices.org` and the COIN-OR Trac page `projects.coin-or.org` for more information. The OS project provides the following:

1. A set of XML based standards for representing optimization instances (OSiL), optimization results (OSrL), and optimization solver options (OSoL). There are other standards, but these are the main ones. The schemas for these standards are described in Section 3.1.

2. A robust solver and modeling language interface (API) for linear and nonlinear optimization problems. Corresponding to the OSiL problem instance representation there is an in-memory object, `OSInstance`, along with a set of `get()`, `set()`, and `calculate()` methods for accessing and creating problem instances. This is a very general API for linear, integer, and nonlinear programs. Any modeling language that can produce OSiL can easily communicate with any solver that uses the OSInstance API. The `OSInstance` object is described in more detail in Section 5. The nonlinear part of the API is based on the COIN project `projects.coin-or.org/CppAD` by Brad Bell but is written in a very general manner and could be used with other algorithmic differentiation packages. More detail on algorithmic differentiation is provided in Section 6.

3. A command line executable `OSSolverService` for reading problem instances (OSiL format, nl format, MPS format) and calling a solver either locally or on a remote server. This is described in Section 7.

4. Utilities that convert AMPL nl files into the OSiL XML format and MPS files into the OSiL XML format. This is described in Section 4.3.

5. Standards that facilitate the communication between clients and optimization solvers using Web Services. In Section 4.1 we describe the `OSAgent` part of the OS library that is used to create Web Services SOAP packages with OSiL instances and contact a server for solution.

6. An executable program `amplClient` that is designed to work with the AMPL modeling language. The `ampClient` appears as a "solver" to AMPL and, based on options given in AMPL, contact solvers either remotely or locally to solve instances created in AMPL. This is described in Section 9.1.

7. Server software that works with Apache Tomcat and Apache Axis. This software uses Web Services technology and acts a middleware between the client that creates the instance and solver on the server that optimizes the instance and returns the result. This is illustrated in Section 8

2 Download and Installation

OS is released as open source code under the Common Public License (CPL). This project was created by Robert Fourer, Jun Ma, and Kipp Martin. The code has been written primarily by Jun Ma, Kipp Martin, Robert Fourer, and Huanyuan Sheng. Jun Ma and Kipp Martin are the COIN

project leaders for OS. Below we describe different methods for obtaining the C++ source code and binaries.

2.1 Obtaining the Source Code Subversion Repository (SVN)

The C++ source code can be obtained using Subversion. Users with Unix operating systems will most likely have an svn client. For Windows users wishing to obtain and SVN client we recommend TortoiseSVN. See `tortoisesvn.tigris.org`.

The OS project page with a Wiki is available at `projects.coin-or.org\OS`. Execute the following steps to get the source code using SVN.

Step 1: Connect to a directory where you want the OS project to go. The following command will download the project into the directory COIN-OS

```
svn co https://projects.coin-or.org/svn/OS/stable/1.0 COIN-OS
```

Step 2: Connect to the distribution root directory.

```
cd COIN-OS
```

Step 3: Run the configure script that will generate the makefiles.

```
./configure
```

Step 4: Run the make files.

```
make
```

Step 5: Run the unitTest.

```
make test
```

Depending upon which third party software you have installed, the result of running the unitTest should look something like:

```
HERE ARE THE UNIT TEST RESULTS:

Solved problem avion2.osil with Ipopt
Solved problem HS071.osil with Ipopt
Solved problem rosenbrockmod.osil with Ipopt
Solved problem parincQuadratic.osil with Ipopt
Solved problem parincLinear.osil with Ipopt
Solved problem callBack.osil with Ipopt
Solved problem callBackRowMajor.osil with Ipopt
Solved problem parincLinear.osil with Clp
Solved problem p0033.osil with Cbc
Solved problem rosenbrockmod.osil with Knitro
Solved problem callBackTest.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem parincQuadratic.osil with Knitro
Solved problem p0033.osil with SYMPHONY
Solved problem parincLinear.osil with DyLP
Solved problem lindoapiaddins.osil with Lindo
```

```
Solved problem rosenbrockmod.osil with Lindo
Solved problem parincQuadratic.osil with Lindo
Solved problem wayneQuadratic.osil with Lindo
Test the MPS -> OSiL converter on parinc.mps usig Cbc
Test the AMPL nl -> OSiL converter on hs71.nl using LINDO
Test a problem written in b64 and then converted to OSInstance
Successful test of OSiL parser on problem parincLinear.osil
Successful test of OSrL parser on problem parincLinear.osrl
Successful test of prefix and postfix conversion routines on problem rosenbrockmod.osil
Successful test of all of the nonlinear operators on file testOperators.osil
Successful test of AD gradient and Hessian calculations on problem CppADTestLag.osil


CONGRATULATIONS! YOU PASSED THE UNIT TEST
```

If you do not see

```
CONGRATULATIONS! YOU PASSED THE UNIT TEST
```

then you have not passed the unitTest and hopefully some semi-inteligble error message was given.

Step 6: Install the libraries.In addition you will have the following directories.

```
make install
```

This will install all of the libraries in the `lib` directory under the distribution root. In particuar, the main OS library `libOS` along with the libraries of the other COIN-OR project that download with the OS project will get installed in the `lib` directory. In addtion the `make install` command will install four executable programs in the `bin` directory. One of these binaries is `OSSolverService` which is main OS project executable. This is described in Section 7. In addition `clp`, `cbc`, `cbc-generic`, and `symphony` get installed in the `bin` directory.

2.2 Obtaining the Source Code From a Tarball or Zip File

The OS source code can also be obtained from either a tarball or zip file. This may be preferred for users who are not managing other COIN-OR projects wish to only work with periodic release versions of the code. In order to obtain the code from a Tarball or Zip file do the following.

Step 1: In a browser go the link `http://www.coin-or.org/Tarballs/OS/`. Listed at this page are files in the format:

```
OS-release_number.tgz
OS-release_number.zip
```

Step 2: Click on either the `tgz` or `zip` file and download to the desired directory.

Step 3: Upack the files. For `tgz` do the following at the command line:

```
gunzip OS-release_number.tgz
tar -xvf OS-release_number.tar
```

Windows users should be able to double click on the file `OS-release_number.zip` and have the directory unpacked.

Step 4: Rename `OS-release_number` to `COIN-OS`. Next follow Steps 2 - 6 outlined in Section 2.1.

Table 1: Tested Platforms for Solvers

	Mac	Linux	Cyg-gcc	Msys-cl	Msys-gcc	MSVS
AMPL-Client	x	x		x		
Cbc	x	x	x	x		
Clp	x	x	x	x		
Cplex	x	x				
DyLP	x	x	x	x		
Ipopt	x	x				
Knitro	x					
Lindo	x	x		x		
SYMPHONY	x	x	x	x		

Table 2: Platform Description

	Operating System	**Compiler**	**Hardware**
Mac	Mac OS X 10.4.9	gcc 4.0.1	Power PC
Linux	Red Hat 3.4.6-8	gcc 3.4.6	Dell Intel 32 bit chip
Cyg-gcc	Windows 2003 Server	gcc 3.4.4	Dell Intel 32 bit chip
Msys-cl	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip
Msys-gcc			
MSVS	Windows XP	Visual Studio 2003	Dell Intel 32 bit chip

2.3 Obtaining and Installing a Visual Studio Project

2.4 Obtaining the Binaries

kipp – discuss with Jun

2.5 Bug Reporting

Bug reporting is done through the project Trac page. This is at `http://projects.coin-or.org/OS`. To report a bug, you must be a registered user. For instructions on how to register go to `http://www.coin-or.org/usingTrac.html` After registering, log in and then file a trouble ticket by going to `http://projects.coin-or.org/OS/newticket`.

2.6 Obtaining the Server Software

2.7 Platforms

The build process described in Section 2.1 has been tested on Linux, Mac OS X, and on Windows using MINGW/MSYS and CYGWIN. The gcc/g++ and Microsoft cl compiler have been tested. A number of solvers have also been tested with the OS library. For a list of tested solvers and platforms see Table 1. More detail on the platforms listed in Table 1 is given in Table 2.

3 The OS Project Components

The directories in the project root are outlined in Figure 1.

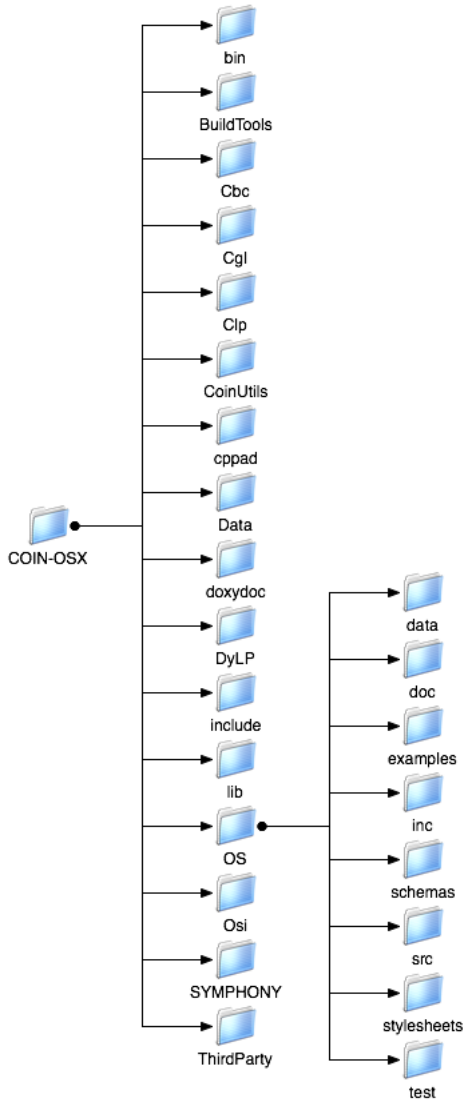

Figure 1: The OS project root directory.

If you download the OS package, you get these additional COIN-OR projects. The links to the project home pages are provided below and give more information on these projects.

- `BuildTools` `projects.coin-or.org\BuildTools`
- `Cbc` `projects.coin-or.org\Cbc`
- `Clp` `projects.coin-or.org\Clp`
- `CppAD` `projects.coin-or.org\CppAD`
- `Dylp` `projects.coin-or.org\Dylp`
- `Osi` `projects.coin-or.org\Osi`
- `SYMPHONY` `projects.coin-or.org\SYMPHONY`

The following directories are also in the project root.

- `bin` after executing `make install` the bin directory will contain `OSSolverService`, `clp`, `cbc`, `cbc-generic` and symphony.
- `Data` this directory contains numerous test problems that are used by some of the COIN-OR project's unitTest.
- `doxydoc` is a folder for documentation
- `include` is a directory for header files. If the user wishes to write code to link against any of the libraries in the `lib` directory, it may be necessary to include these header files.
- `lib` is a directory of libraries. After running `make install` the OS library along with all other COIN-OR libraries are installed in `lib`.
- `ThirdParty` is a directory for third party software. For example, if AMPL related software is used such as `amplClient` is used, then certain AMPL libraries need to be present. This should go into the `ASL` directory in `ThirdParty`.

The directories in the OS directory are outlined in Figure 2.
The OS directories include the following:

- `data` is a directory that holds test problems. These test problems are used by the `unitTest`. Many of these files are also used to illustrate how the `OSSovlerService` works. See Section 7.
- `doc` is the directory with documentation, include this *OS User's Manual.*
- `examples` is a directory with code examples that illustrate various aspects of the OS project. These are described in Section 9.
- `inc` is the directory with the config˙os.h file which has information about which projects are included in the distribution.
- `schemas` is the directory that contains the W3C XSD (see `www.w3c.org`) schemas that are behind the OS standards. These are described in more detail in Section 3.1.

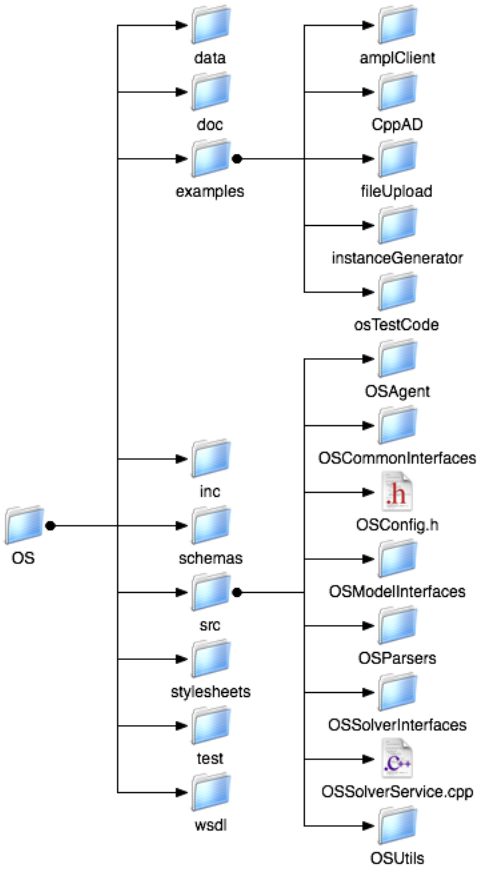

Figure 2: The OS directory.

- `src` is the directory with all of the source code for the OS Library and for the executable `OSSolverService`. The OS Library components are described in Section 4.
- `stylesheets` this directory contains the XSLT stylesheet that is used to transform the solution instance in OSrL format into HTML so that it can be displayed in a browser.
- `test` this directory contains the `unitTest`.
- `wsdl` is a directory of WSDL (Web Services Discovery Language) files. These are used to specify the inputs and outputs for the methods provided by a Web service. The most relevant file for the current version of the OS project is `OShL.wsdl`. This describes the set of inputs and outputs for the methods implemented in the `OSSolverService`. See Section 7.

3.1 Key Protocols

The objective of (OS) is to provide a set of standards for representing optimization instances, results, solver options, and communication between clients and solvers in a distributed environment using Web Services. These standards are specified by W3C XSD schemas. The schemas for the OS project are contained in the `schemas` folder under the `OS` root. There are numerous schemas in this directory that are part of the OS standard. For a full description of all the schemas see Ma [1]. We briefly discuss the standards most relevant to the current version of the OS project.

OSiL (Optimization Services instance Language): an XML-based language for representing instances of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs.

OSiL, stores optimization problem instances as XML files. Consider the following problem instance that is a modification of an example of Rosenbrock [2]:

$$\begin{aligned} \text{Minimize} \quad & (1-x_0)^2 + 100(x_1 - x_0^2)^2 + 9x_1 && (1)\\ \text{Subject to} \quad & x_0 + 10.5x_0^2 + 11.7x_1^2 + 3x_0x_1 \leq 25 && (2)\\ & \ln(x_0x_1) + 7.5x_0 + 5.25x_1 \geq 10 && (3)\\ & x_0, x_1 \geq 0 && (4) \end{aligned}$$

There are two continuous variables, x_0 and x_1, in this instance, each with a lower bound of 0. Figure 3 shows how we represent this information in an XML-based OSiL file. Like all XML files, this is a text file that contains both *markup* and *data*. In this case there are two types of markup, *elements* (or *tags*) and *attributes* that describe the elements. Specifically, there are a `<variables>` element and two `<var>` elements. Each `<var>` element has attributes `lb`, `name`, and `type` that describe properties of a decision variable: its lower bound, "name", and domain type.

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

Figure 3: The `<variables>` element for the example (1)–(4).

To be useful for communication between solvers and modeling languages, OSiL instance files must conform to a standard. An XML-based representation standard is imposed through the use of a *W3C XML Schema.* The W3C, or World Wide Web Consortium (`www.w3.org`), promotes standards for the evolution of the web and for interoperability between web products. XML Schema (`www.w3.org/XML/Schema`) is one such standard. A schema specifies the elements and attributes that define a specific XML vocabulary. The W3C XML Schema is thus a schema for schemas; it specifies the elements and attributes for a schema that in turn specifies elements and attributes for an XML vocabulary such as OSiL. An XML file that conforms to a schema is called *valid* for that schema.

By analogy to object-oriented programming, a schema is akin to a header file in C++ that defines the members and methods in a class. Just as a class in C++ very explicitly describes member and method names and properties, a schema explicitly describes element and attribute names and properties.

Figure 4 is a piece of our schema for OSiL. In W3C XML Schema jargon, it defines a *complexType,* whose purpose is to specify elements and attributes that are allowed to appear in a valid XML instance file such as the one excerpted in Figure 3. In particular, Figure 4 defines the complexType named `Variables`, which comprises an element named `<var>` and

```
<xs:complexType name="Variables">
    <xs:sequence>
        <xs:element name="var" type="Variable" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="numberOfVariables"
            type="xs:positiveInteger" use="required"/>
</xs:complexType>
```

Figure 4: The `Variables` complexType in the OSiL schema.

```
<xs:complexType name="Variable">
    <xs:attribute name="name" type="xs:string" use="optional"/>
    <xs:attribute name="init" type="xs:string" use="optional"/>
    <xs:attribute name="type" use="optional" default="C">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:enumeration value="C"/>
                <xs:enumeration value="B"/>
                <xs:enumeration value="I"/>
                <xs:enumeration value="S"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="lb" type="xs:double" use="optional" default="0"/>
    <xs:attribute name="ub" type="xs:double" use="optional" default="INF"/>
</xs:complexType>
```

Figure 5: The `Variable` complexType in the OSiL schema.

an attribute named `numberOfVariables`. The `numberOfVariables` attribute is of a standard type `positiveInteger`, whereas the `<var>` element is a user-defined complexType named `Variable`. Thus the complexType `Variables` contains a sequence of `<var>` elements that are of complexType `Variable`. OSiL's schema must also provide a specification for the `Variable` complexType, which is shown in Figure 5.

In OSiL the linear part of the problem is stored in the `<linearConstraintCoefficients>` element, which stores the coefficient matrix using three arrays as proposed in the earlier LPFML schema [**?**]. There is a child element of `<linearConstraintCoefficients>` to represent each array: `<value>` for an array of nonzero coefficients, `<rowIdx>` or `<colIdx>` for a corresponding array of row indices or column indices, and `<start>` for an array that indicates where each row or column begins in the previous two arrays.

```
<linearConstraintCoefficients numberOfValues="3">
    <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.</el><el>7.5</el><el>5.25</el>
    </value>
</linearConstraintCoefficients>
```

Figure 6: The `<linearConstraintCoefficients>` element for constraints (2) and (3).

The quadratic part of the problem is represented as follows.

```
<quadraticCoefficients numberOfQuadraticTerms="3">
     <qTerm idx="0" idxOne="0" idxTwo="0" coef="10.5"/>
     <qTerm idx="0" idxOne="1" idxTwo="1" coef="11.7"/>
     <qTerm idx="0" idxOne="0" idxTwo="1" coef="3."/>
</quadraticCoefficients>
```

Figure 7: The `<quadraticCoefficients>` element for constraint (2).

The nonlinear part of the problem is given in Figure 8.

The complete OSiL representation is given in the Appendix.

OSrL (Optimization Services result Language): an XML-based language for representing the solution of large-scale optimization problems including linear programs, mixed-integer programs, quadratic programs, and very general nonlinear programs. As example solution (for the problem given in (1)–(4)) in OSrL format is given below.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<nl idx="-1">
     <plus>
          <power>
               <minus>
                    <number value="1.0"/>
                    <variable coef="1.0" idx="0"/>
               </minus>
               <number value="2.0"/>
          </power>
          <times>
               <power>
                    <minus>
                         <variable coef="1.0" idx="0"/>
                         <power>
                              <variable coef="1.0" idx="1"/>
                              <number value="2.0"/>
                         </power>
                    </minus>
                    <number value="2.0"/>
               </power>
               <number value="100"/>
          </times>
     </plus>
</nl>
```

Figure 8: The <nl> element for the nonlinear part of the objective (1).

```
<?xml-stylesheet type = "text/xsl"
    href = "/Users/kmartin/Documents/files/code/cpp/OScpp/COIN-OSX/OS/stylesheets/OSrL.xsl
<osrl xmlns="os.optimizationservices.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
    xsi:schemaLocation="os.optimizationservices.org http://www.optimizationservices.org/sc
    <resultHeader>
        <generalStatus type="success"/>
        <serviceName>Solved using a LINDO service</serviceName>
        <instanceName>Modified Rosenbrock</instanceName>
    </resultHeader>
    <resultData>
        <optimization numberOfSolutions="1" numberOfVariables="2" numberOfConstraints="2"
            numberOfObjectives="1">
            <solution objectiveIdx="-1">
                <status type="optimal"/>
                <variables>
                    <values>
                        <var idx="0">0.87243</var>
                        <var idx="1">0.741417</var>
                    </values>
                    <other name="reduced costs" description="the variable reduced costs">
                        <var idx="0">-4.06909e-08</var>
                        <var idx="1">0</var>
```

```
            </other>
        </variables>
        <objectives>
            <values>
                <obj idx="-1">6.7279</obj>
            </values>
        </objectives>
        <constraints>
            <dualValues>
                <con idx="0">0</con>
                <con idx="1">0.766294</con>
            </dualValues>
        </constraints>
    </solution>
</optimization>
```

OSoL (Optimization Services option Language): an XML-based language for representing options that get passed to an optimization solver.

OSnL (Optimization Services nonlinear Language): The OSnL schema is imported by the OSiL schema and is used to represent the nonlinear part of an optimization instane. This is explained in greater detail in Section 4.2.2. Also refer to Figue 8 for an illustration of elements from the OSnL standard.

OSpL (Optimization Services process Language): is a standard for dynamic process information that is kept by the Optimization Services registry. It is the result of a `knock` operation. See the example given in Section 7.3.5.

4 The OS Library Components

4.1 OSAgent

The `OSAgent` part of the library is used to facilitate communication with remote solvers. It is not used if the solver is invoked locally (i.e. on the same machine).

4.2 OSCommonInterfaces

Mention the OSnL node class. List which NL operators can be used.

4.2.1 The OSInstance Class

4.2.2 The OSExpressionTree OSnLNode Classes

The `OSExpressionTree` class provides the in-memory representation of the nonlinear terms. Our design goal is to allow for efficient parsing of OSiL instances, while providing an API that meets the needs of diverse solvers. Conceptually, any nonlinear expression in the objective or constraints is represented by a tree. The expression tree for the nonlinear part of the objective function (1), for example, has the form illustrated in Figure 9. The choice of a data structure to store such a tree

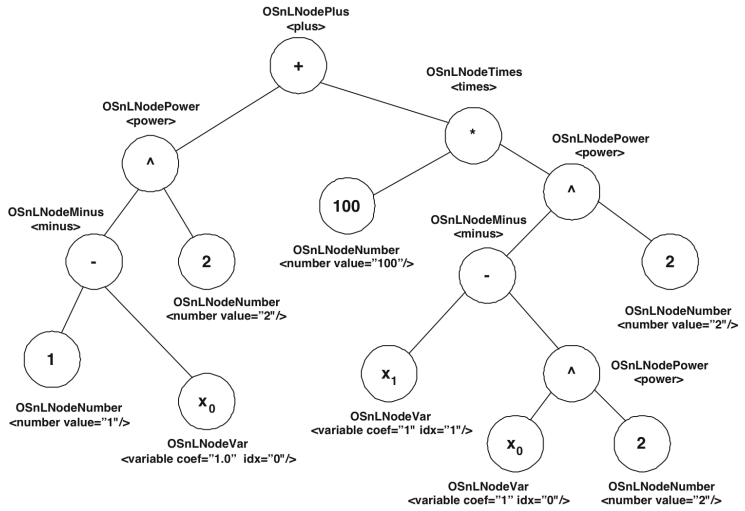

Figure 9: Conceptual expression tree for the nonlinear part of the objective (1).

— along with the associated methods of an API — is a key aspect in the design of the `OSInstance` class.

A base abstract class `OSnLNode` is defined and all of an OSiL file's operator and operand elements used in defining a nonlinear expression are extensions of the base element type `OSnLNode`. There is an element type `OSnLNodePlus`, for example, that extends `OSnLNode`; then in an OSiL instance file, there are `<plus>` elements that are of type `OSnLNodePlus`. Each `OSExpressionTree` object contains a pointer to an `OSnLNode` object that is the root of the corresponding expression tree. To every element that extends the `OSnLNode` type in an OSiL instance file, there corresponds a class that derives from the `OSnLNode` class in an `OSInstance` data structure. Thus we can construct an expression tree of homogenous nodes, and methods that operate on the expression tree to calculate function values, derivatives, postfix notation, and the like do not require switches or complicated logic.

```
double OSnLNodePlus::calculateFunction(double *x){
   m_dFunctionValue =
      m_mChildren[0]->calculateFunction(x) +
      m_mChildren[1]->calculateFunction(x);
   return m_dFunctionValue;
} //calculateFunction
```

Figure 10: The function calculation method for the "plus" node class with polymorphism

The `OSInstance` class has a variety of `calculate()` methods, based on two pure virtual functions in the `OSInstance` class. The first of these, `calculateFunction()`, takes an array of `double` values corresponding to decision variables, and evaluates the expression tree for those values. Every class that extends `OSnLNode` must implement this method. As an example, the `calculateFunction` method for the `OSnLNodePlus` class is shown in Figure 10. Because the OSiL instance file must be validated against its schema, and in the schema each `<OSnLNodePlus>` element is specified to have exactly two child elements, this `calculateFunction` method can assume that there are exactly two children of the node that it is operating on. Thus through the use of polymorphism and

recursion the need for switches like those in Figure **??** is eliminated. This design makes adding new operator elements easy; it is simply a matter of adding a new class and implementing the `calculateFunction()` method for it.

The following `OSnLNode` classes are currently supported.

- OSnLNodeVariable
- OSnLNodeTimes
- OSnLNodePlus
- OSnLNodeSum
- OSnLNodeMinus
- OSnLNodeNegate
- OSnLNodeDivide
- OSnLNodePower
- OSnLNodeProduct
- OSnLNodeLn
- OSnLNodeSqrt
- OSnLNodeSquare
- OSnLNodeSin
- OSnLNodeCos
- OSnLNodeExp
- OSnLNodeif
- OSnLNodeAbs
- OSnLNodeMax
- OSnLNodeMin
- OSnLNodeE
- OSnLNodePI
- OSnLNodeAllDiff

4.3 OSModelInterfaces

4.4 OSParsers

4.5 OSSolverInterfaces

The `OSSolverInterfaces` library is designed to facilitate linking the OS library with various solver APIs. We first describe how to take a problem instance in OSiL format and connect to a solver that has a COIN-OR OSI interface. See the OSI project `www.projects.coin-or.org/Osi`. We then describe hooking to the COIN-OR nonlinear code `Ipopt`. See `www.projects.coin-or.org/Ipopt`. Finally we describe hooking to two commercial solvers KNITRO and LINDO.

The OS library has been tested with the following solvers using the Osi Interface.

- Cbc
- Clp
- Cplex
- DyLP
- Glpk
- SYMPHONY

In the `OSSolverInterfaces` library there is an abstract class `DefaultSolver` that has the following key members:

```
std::string osil;
std::string osol;
std::string osrl;
OSInstance *osinstance;
OSResult  *osresult;
```

and the pure virtual function

```
virtual void solve() = 0 ;
```

In order to use a solver through the COIN-OR `Osi` interface it is necessary to an object in the `CoinSolver` class which inherits from the `DefaultSolver` class and implements the appropriate `solve()` function. We illustrate with the Clp solver.

```
DefaultSolver *solver  = NULL;
solver = new CoinSolver();
solver->m_sSolverName = "clp";
```

Assume that the data file containing the problem has been read into the string `osil` and the solver options are in the string `osol`. Then the `Clp` solver is invoked as follows.

```
solver->osil = osil;
solver->osol = osol;
solver->solve();
```

Finally, get the solution in OSrL format as follows

```
cout << solver->osrl << endl;
```

Even though LINDO and KNITRO are commercial solvers and do not have a COIN-OR Osi interface these solvers are used in exactly the same manner as a COIN-OR solver. For example, to invoke the LINDO solver we do the following.

```
solver = new LindoSolver();
```

Similarly for KNITRO and Ipopt. In the case of the KNITRO, the KnitroSolver class inherits from both DefaultSolver class and the KNITRO NlpProblemDef class. See Kipp--putinKnitromanuallink for more information on the KNITRO solver C++ implementation and the NlpProblemDef class. Similarly, for Ipopt the IpoptSolver class inherits from both the DefaultSolver class and the Ipopt TNLP class. See Kipp--putinIpoptmanuallink for more information on the Ipopt solver C++ implementation and the TNLP calss.

In the examples above the problem instance was assumed to be read from a file into the string osil and then into the class member solver->osil. However, everything can be done entirely in memory. For example, it is possible to use the OSInstance class to create an in-memory problem representation and give this representation directly to a solver class that inherits from DefaultSolver. The class member to use is osinstance. This is illustrated in the example given in Section 9.4.

4.6 OSUtils

5 The OSInstance API

5.1 Get Methods

Don't forget to mention getting prefix an postfix. Illustrate some of this like in the unitTest.

5.2 Set Methods

5.3 Calculate Methods

6 Hooking to An Algorithmic Differentiation Package

7 The OSSolverService

The OSSolverService is a command line executable designed to pass problem instances in either OSiL, AMPL nl, or MPS format to solvers and get the optimization result back to be displayed either to standard output or a specified browser. The OSSovlerService can be used to invoke a solver locally or on a remote server. It can work either synchronously or asynchronously.

7.1 OSSolverService Input Parameters

At present, the OSSolverService takes the following parameters. The order of the parameters is irrelevant. Not all the parameters are required. However, if the solve or send service methods are invoked a problem instance location must be specified.

-osil xxx.osil this is the name of the file that contains the optimization instance in OSiL format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. If this option is not specified then the instance location must be specified in the OSoL solver options file.

-osol xxx.osol this is the name of the file that contains the solver options. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. It is not necessary to specify this option.

-osrl xxx.osrl this is the name of the file that contains the solver solution. A valid file path must be given on the machine that is running `OSSolverService`. It is not necessary to specify this option.

-serviceLocation is the URL of the solver service. This is not required, and if not specified it is assumed that the problem is solved locally.

-serviceMethod method this is the solver service required. The options are `solve`, `send`,`kill`,`knock`, `getJobID`, and `retrieve`. The use of these options is illustrated in the examples below. This option is not required, and the default value is `solve.`

-mps xxx.mps this is the name of the mps file if the problem instance is in mps format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. The default file format is OSiL so this option is not required.

-nl xxx.nl this is the name of the AMPL nl file if the problem instance is in AMPL nl format. It is assumed that this file is available in a directory on the machine that is running `OSSolverService`. The default file format is OSiL so this option is not required.

-solver solverName Possible values for default OS installation are tt clp (COIN-OR Clp), `cbc` (COIN-OR Cbc), `dylp` (COIN-OR DyLP), and `symphony` (COIN-OR SYMPHONY). Other solvers supported (if the necessary libraries are present) are `cplex` (Cplex through COIN-OR Osi), `glpk` (glpk through COIN-OR Osi), `ipopt` (COIN-OR Ipopt), `knitro` (Knitro), and `lindo LINDO`. If no value is specified for this parameter, then `cbc` is the default value of this parameter if the the `solve` or `send` service methods are used.

-browser browserName this paramater is a path to the browser on the local machine. If this optional parameter is specified then the solver result in OSrL format is transformed using XSLT into HTML and displayed in the browser.

-config pathToConfigureFile this parameter specifies a path on the local machine to a text file containing values for the input parameters. This is convenient for the user not wishing to constantly retype parameter values.

The input parameters to the `OSSolverService` may be given entirely in the command line or in a configuration file. We first illustrate giving all the parameters in the command line. The following command will invoke the `Clp` solver on the local machine to solve the problem instance `parincLinear.osil`.

```
OSSolverService -solver clp -osil ../data/parincLinear.osil
```

Alternatively, these parameters can be put into a configuration file. Assume that the configuration file of interest is `testlocalclp.config`. It would contain the two lines of information

```
-osil ../data/parincLinear.osil
-solver clp
```

Then the command line is

```
OSSolverService -config ../data/testlocalclp.config
```

Some Rules:

1. When using the `send()` or `solve()` methods a problem instance file location *must* be specified either at the command line, in the configuration file, or in the `<instanceLocation>` element in the OSoL options file file.
2. The default `serviceMethod` is `solve` if another service method is not specified. The service method cannot be specified in the OSoL options file.
3. If the `solver` option is not specified, the COIN-OR solver `Cbc` is the default solver used. In this case an error is thrown if the problem instance has quadratic or other nonlinear terms.
4. If the options `send`, `kill`, `knock`, `getJobID`, or `retrieve` are specified, a `serviceLocation` must be specified.

Parameters specified in the configure file are overridden by parameters specified at the command line. This is convenient if a user has a base configure file and wishes to override only a few options. For example,

```
OSSolverService -config ../data/testlocalclp.config -solver lindo
```

or

```
OSSolverService -solver lindo -config ../data/testlocalclp.config
```

will result in the LINDO solver being used even though Clp is specified in the `testlocalclp` configure file.

7.2 Solving Problems Locally

Generally, when solving a problem locally the user will use the `solve` service method. The `solve` method is invoked synchronously and waits for the solver to return the result. This is illustrated in Figure 12. As illustrated, the `OSSolverService` reads a file on the hard drive with the optimization instance, usually in OSiL format. The optimization instance is parsed into a string which is passed to the `OSLibrary` which is linked with various solvers. The result of the optimization is passed back to the `OSSolverService` as a string in OSrL format.

Here is an example of using a configure file, `testlocal.config`, to invoke `Ipopt` locally using the `solve` command.

```
-osil ../data/parincQuadratic.osil
-solver ipopt
-serviceMethod solve
-browser /Applications/Firefox.app/Contents/MacOS/firefox
-osrl /Users/kmartin/temp/test.osrl
```

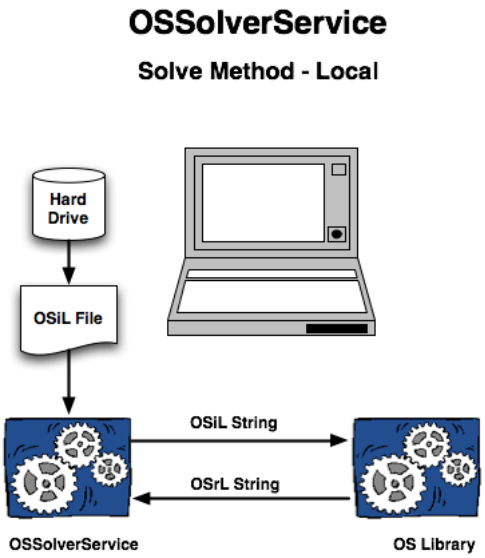

Figure 11: A local call to `solve`.

The first line of `testlocal.config` gives the local location of the OSiL file, `parincQuadratic.osil`, that contains the problem instance. The second parameter, `-solver ipopt`, is the solver to be invoked, in this case COIN-OR Ipopt. The third parameter `-serviceMethod solve` is not really needed, but included only for illustration. The default solver service is `solve`. The fourth parameter is the location of the browser on the local machine. It will read the OSrL file on the local machine using the path specified by the value of the `osrl` parameter, in this case `/Users/kmartin/temp/test.osrl`.

Parameters may also be contained in an XML-file in OSoL format. In the configuration file `testlocalosol.config` we illustrate specifying the instance location in an OSoL file.

```
-osol ../data/demo.osol
-solver clp
```

The file `demo.osol` is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <instanceLocation locationType="local">
            ../data/parincLinear.osil
        </instanceLocation>
    </general>
</osol>
```

7.3 Solving Problems Remotely with Web Services

In many cases the client machine may be a "weak client" and using a more powerful machine to solve a hard optimization instance is required. Indeed, one of the major purposes of Optimization Services is to facilitate optimization in a distributed environment. We now provide examples that illustrate using the `OSSolverService` executable to call a remote solver service. By remote solver

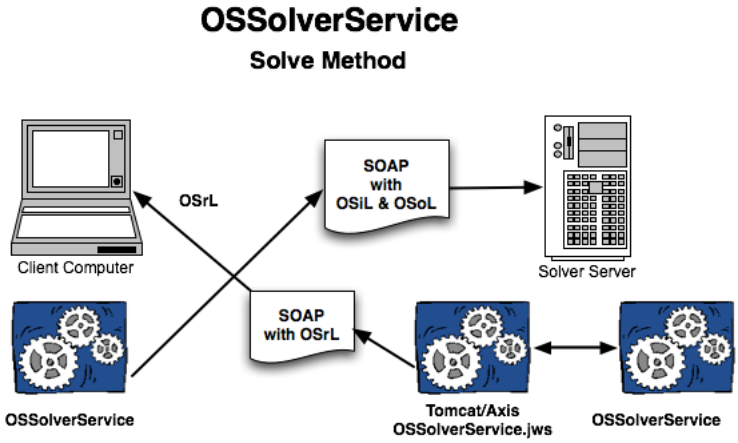

Figure 12: A remote call to `solve`.

service we mean a solver service that is called using Web Services. The OS implementation of the solver service uses Apache Tomcat. See `tomcat.apache.org`. The Web Service running on the server is a Java program based on Apache Axis. See `ws.apache.org/axis`. This is described in greater detail in Section 8. This Web Service is called `OSSolverService.jws`. It is not necessary to use the Tomcat/Axis combination.

See Figure 12 for an illustration of this process. The client machine uses `OSSolverService` executable to call one of the six service methods, e.g. `solve`. The information such as the problem instance in OSiL format and solver options in OSoL format are packaged into a SOAP envelope and sent to the server. The server is running the Java Web Service `OSSolverService.jws`. This Java program running in the Tomcat Java Servlet container implements the six service methods. If a `solve` or `send` request is sent to the server from the client, an optimization problem must be solved. The Java solver service solves the optimization instance by calling the OSSolverService on the server. So there is an OSSolverService on the client that calls the Web Service `OSSolverService.jws` that in turn calls the executable `OSSovlerService` on the server. The Java solver service passes options to the local `OSSolverService` such as where the OSiL file is located and where to write the solution result.

In the following sections we illustrate each of the six service methods.

7.3.1 The `solve` Service Method

First we illustrate a simple call to `OSSolverService.jws` and request a solution using the COIN-OR `Clp` solver. The call on the client machine is

```
OSSolverService -config ../data/testremote.config
```

where the `testremote.config` file is

```
-osil ../data/parincLinear.osil
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
```

No solver is specified so by default the `Cbc` solver will be used on the server.

Now use an OSoL options file

```
OSSolverService -osol ../data/remoteSolve1.osol -osil ../data/parincLinear.osil
```

where `remoteSolve1.osol` is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
    </general>
    <optimization>
     <other name="os_solver">clp</other>
    </optimization>
</osol>
```

In this case we specify a sover to use, name `Clp`.

Next we illustrate a call to the remote SolverService and specify an OSiL instance that is on the remote machine.

```
OSSolverService -osol ../data/remoteSolve2.osol
```

where the `remoteSolve2.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
         <instanceLocation locationType="local">
     /home/kmartin/files/code/OSRepository/linear/continuous/pilot.osil
     </instanceLocation>
    </general>
    <optimization>
        <other name="os_solver">clp</other>
    </optimization>
</osol>
```

If we were to change to the `locationType` attribute in the `<instanceLocation>` element to `http` then we could specify the intance location to on yet another machine. This is illustrated below for `remoteSovle3.osol`. The scenario is depicted in Figure 13. The OSiL string passed from the client to the solver service is empty. However, the OSoL element `<instanceLocation>` has an attribute `locationType` equal to `http`. In this case, the text of the `<instanceLoction>` element contains the URL of a third machine which has the problem intance `parincLinear.osil`. The solver service will contact the machine with URL `gsbkip.chicagogsb.edu` and download this test problem.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <serviceURI>http://128.135.130.17:8080/os/OSSolverService.jws</serviceURI>
         <instanceLocation locationType="http">
     http://gsbkip.chicagogsb.edu/testproblems/parincLinear.osil
     </instanceLocation>
```

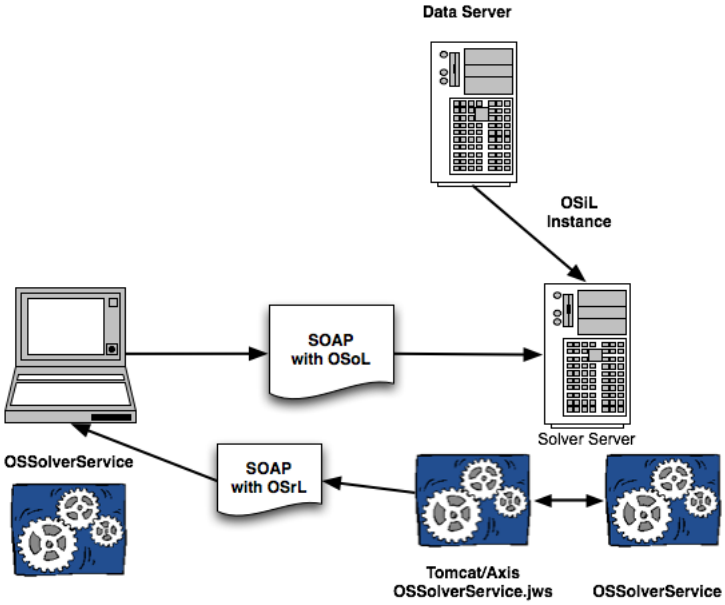

Figure 13: Downloading the instance from a remote source.

```
    </general>
    <optimization>
        <other name="os_solver">clp</other>
    </optimization>
</osol>
```

7.3.2 The `send` Service Method

When the `solve` service method is used, the `OSSolverService` does not finish execution until the solution is returned from the remote solver service. The `solve` method communicates synchronously with the remote solver service. This may not be desirable for large problems when the user does not want to wait for a response. The `send` service method should be used when asynchronous communication is desired. When the send method is used the instance is communicated to the remote service and the `OSSolverService` terminates after submission. An example of this is

```
OSSolverService -config ../data/testremoteSend.config
```

where the `testremoteSend.config` file is

```
-nl ../data/hs71.nl
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod send
```

In this example the COIN-OR `Ipopt` solver is specified. The input file `hs71.nl` is in AMPL format. Before sending this to the remote solver service the `OSSolverService` executable converts the nl format into the OSiL XML format and packages this into the SOAP envelope used by Web Services.

Since the `send` method involves asynchronous communication the remote solver service must keep track of jobs. The send methd requires a `JobID`. In the above example no `JobID` was specified.

When no JobID is specified the OSSolverService method first invokes the getJobID service method to get a JobID and then puts this information into a created OSoL file and send the information to the server. More information on the getJobID service method is provided in Section 7.3.4. The OSSolverService prints the OSoL file to standard output before termination. This is illustrated below,

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>
        gsbrkm4__127.0.0.1__2007-06-16T15.46.46.075-05.00149771253
        </jobID>
    </general>
    <optimization>
        <other name="os_solver">ipopt</other>
    </optimization>
</osol>
```

The JobID is one that is randomly generated by the server and passed back to the OSSolverService. The user can also provide a JobID in their OSoL file. For example, below is a user-provided OSoL file that could be specified in a configuration file or on the command line.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
    </general>
    <optimization>
        <other name="os_solver">ipopt</other>
    </optimization>
</osol>
```

In order to be of any use, it is necesary to get the result of the optimization. This is described in Section 7.3.3. Before proceeding to this section, we describe two ways for knowing when the optimization is complete. One feature of the standard OS remote SolverService is the ability to send an email when the job is complete. Below is an example of the OSoL that uses the email feature.

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
        <contact transportType="smtp">
            kipp.martin@chicagogsb.edu
        </contact>
    </general>
    <optimization>
        <other name="os_solver">lindo</other>
    </optimization>
</osol>
```

The remote Solver Service will send an email to the above address when the job is complete. A second option for knowing when a job is complete is to use the knock method.

Note that in all of these examples we provided a value for the `name` attribute in the `<other>` element. The remote solver service will use Cbc if another solver is not specified.

7.3.3 The `retrieve` Service Method

The `retrieve` has a single string argument which is an OSoL instance. Here is an example of using the `retrieve` method with `OSSolverService`.

```
OSSolverService -config ../data/testremoteRetrieve.config
```

The `testremoteRetrieve.config` file is

```
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-osol ../data/retrieve.osol
-serviceMethod retrieve
-osrl /home/kmartin/temp/test.osrl
```

and the `retrieve.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

The OSoL file `retrieve.osol` contains a tag `<jobID>` that is communicated to the remote service. The remove service locates the result returns it as a string. The string that is returned is an OSrL instance.

7.3.4 The `getJobID` Service Method

Before submitting a job with the `send` method a `JobID` is required. The `OSSolverService` can get a `JobID` as follows

```
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod getJobID
```

Note that no OSoL input file is specified. In this case, the `OSSolverService` sends an empty string. A string is returned with the `JobID`. This `JobID` is then put into a `<jobID>` element in an OSoL string that would be used by the `send` method.

7.3.5 The `knock` Service Method

The OSSolverService terminates after executing the `send` method. Therefore, it is necessary to know when the job is completed on the remote server. One way is to include an email address in the `<contact>` element with the attribute `transportType` set to `smtp`. This was illustrated in Section 7.3.1. A second way to check on the status of a job is to use the `knock` service method. For example, assume a user wants to know if the job with `JobID 123456abcd` is complete. A user would make the request

```
OSSolverService -config ../data/testRemoteKnock.config
```

where the `testRemoteKnock.config` file is

```
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-osplInput ../data/demo.ospl
-osol ../data/retrieve.osol
-serviceMethod knock
```

the `demo.ospl` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<ospl xmlns="os.optimizationservices.org">
<processHeader>
<request action="getAll"/>
</processHeader>
<processData/>
</ospl>
```

and the `retrieve.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
  <general>
  <jobID>123456abcd</jobID>
</general>
</osol>
```

The result of this request is a string in OSrL format. Part of the return format is illustrated below.

```
<jobs>
    <job jobID="123456abcd">
        <state>finished</state>
        <serviceURI>http://128.135.130.17:8080/ipopt/IPOPTSolverService.jws</serviceURI>
        <submitTime>2007-06-16T14:57:36.678-05:00</submitTime>
        <startTime>2007-06-16T14:57:36.678-05:00</startTime>
        <endTime>2007-06-16T14:57:39.404-05:00</endTime>
        <duration>2.726</duration>
 </job>
</jobs>
```

Notice the `<state>` element indicating that the job is finished.

When making a `knock` request, the OSoL string can be empty. In this example, if the OSoL string had been empty the status of all jobs kept in the file ospl.xml is reported. In our default solver service implementation, there is a configuration file `OSParameter` that has a parameter `MAX_JOBIDS_TO_KEEP` . The current default setting is 100. In a large-scale or commercial implementation it might be wise to keep problem results and statistics in a database. Also, there are values other than `getAll` for the OSpL `action` attribute in the `<request>` tag. For example, the `action` can be set to a value of `ping` if the user just wants to check if the remote solver service is up and running.

7.3.6 The `kill` Service Method

If the user submits a job that is taking too long or is a mistake it is possible to kill the job on the remote server using the `kill` service method. For example to kill job `123456abcd` . At the command line type

```
OSSolverService -config  ../data/kill.config
```

where the configure file `kill.config` is

```
-osol ../data/kill.osol
-serviceLocation http://128.135.130.17:8080/os/OSSolverService.jws
-serviceMethod kill
```

and the `kill.osol` file is

```
<?xml version="1.0" encoding="UTF-8"?>
<osol xmlns="os.optimizationservices.org">
    <general>
        <jobID>123456abcd</jobID>
    </general>
</osol>
```

7.3.7 Summary

Below is a summary of the inputs and outputs of the six service methods. See also Figures 14 and 15.

- `solve(osil, osol)`:
  - Inputs: a string with the instance in OSiL format and a string with the solver options in OSoL format
  - Returns: a string with the solver solution in OSrL format
  - Synchronous call, blocking request/response
- `send(osil, osol)`
  - Inputs: a string with the instance in OSiL format and a string with the solver options in OSoL format
  - Returns: a boolean, true if the problem was successfully submitted, false otherwise
  - Has the same signature as `solve`
  - Asynchronous (server side), non-blocking call
  - The `osol` string should have a `JobID` in the `<jobID>` element
- `getJobID( osol)`
  - Inputs: a string with the solver options in OSoL format (in this case, the string may be empty because no options are required to get the JobID)
  - Returns: a string which is the unique job id generated by the solver service

  - Used to maintain session and state on a distributed system

- `knock(ospl, osol)`
  - Inputs: a string in OSpL format and a string with the solver options in OSoL format
  - Returns: process and job status information from the remote server in OSpL format

- `retrieve( osol)`
  - Inputs: a string with the solver options in OSoL format
  - Returns: a string with the solver solution in OSrL format
  - The `osol` string should have a `JobID` in the `<jobID>` element

- `kill( osol)`
  - Inputs: a string with the solver options in OSoL format
  - Returns: process and job status information from the remote server in OSpL format
  - Critical in long running optimization jobs

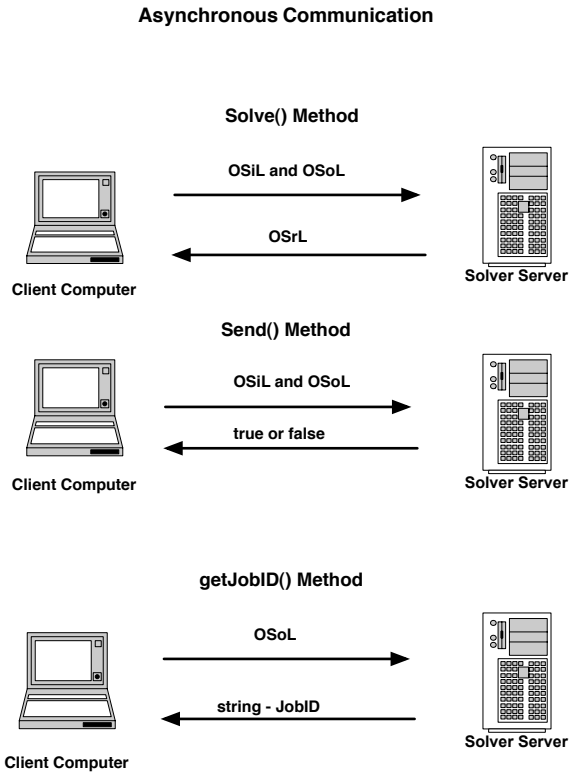

Figure 14: Input and output for `solve`, `send`, and `getJobID` methods.

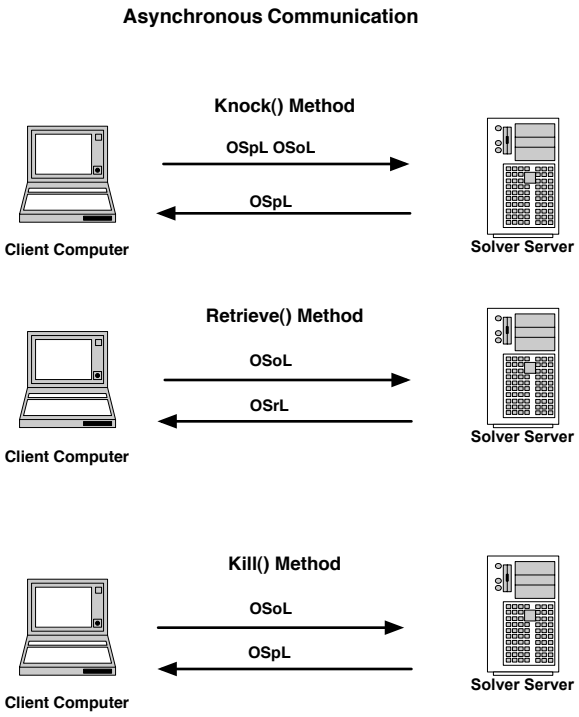

Figure 15: Input and output for `knock`, `retrieve`, and `kill` methods.

8 Setting up a Solver Service with Tomcat

9 Examples

9.1 AMPL Client: Hooking AMPL to Solvers

The `amplClient` executable is designed to work with the AMPL program. See `www.ampl.com`. The `amplClient` acts like an AMPL "solver." The `amplClient` is linked with the OS library and can be used to solve problems either remotely. In both cases the `amplClient` uses the `OSnl2osil` class to convert the AMPL generated nl file (which represents the problem instance) into the corresponding instance representation in the OSiL format.

For example, assume that there is a problem instance, `hs71.mod` in AMPL model format. To solve this problem locally by calling the `amplClient` from AMPL first start AMPL and then execute the following commands. In this case we are assuming that the local solver used is `Ipopt.`

```
# take in problem 71 in Hock and Schittkowski
# assume the problem is in the AMPL directory
model hs71.mod;
# tell AMPL that the solve is amplClient
option solver amplClient;
# now tell amplClient to use Ipopt
option amplClient_options "solver ipopt";
# the name of the nl file (this is optional)
write gtestfile;
# now solve the problem
```

```
solve;
```

This will invoke `Ipopt` locally and the result in OSrL format will be displayed on the screen. In order to call a remote solver service, after the command

```
option amplClient_options "solver ipopt";
```

provide an option which has the address of the remote solver service.

```
option ipopt_options "http://128.135.130.17:8080/os/OSSolverService.jws";
```

9.2 CppAD: Using the CppAD Algorithmic Differentiation Package

9.3 File Upload: Using a File Upload Package

9.4 Instance Generator: Using the OSInstance API to Generate Instances

10 References

Kipp – put in some links to OSiL paper and INFORMS talks.

11 Appendix

This is the complete OSiL representation for the optimization problem given in (1)–(4).

```
<?xml version="1.0" encoding="UTF-8"?>
<osil xmlns="os.optimizationservices.org">
     <instanceHeader>
          <name>Modified Rosenbrock</name>
          <source>Computing Journal 3:175-184, 1960</source>
          <description>Rosenbrock problem with constraints</description>
     </instanceHeader>
     <instanceData>
          <variables numberOfVariables="2">
               <var lb="0" name="x0" type="C"/>
               <var lb="0" name="x1" type="C"/>
          </variables>
          <objectives numberOfObjectives="1">
               <obj maxOrMin="min" name="minCost" numberOfObjCoef="1">
                    <coef idx="1">9.0</coef>
               </obj>
          </objectives>
          <constraints numberOfConstraints="2">
               <con ub="25.0"/>
               <con lb="10.0"/>
          </constraints>
```

```
<linearConstraintCoefficients numberOfValues="3">
     <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.</el><el>7.5</el><el>5.25</el>
     </value>
</linearConstraintCoefficients>
<quadraticCoefficients numberOfQuadraticTerms="3">
     <qTerm idx="0" idxOne="0" idxTwo="0" coef="10.5"/>
     <qTerm idx="0" idxOne="1" idxTwo="1" coef="11.7"/>
     <qTerm idx="0" idxOne="0" idxTwo="1" coef="3."/>
</quadraticCoefficients>
```

```
        <nonlinearExpressions numberOfNonlinearExpressions="2">
             <nl idx="-1">
                  <plus>
                       <power>
                            <minus>
                                 <number type="real" value="1.0"/>
                                 <variable coef="1.0" idx="0"/>
                            </minus>
                            <number type="real" value="2.0"/>
                       </power>
                       <times>
                            <power>
                                 <minus>
                                      <variable coef="1.0" idx="0"/>
                                      <power>
                                           <variable coef="1.0" idx="1"/>
                                           <number type="real" value="2.0"/>
                                      </power>
                                 </minus>
                                 <number type="real" value="2.0"/>
                            </power>
                            <number type="real" value="100"/>
                       </times>
                  </plus>
             </nl>
             <nl idx="1">
                  <ln>
                       <times>
                            <variable coef="1.0" idx="0"/>
                            <variable coef="1.0" idx="1"/>
                       </times>
                  </ln>
             </nl>
        </nonlinearExpressions>
   </instanceData>
</osil>
```

References

[1] J. Ma. Optimization services (OS), a general framework for optimization modeling systems, 2005. Ph.D. Dissertation, Department of Industrial Engineering & Management Sciences, North-

western University, Evanston, IL.

[2] H.H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Comp. J.*, 3:175–184, 1960.